BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

TESIS PROFESIONAL

Para Obtener el Título de:

ING. EN CIENCIAS DE LA COMPUTACIÓN

Reconstrucción 3D de Estructuras Dendríticas usando un Microscopio de Luz Clara

Presenta : ALFONSO BALANDRA ANTELIS

Asesor : DR. MANUEL MARTIN ORTIZ

Octubre del 2009

Puebla, Puebla.

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Reconstrucción 3D de Estructuras Dendríticas usando un Microscopio de Luz Clara

ALFONSO BALANDRA ANTELIS

Tesis presentada a la Comisión integrada por:

DR. MANUEL MARTIN ORTIZ
DR. GONZALO FLORES ALVAREZ
DR. ABRAHAM SANCHEZ LÓPEZ
M.C. HILDA CASTILLO ZACATELCO

Para completar las exigencias del grado de
Ingeniero en Ciencias de la Computación

Octubre del 2009
Puebla, Puebla.

A mi familia por todos estos años de paciencia, apoyo y comprensión.

जय गुरुदेव ॐ

AGRADECIMIENTOS

Primero quiero agradecer a mi familia por brindarme durante todos estos años su apoyo, paciencia y comprensión. En verdad me siento muy afortunado de tenerlos a mi lado; esta tesis es en parte suya.

También quiero agradecer a mi novia Luz Venecia Salamanca por darme su apoyo, su cariño, su comprensión y por todo el ánimo que recibí de ella para terminar este trabajo de tesis. Gracias por brindarme la luz de tu compañía y de tu amor, los cuales son los más grandes tesoros que la vida me pudo haber otorgado.

Gracias al Dr. Manuel Martin por guiar a este alucinado chico a cumplir sus metas y por brindarme un poco de su sabiduría en cada uno de sus comentarios, los cuales nunca pienso olvidar.

Además gracias especiales a la M.C Gloria Yadira Torralba por su invaluable ayuda durante todo el proceso de investigación e implementación.

También al Dr. Gonzalo Flores y al M.C Ismael García Juárez por su paciente cooperación y guía durante este proceso de investigación. Y la M.C Hilda Castillo Zacatelco y al Dr. Abraham Sánchez López que gracias a su excelente cátedra me han ayudado a forjarme como ingeniero.

Y por último gracias a todas las personas que a lo largo de mi vida me han dado una lección de vida, brindándome la oportunidad de ser una mejor persona.

GRACIAS

INDICE GENERAL

Pág.

INDICE DE TABLAS

Pág.

INDICE DE FIGURAS

Pág.

RESUMEN

Este trabajo de tesis plantea una estrategia para generar un modelo tridimensional de una neurona, a partir de una serie de fotografías axiales que contienen de manera implícita la información tridimensional de dicha neurona. Las neuronas han sido teñidas usando el método de tinción de Golgi-Cox y son fotografiadas usando un microscopio de convencional de luz clara junto con una cámara fotográfica convencional adaptada a la óptica del microscopio. Para generar el modelo se utiliza el Algoritmo de los Marching Cubes como base para la reconstrucción y una estrategia semi-automatizada en la cual el usuario de manera indirecta proporciona todos los parámetros de entrada usados por el algoritmo de reconstrucción. Cómo los parámetros de entrada del algoritmo de reconstrucción no pueden ser elegidos de manera arbitraria, estos deben de ser calculados a partir de tonos de gris de las imágenes. Para determinar estos parámetros el usuario elige algunas regiones de la imagen en donde la estructura de la neurona esté foco y otras en donde solo haya fondo, empleando esta información se estiman los parámetros de entrada del Algoritmo de los Marching Cubes. Usando el conjunto de imágenes y los parámetros adecuados el Algoritmo de los Marching Cubes genera el modelo tridimensional de la célula, para que a partir de este el usuario pueda medir manualmente la extensión de las diferentes estructuras dendríticas.

Palabras Claves: Cubos Marchantes, Reconstrucción 3D, Reconstrucción Tridimensional, Imágenes Médicas, Neuronas, Dendritas

ABSTRACT

This thesis establishes a strategy to generate a three-dimensional model of a neuron, from axial photographs where three-dimensional info of the neuron lies. These neurons have been painted using Golgi-Cox method and photographed using a conventional light microscope and normal digital camera adapted to the microscope's optics. Marching Cubes Algorithm is used to generate the 3D model of a neuron from those photographs. Also a semiautomatic strategy has been implemented to get all the input parameters to feed the Marching Cubes Algorithm. These parameters need to be chosen very carefully from the grayscale values of the images, because they can affect the final model result. The user assistant is needed to select some regions with a focused structure and some regions that only have background. With those selected regions the input the parameters for the Marching Cubes Algorithm are computed. The real resolution of the image stack can be estimated measuring a dendrite diameter in pixels and relating with the estimated user diameter in microns. Having the pixel/micron relation we can measure the 3D model in microns.

Keywords: Marching Cubes, 3D Reconstruction, Tridimensional Reconstruction, Medical Images, Neurons, Dendrites

1. ANTECEDENTES

En este capítulo se introducirán los conceptos básicos de microscopía, tomografía y la relación que existe entre estas dos disciplinas y las Imágenes Digitales. También se mencionarán conceptos básicos de neurología como las partes de la neurona y una breve descripción de su funcionamiento. De modo que se implanten los conceptos y términos que serán utilizados en los subsecuentes capítulos.

1.1 Tomografía.

En 1972 el Ingeniero Eléctrico Godfrey Hounsfield inventó el primer tomógrafo. El principio básico del tomógrafo es cuantificar la radiodensidad de los tejidos, tomando Rayos-X alrededor del objeto y después analizar las mediciones tomadas usando métodos matemáticos. Desafortunadamente su uso no se hizo común hasta la llegada de las primeras computadoras de bajo costo, ya que el tomógrafo de Hounsfield usaba un método matemático llamado Filtro de Proyección Inversa de Fourier, el cual es utilizado para reconstruir las mediciones tomadas de una sección radial del tejido y formar una imagen en 2D que representa cada una de las mediciones de radiodensidad tomadas por el tomógrafo. Debido a que es necesario interpretar las mediciones que hace el tomógrafo usando una computadora, a la tomografía también se le conoce como (TC) Tomografía Computarizada.

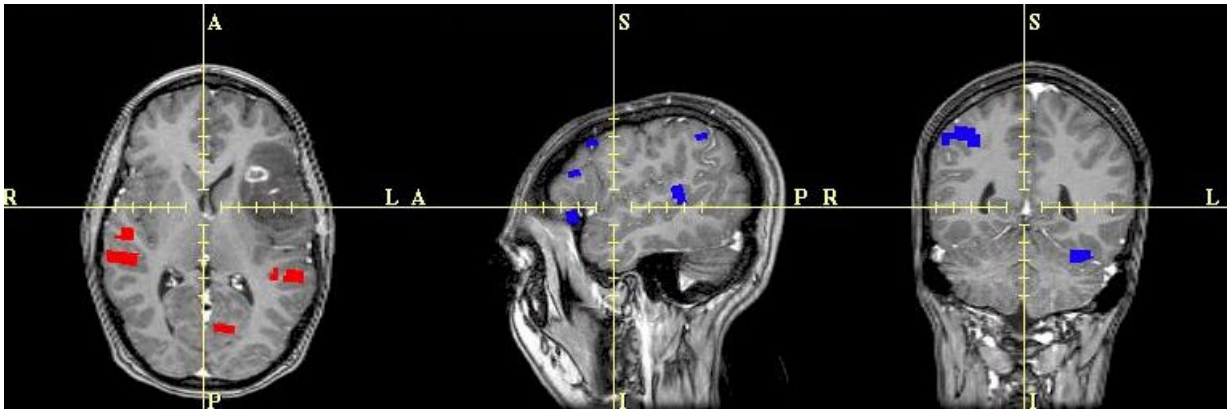

Figura 1.1 Tres imágenes tomadas con un tomógrafo computarizado, (de izquierda a derecha) la primera con una vista aérea, las dos restantes con vistas lateral y posterior respectivamente.

En la **Figura 1.1** se muestran tres imágenes de una tomografía computarizada del cerebro, si se toma un conjunto de estas imágenes se puede realizar una reconstrucción en 3D de un tejido en específico como hueso, dientes, médula, cerebro, tejido glandular etc. Esto se puede hacer debido a que cada tipo de tejido tiene una firma de radiodensidad única la cual es representada en un tono de gris en la imagen. En la **Figura 1.2** se puede apreciar la reconstrucción en 3D de un tumor cerebral, visto de color verde, a partir de las imágenes mostradas en la **Figura 1.1**.

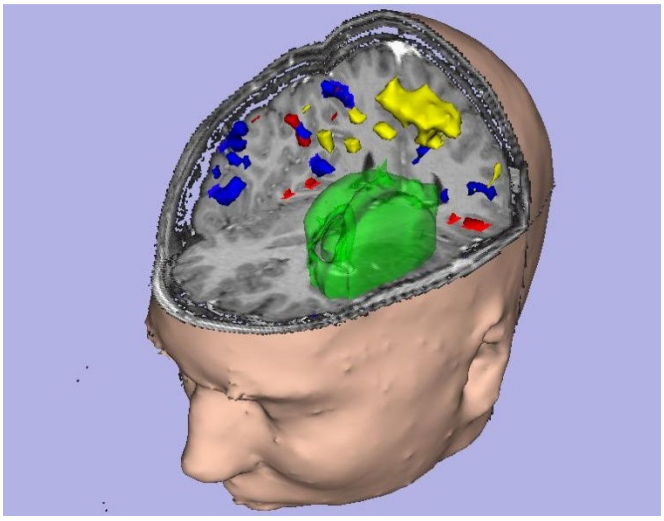

Figura 1.2 Reconstrucción 3D de una serie de imágenes tomográficas cerebrales usando voxeles. Se puede apreciar que el paciente sufre de un tumor cerebral resaltado en color verde.

Como se puede apreciar en las imágenes anteriores la tomografía es un muy buen método para obtener reconstrucciones en 3D de cualquier tejido, debido a la exactitud que tiene el tomógrafo en sus mediciones, es fácil identificar cualquier tipo de tejido. La representación en 3D de las imágenes tomográficas generalmente se realiza usando voxeles**¡Error! Marcador no definido.** en vez de usar una malla poligonal, debido a que usar voxeles facilita la representación volumétrica del tejido que se desea visualizar.

1.2 Microscopía.

La microscopía tiene sus orígenes durante el siglo I A.C, los romanos experimentaban con diferentes tipos y formas de vidrio, notaron que al tener un pedazo de vidrio más grueso en medio y más delgado alrededor, se amplificaba la imagen que pasaba a través de este. Después durante el siglo XIII D.C se empezaron a fabricar los primeros lentes para corregir los defectos de visión.

Finalmente el microscopio moderno hace su aparición en el año de 1665 a manos del físico inglés Robert Hooke, que fue el primer ser humano en ver una célula, quien es conocido como el padre de la microscopía Inglesa. Años más tarde en 1674 el holandés Anton van Leeuwenhoek interesado por el asunto aprendió a pulir sus propios lentes y logró magnificaciones de hasta 270x, usó su microscopio para observar todo tipo de objetos, sangre, plantas, semen, insectos, etc. Todas estas observaciones fueron documentadas y publicadas, por esa razón él es considerado el padre de la microscopia moderna.

Hoy en día existen muchos tipos de microscopio como los microscopios ópticos, los confocales, los electrónicos, los de luz polarizada, etc. En este caso analizaremos el funcionamiento del microscopio óptico de luz clara y el microscopio confocal, ya que las características ópticas de esos microscopios permiten realizar reconstrucciones en 3D del tejido cerebral.

1.2.1 Microscopio Confocal

A pesar de que en este trabajo no se utilizaron imágenes de un microscopio confocal, cabe dejar claro su funcionamiento, ya que este aparato tiene la capacidad de explorar en secciones transversales un material grueso y tranparente; sin necesidad de tener que rebanar la muestra. Además este instrumento tiene la capacidad de eliminar las regiones que se encuentran fuera de foco de una sección transversal y solo mostrar lo que se encuentra en foco. Estas características hacen al microscopio confocal una herramienta perfecta para generar una reconstrucción 3D del espécimen.

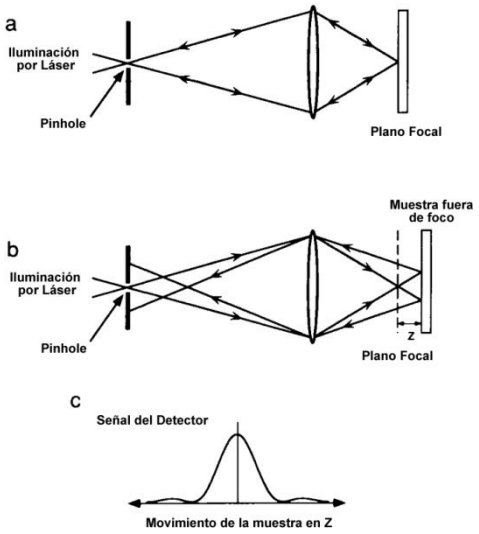

Figura 1.3 Esquema simplificado de un microscopio de escaneo confocal, mostrando una muestra a) que se encuentra en foco b) y una muestra fuera de foco. c) La forma de la señal de salida del detector como una función, mientras la muestra sale de foco.

El principio básico de funcionamiento de un microscopio confocal, es el de iluminar solamente una pequeña región de la muestra a la vez, esto a través de un dispositivo llamado pinhole, el cual es muy parecido al obturador de una cámara convencional. Y así barriendo la muestra punto por punto se puede ir formando una imagen completa de la muestra, como se ve en la **Figura 1.5.a**. Ahora si la región que fue iluminada se encuentra fuera de foco, como se puede ver en la **Figura 1.3.b**, la luz reflejada se encuentra fuera de foco y no puede pasar correctamente a través del pinhole y por ende la luz no es captada por el detector situado detrás del pinhole. El resultado de usar un pinhole es que en la imagen generada no hay regiones fuera de foco. En la **Figura 1.3.c** se puede ver la señal de salida obtenida por el detector detrás del pinhole mientras la muestra va saliendo de foco.

Otra diferencia entre un Microscopio Óptico y un Microscopio Confocal, es que el microscopio confocal en vez de usar luz convencional para iluminar la muestra usa un haz laser colimado de cierta frecuencia, el cual pasa a través de un pinhole hacia el objetivo del microscopio creando una región de difracción limitada, como se puede ver en la **Figura 1.5**.

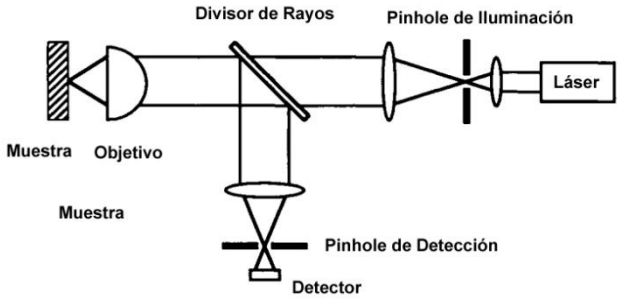

Figura 1.4 Un microscopio confocal, empleando dos pinholes en su sistema óptico.

Todos los especímenes que se analizan en un microscopio confocal deben de ser tratados con materiales que emitan fluorescencia. Debido a que la incidencia del haz laser sobre la muestra provoca la excitación de los átomos que la componen, cuando el haz se retira los átomos del espécimen tratado empiezan a emitir luz fluorescente, la cual es capturada por el detector detrás del pinhole. El Microscopio Confocal recibe ese nombre porque los lentes del objetivo son usados dos veces, una vez para iluminar la muestra usando el haz láser y la otra para ir construyendo la imagen pixel por pixel.

1.2.2 Microscopio Óptico de Luz Clara

Existen solamente dos tipos de Microscopios Ópticos los simples y los compuestos, como su nombre lo dice los simples solamente tienen un lente, mientras que los compuestos tienen más de un lente para amplificar la imagen. Un esquema funcional de un microscopio óptico se puede ver en la **Figura 1.5**.

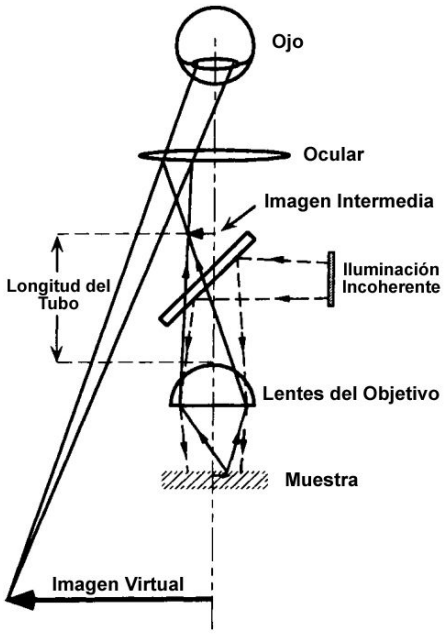

Figura 1.5 Un esquema funcional simplificado de un microscopio óptico estándar.

En este instrumento, la muestra es uniformemente iluminada usando una lámpara de filamento u otra fuente de luz incoherente como una lámpara de vapor de mercurio. Después los lentes del objetivo capturan la luz y forman una imagen inversa del objeto en el plano intermedio del microscopio. La distancia que hay entre el plano intermedio y la base del plano focal del objetivo se le conoce como la "longitud del tubo".

La imagen final es vista desde el ocular el cual forma una imagen virtual a una distancia confortable del ojo, normalmente a 250 mm para un observador estándar, proveyendo una magnificación adicional. Con este sistema la magnificación total del microscopio es obtenida multiplicando las magnificaciones del objetivo y del ocular, con lo cual se puede obtener una magnificación total de más de 2000x en un espacio muy pequeño.

Figura 1.6 Esquema de las partes de un microscopio óptico estándar.

En la **Figura 1.6** se pueden ver las diferentes partes del Microscopio Óptico o de Luz Clara. En el ocular se pude ver la imagen totalmente magnificada de la muestra; el ocular, el tubo de observación, la óptica interna del microscopio y el revólver son soportados por el brazo para permitir el libre movimiento de la platina. Los objetivos pueden cambiarse de posición usando el revólver y dependiendo del objetivo que apunte a la muestra será la amplificación obtenida. En cuanto a la fuente de luz, el diafragma y el condensador sirven para iluminar la muestra y limitar la cantidad de luz que llega a esta. Las perillas de foco grueso y fino sirven para ajustar el foco de manera tosca o de manera fina moviendo la platina pequeñas o grandes distancias.

1.2.3 Microscopio Digital

Este tipo de microscopio es solo un caso especial de microscopio óptico, ambos poseen los mismos sistemas ópticos y las mismas partes, la única diferencia entre ellos es que el digital no cuenta con objetivos para ver directamente la muestra, sino que esta solo puede ser vista a través de un dispositivo de captura digital como una cámara o una videocámara digital.

En el mercado también existen microscopios de luz clara que poseen la capacidad de funcionar como microscopios digitales, ya que estos además de contar con los oculares estándar para ver la muestra

directamente, también poseen un tubo de observación especial al cual se le puede adaptar una cámara o una videocámara digital, haciendo posible ver el espécimen de manera analógica y digital al mismo tiempo.

Las cámaras digitales ya sean fotográficas o de video en su interior tienen un chip CCD (Charged-Coupled Device o Dispositivo de Cargas Interconectadas) el cual está formado por millones de detectores de fotones alineados en una matriz los cuales almacenan la información hasta que se realiza una operación de lectura. Como los fotodetectores de un CCD están alineados en forma cuadriculada cada uno de estos representa un pixel dentro de la imagen, por lo tanto entre más detectores tenga el CCD mucha más resolución tendrá la imagen final. Es por eso que en un microscopio digital la resolución final de la imagen depende de la resolución de la cámara digital y el campo de visión del objetivo con el que se adapta la cámara al microscopio.

1.2.4 Cámara Lucida

La cámara lucida es un dispositivo creado por William Hyde Wollaston fue patentado en 1807. Este dispositivo en un principio fue usado por los artistas para realizar bocetos y dibujos. Este dispositivo ayuda al artista por que le permite mirar la escena que quiere pintar y al mismo tiempo el lienzo donde quiere dibujar, por lo que el artista solo se debe limitar a calcar la escena sobre el papel. Su introducción a la microscopia fue prácticamente inmediata, debido a que el uso de la cámara lucida hacia mucho más fácil el dibujo de cualquier estructura microscópica.

Así la cámara lucida permite al usuario observar de manera translúcida una superficie externa y al mismo tiempo ver la muestra que se encuentra en el microscopio. Por ejemplo si se tienen una hoja de papel bajo la cámara lucida y en el objetivo del microscopio se visualiza una neurona, si se usa la cámara lucida se pueden observar tanto la neurona como la hoja de papel; por lo que el trabajo de dibujar a la neurona se reduce a calcarla en la hoja de papel que se encuentra bajo la cámara. Como se puede ver en la **Figura 1.7** se ve a una chica realizando un boceto de una neurona usando un microscopio con cámara lucida.

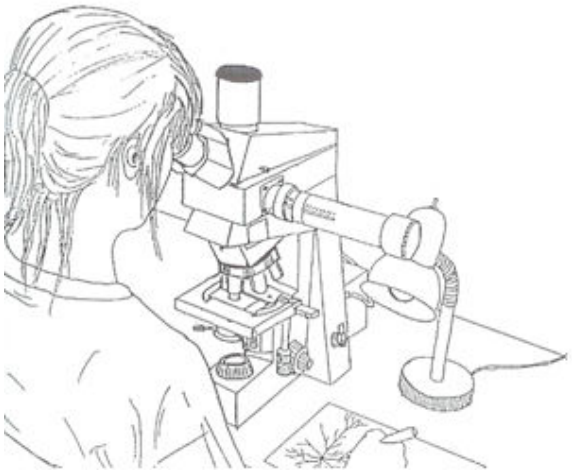

Figura 1.7 Persona realizando en boceto de una neurona usando un microscopio óptico de luz clara, adaptado con una cámara lucida

1.3 Platina Motorizada

Una vez que el microscopio fue adaptado y cuenta con una cámara digital es factible tomar una serie de imágenes de una sola célula en su totalidad de manera axial, pero el proceso suele ser bastante engorroso debido a que es necesario girar la perilla de foco fino hacia arriba o hacia abajo cada vez que se toma una fotografía y repetir el proceso hasta que se haya recorrido toda la célula. Para fotografiar de manera automática una célula se pude usar una platina motorizada que se adapta al sistema de engranes del microscopio y mueve la platina hacia arriba y hacia abajo.

Una platina motorizada, consta de un motor a pasos que va conectado directamente al sistema de engranes que al ser accionado mueve la platina y dependiendo de su polaridad puede moverla hacia arriba o hacia abajo. El motor de una platina puede ser controlado analógicamente usando una perilla o un control de encendido y apagado, y también pueden ser manejado digitalmente conectándolos a una computadora que controla el interruptor de encendido y la polaridad del motor. De manera que la platina del microscopio se mueva en distancias constantes, hacia arriba y hacia abajo, permitiendo así fotografiar la muestra de manera automática a intervalos constantes sobre el eje *z*, sin necesidad de mover la perilla de foco manualmente.

En la **Figura 1.8** se muestra el esquema de conexión del microscopio de luz clara con la cámara digital y la platina motorizada, en este diagrama se puede apreciar que tanto la cámara como la platina motorizada se encuentran conectadas a la computadora usando interfaces diferentes lo cual permite controlar la cámara y la platina de manera independiente.

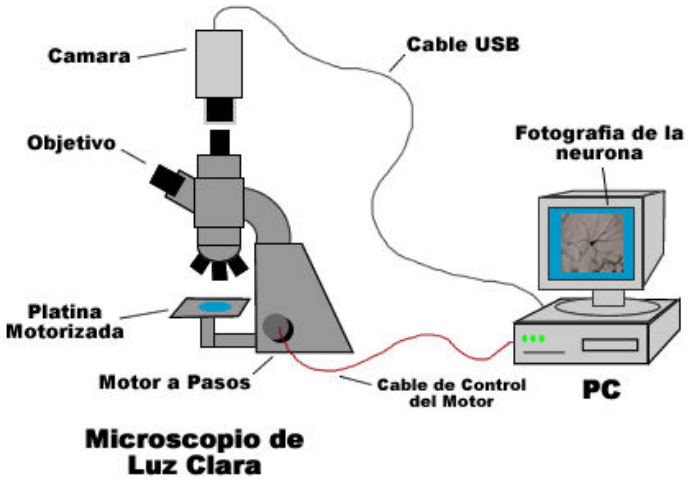

Figura 1.8 Diagrama de conexión de un microscopio de luz clara adaptado con una platina motorizada y una cámara digital conectado a una computadora

1.4 Neurología

La neurología es una rama de la medicina que se estudia y el trata las enfermedades del sistema nervioso desde un punto de vista fisiológico. Médicos como Andrés Vesalio y Claudio Galeno realizaron las primeras disecciones documentadas y serias sobre el sistema nervioso periférico y central, haciendo grandes aportes a la medicina en su tiempo. Pero la neurología no tiene su verdadero auge hasta que hasta que el médico italiano Camillo Golgi (1843-1926) quien descubrió una técnica para teñir selectivamente a un conjunto de neuronas, lo que permitió por primera vez visualizar una célula nerviosa. Años después el anatomista español Santiago Ramón y Cajal (1852-1934) usando la técnica de teñido de Golgi, desarrollo una

propuesta sobre el funcionamiento del sistema nervioso, considerando a cada neurona como una unidad funcional dentro de un complejo sistema discreto, en donde cada neurona tiene la capacidad de comunicarse con otras neuronas. A la propuesta desarrollada por Cajal se le conoce como “la doctrina de la neurona”. Por dichos logros Santiago Ramón y Cajal y Camillo Golgi recibieron el premio Nobel de Fisiología o Medicina en 1906.

El descubrimiento de la neurona como una entidad separada lleva claramente a la deducción que existe una comunicación entre neuronas y que mediante esa comunicación es como se transmite información a través de todo el sistema nervioso. No es hasta un siglo después cuando el médico Ingles Charles Scott Sherrington (1857-1952) acuñó el término sinapsis a los sitios en donde se unen las neuronas para comunicarse entre sí. Después Otto Loewi (1873-1961) en 1921 demuestra experimentalmente como es que las neuronas se comunican entre sí a través de las sinapsis usando sustancias químicas. Durante las siguientes décadas se da una explosión de investigaciones sobre el funcionamiento de la sinapsis y es así como descubren la existencia de los neurotransmisores que afectan el funcionamiento de la sinapsis. Además se plantea el funcionamiento de la sinapsis a un nivel molecular lo que permite entender la forma en la que un conjunto de neuronas realizan cálculos y almacenan información.

1.4.1 Neuropsiquiatría

La neuropsiquiatría es una rama de la medicina que estudia diferentes trastornos mentales relacionándolos con la fisiología y las enfermedades propias del sistema nervioso. Se dice que es una fusión entre la neurología, que estudia las enfermedades y el funcionamiento del sistema nervioso, con la psicología que estudia y trata los trastornos mentales como la ansiedad, stress, psicosis, depresión, etc.

Esta disciplina médica usa diferentes técnicas para hallar las causas de ciertos trastornos psiquiátricos. Una de ellas es el estudio morfológico de neuronas, la cual consiste en analizar la morfología neuronal en un modelo animal. Este método usa un modelo animal sano y se le realiza alguna cirugía o se le administra algún fármaco que produzca el trastorno mental que se desea estudiar. Una vez que se tiene un conjunto de especímenes enfermos se sacrifican y se toma una muestra de tejido cerebral, para después analizar la estructura dendrítica de sus neuronas. Dependiendo del experimento se estiman las longitudes de la estructura dendrítica y se realiza un conteo de espinas neuronales; haciendo un boceto de la neurona a mano, para después realizar un estudio estadístico de cientos de bocetos para determinar la validez de la hipótesis planteada.

El método mencionado anteriormente refleja las lesiones a nivel celular que sufre un paciente con este tipo de enfermedades, como la disminución en la longitud de la estructura dendrítica o la disminución en la cantidad de espinas dendríticas. El estudiar a detalle la estructura morfológica de las células nerviosas puede llevar al desarrollo de nuevas técnicas para tratar desordenes mentales cómo la esquizofrenia, la depresión, la neurosis, etc.

1.4.2 Neurona

La palabra neurona viene del griego *neuron* o *nerurone* que significa árbol, por la forma que tiene. Una neurona es una célula especializada, la cual es la unidad fundamental del sistema nervioso. Si analizamos a la neurona desde un punto de vista funcional, esta célula es una unidad de procesamiento que recibe y combina las señales recibidas de otras neuronas y que dependiendo de las señales recibidas envía una señal de respuesta a otras neuronas. Y trabajando todas en conjunto es como un individuo puede interpretar la realidad usando sus sentidos, controlar y mover su cuerpo, memorizar, tomar decisiones, etc.

La neurona al ser una célula especializada posee particularidades que la hacen diferente a otros tipos de célula. Como por ejemplo una neurona es capaz de reaccionar, conducir y emitir pulsos eléctricos de alto voltaje a través de ella, mientras que cualquier otra célula al ser expuesta a estos voltajes moriría. Pero una célula nerviosa no tiene capacidades de regeneración como las tiene una célula muscular o una célula de la piel; así que si una neurona muere no puede volver a regenerarse o ser sustituida por otra.

La neurona como cualquier otra célula del cuerpo tiene un núcleo y una membrana pero además de esto posee una forma y estructuras propias que le permiten realizar su tarea. Por su morfología la neurona se divide en soma, árbol dendrítico y axón. El soma es la región donde se encuentra el núcleo de la célula, el axón es la estructura mediante la cual la neurona envía señales a otras células nerviosas y el árbol dendrítico está formado por un conjunto de dendritas que están dispuestas en forma de árbol, en donde la neurona recibe las señales de otras neuronas. Dependiendo de la posición del soma existen tres clases diferentes de célula nerviosa, las unipolares, las bipolares y las multipolares, véase la **Figura 1.9**.

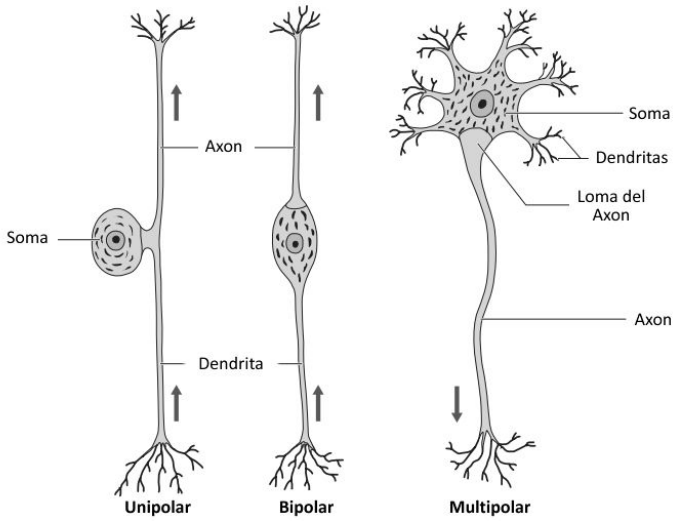

Figura 1.9 Los tres tipos fundamentales de neurona, la flecha rosa indica la dirección del impulso eléctrico.

1.4.3 Funcionamiento de la Neurona

El funcionamiento de la neurona es bastante complejo como para ser expuesto en su totalidad en este trabajo, a pesar de esto se expondrá su funcionamiento de manera simplificada. Se puede decir que la neurona funciona como una especie de mecanismo integrador, primero la neurona recibe señales a través de sus dendritas, estas señales excitan o inhiben el mecanismo de disparo de la neurona, dependiendo si membrana de la dendrita fue despolarizada la señal es excitante y si la membrana es hiperpolarizada la señal es inhibitoria. Estas señales viajan a través de la membrana de la neurona hasta llegar a la loma del axón, en su camino van despolarizando o hiperpolarizando la membrana de la neurona. Al final cuando la membrana de la neurona llegó a su límite de despolarización desde la loma del axón se emite una señal que viaja a través del axón para llegar a otras neuronas, a esta señal se le conoce con el nombre de acción potencial.

Ya que la resistencia eléctrica del citoplasma de una neurona es de alrededor de 10,000 ohm/cm, es muy difícil que una señal eléctrica pueda propagarse a través del axón sin perder potencia, debido a que la resistencia del citoplasma es bastante grande. Para evitar este problema el axón está diseñado de una manera

bastante ingeniosa, este tiene repetidores a lo largo de su trayectoria, al igual que los sistemas de transmisión modernos. Estos repetidores están hechos de un químico llamado mielina, la cual reduce la capacitancia de la membrana causando un incremento en la velocidad de propagación de la señal que a su vez amplifica la fuerza de la señal. Cada cierta distancia en el axón existen pequeños bloques de mielina que lo rodean, entre ellos hay pequeñas interrupciones llamadas nodos de Ranvier. Cada una de estas interrupciones funciona como repetidores que regeneran la señal periódicamente y le permiten llegar a las terminales del axón, en la **Figura 1.10** se puede ver un diagrama de los bloques de mielina.

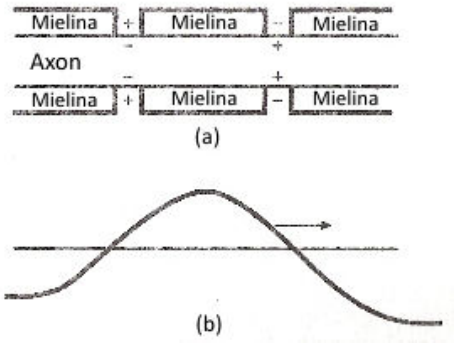

Figura 1.10 a) Distribución de los bloques de mielina indicando la polaridad de la carga, b) Señal de acción potencial, indicando su dirección.

Cuando el impulso eléctrico generado por la neurona finalmente llega al final del axón, mejor conocido como sinapsis, que es lugar donde dos neuronas se conectan. La sinapsis tiene la forma de un botón, al final del axón se le conoce como terminal presináptica mientras que la parte receptora de la otra neurona se le conoce como terminal postsináptica. En medio de estas dos terminales existe una separación conocida como el precipicio sináptico. En la terminal presináptica hay vesículas y una mitocondria, esta produce químicos neurotransmisores mientras que las vesículas los almacenan. Después dado que la membrana de la célula viene cargada con el impulso eléctrico, este impulso cuando llega a la terminal presináptica atrae a los iones de calcio que se encuentran en el ambiente hacia las vesículas haciendo que estas se juntes y se fusionen con la membrana provocando la liberación del neurotransmisor, después las vesículas vuelven a su estado original e inmediatamente son rellenadas por la mitocondria con más neurotransmisores. Después los neurotransmisores llegan a la terminal postsináptica donde los neurotransmisores entran en canales químicamente activados que dejan pasar al neurotransmisor, dependiendo de qué neurotransmisor se haya recibido la señal puede ser excitante o inhibitoria para la neurona receptora y así el ciclo vuelve a comenzar. En la **Figura 1.11** se puede apreciar un esquema de las sinapsis.

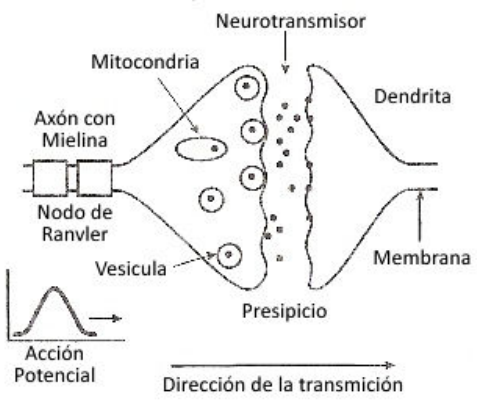

Figura 1.11 Diagrama de una sinapsis en acción.

1.4.4 Método de Tinción de Golgi-Cox

Es necesario mencionar brevemente en qué consiste el método de teñido neuronal de Golgi-Cox, ya que el sistema implementado solamente funciona con imágenes de neuronas tratadas con esta técnica. El método consta de dos fases primero el teñido y después el revelado. Se puede teñir una muestra de cerebro de cualquier animal sumergiéndola en una solución hecha con Dicromato de Potasio, Cloruro de Mercurio y Cromato de Potasio se deja la muestra sumergida durante aproximadamente 15 días. Pasado ese tiempo el tejido cerebral se cambia de solución por una de sacarosa al 30% este cambio se debe realizar en completa obscuridad y se deja reposar un día. Después la muestra puede ser cortada usando un vibrotomo y posteriormente la muestra ya rebanada se revela.

Para revelar la muestra esta se lava con agua destilada y después se la incuba durante 30 min en una solución 1:1 de fijador Kodak con agua destilada. Después el tejido se vuelve a lavar usando agua destilada y posteriormente se deshidrata usando alcohol y xileno. Este proceso tiñe de manera aleatoria el 1% de todas las neuronas que están en la muestra, lo que permite analizar individualmente la morfología de una sola o de varias de ellas bajo un microscopio. En la **Figura 1.12** se puede ver una neurona teñida usando el método de Golgi-Cox.

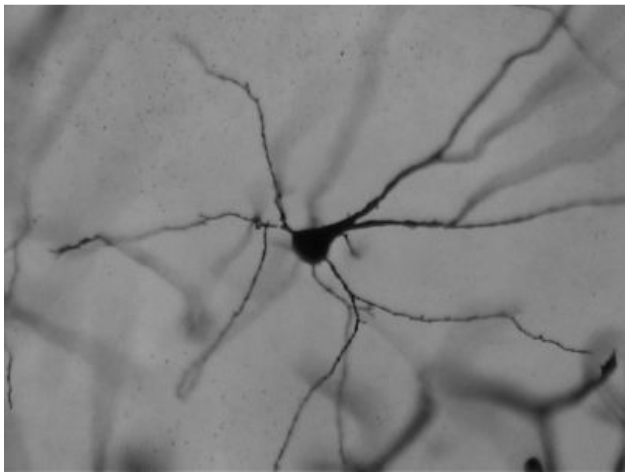

Figura 1.12 Imagen de una neurona teñida con el método de Golgi-Cox.

1.5 Imágenes Digitales

El desarrollo de las imágenes digitales comenzó a principios de los años 60, al inicio fue una rama de la computación a la que se le prestaba poca atención, debido a que la tecnología de la época hacía que el procesamiento digital de imágenes fuera costoso computacional y monetariamente hablando. Es por eso que solo equipos de trabajo especializados y con amplios presupuestos podían costear el lujo de usar imágenes digitales para sus investigaciones. Poco a poco conforme las computadoras fueron más poderosas y sus costos fueron bajando el área de imágenes digitales fue ganando terreno en diversas áreas del conocimiento humano como en la astronomía, la física, la geología, la topografía, la medicina, etc. Así es como el procesamiento digital de imágenes es ahora una herramienta fundamental para varias áreas del conocimiento humano, la cual ayuda a obtener, analizar y apreciar detalles que de otra manera hubieran sido desapercibidos de no haber usado imágenes digitales.

Gracias a descubrimientos en el campo de la física como los Rayos-X, la resonancia magnética, la radioactividad y los ultrasonidos, se han podido desarrollar tecnologías capaces de capturar en una imagen el efecto de todos estos fenómenos sobre el cuerpo humano, creando así herramientas más efectivas para el diagnostico de enfermedades. Las imágenes resultantes de los sistemas médicos deben de almacenarse, cuantificarse, mejorarse, segmentarse, visualizarse e interpretarse. Para facilitar y automatizar todo este trabajo, las imágenes son digitalizadas y procesadas utilizado diferentes técnicas computacionales, lo cual mejora y agiliza el diagnóstico médico, ayuda a planear cirugías y también sirve al desarrollo de nuevas curas para diversas patologías.

Incluso existen tecnologías que no podrían existir si no es por el procesamiento de imágenes digitales, como el tomógrafo computarizado, ya que interpretar la información arrojada por un tomógrafo sería muy difícil o casi imposible, si no se cuenta con una computadora que procese la información generada por el tomógrafo y cree una imagen digital a partir de esos datos.

1.5.1 Definición de Imagen Digital

Una imagen digital puede ser vista de dos maneras matemáticamente hablando como una función de una señal en dos dimensiones o computacionalmente hablando como un conjunto de datos almacenados en una matriz de dos dimensiones. Desde el punto de vista matemático si vemos a una imagen como la función de dos dimensiones de una señal continúa,

$$f(x,y) \qquad \textbf{Ecuación 1.1}$$

Si interpretamos la función que se ve en la **Ecuacion 1.1**, podremos ver que cada coordenada x, y determina la posición de un número que representan una tonalidad de gris de una imagen completa, a este número lo podemos representar como un vector $r = f(x,y)$. A este tipo de funciones se les conoce como funciones de dominio espacial, ya que el par x,y determina la posición exacta donde se ubicará el dato, lo cual nos hace pensar que la función pudiera extenderse hasta el infinito, pero para las imágenes digitales las dimensiones de x,y deben de ser finitas.

$$x, y \in \mathrm{N} \quad \text{donde} \quad \mathrm{x} \in \langle -\mathrm{x_{max}}, \mathrm{x_{max}} \rangle, \mathrm{y} \in \langle -\mathrm{y_{max}}, \mathrm{y_{max}} \rangle \ \mathrm{y}\ \mathrm{r} = \mathrm{f(x,y)} \qquad \textbf{Ecuación 1.2}$$

Nótese que la región descrita por las coordenadas x,y forman un rectángulo finito, lo cual es ideal para definir una imagen digital en escala de grises. Se puede redefinir la función matemática anterior (véase **Ecuación 1.2**) y transformarla en una función vectorial la cual tenga tres componentes por coordenada uno por cada color; los cuales definen el nivel de brillo del rojo, el verde y el azul. Teniendo así la siguiente función vectorial (véase **Ecuación 1.3**).

$$f(x,y) = [f_R(x,y), f_V(x,y), f_A(x,y)]^T \qquad \textbf{Ecuación 1.3}$$

También se pueden definir funciones en los números complejos para representar imágenes, esto con el propósito de simplificar los cálculos al momento de procesar la imagen. Además se pueden definir funciones con un número arbitrario de variables o dimensiones, estas se usan en caso de que se tenga una señal multidimensional como la de las mediciones de radiodensidad arrojadas por un tomógrafo, en el cual se tienen 4 dimensiones, el largo y el alto de la imagen, como se tienen varias imágenes una por cada sección transversal agregamos el ancho y si también tomamos en cuenta el tiempo ya tenemos cuatro dimensiones lo cual genera una función del siguiente tipo (véase **Ecuación 1.4**).

$$f(x,y,z,t) \qquad \textbf{Ecuación 1.4}$$

En términos computacionales cada coordenada *(x,y)* de la **Ecuación 1.3** define un solo pixel, el cual tiene tres componentes de color uno para el rojo, otro para el verde y otro para el azul que en conjunto definen el color final que vemos en pantalla de cada pixel. Los valores numéricos que representan el brillo que cada componente generalmente van del 0 al 255. Si se desea definir una imagen en escala de grises solamente se debe de indicar un solo valor de intensidad de gris por cada pixel, si la imagen cuenta con tres componentes de color y se define el mismo valor para los tres colores la imagen resultante estará en escala de grises, debido a que no existe ninguna diferencia de intensidad entre los tres componentes.

$$f(x,y,z) = [f_R(x,y,z), f_V(x,y,z), f_A(x,y,z)]^T \qquad \textbf{Ecuación 1.5}$$

La **Ecuación 1.5** describe matemáticamente una imagen de tres dimensiones, donde *(x,y,z)* determinan la posición del pixel dentro de un conjunto de imágenes de dos dimensiones ordenadas en un arreglo. En donde la posición del pixel dentro de una imagen individual está dada por *(x,y)* mientras que la coordenada *(z)* determina en que numero dentro del arreglo de imágenes se encuentran dicho pixel; mientras que f_R, f_V, f_A determinan el valor de color de ese pixel en sus tres canales.

Este tipo de imágenes son el resultado final de una resonancia magnética, una tomografía computarizada o del barrido de una célula usando un microscopio confocal o uno de luz clara, ya que en cada uno de estos casos se están tomando mediciones o se está fotografiando a un objeto tridimensional por segmentos axiales. Es por eso que a partir de estos medios es posible generar un arreglo de imágenes que en conjunto contienen la representación implícita de un objeto en 3D y que mediante diferentes algoritmos el objeto que se hallaba implícito dentro de estas imágenes puede llegar a ser reconstruido en tres dimensiones reales.

1.6 Algoritmo de los Cubos Marchantes (Marching Cubes Algorithm)

Durante el desarrollo de esta tesis se buscaron diversas estrategias para resolver el problema de la reconstrucción, algunas ideadas por los integrantes del equipo de trabajo y otras basadas en algoritmos de reconstrucción ya existentes, pero ninguna de estas cumplió con las expectativas propuestas, ya sea por su costo computacional, por su falta de fiabilidad o simplemente su implementación no era factible. Dados estos problemas el equipo de trabajo decidió utilizar los algoritmos de reconstrucción 3D ya existentes y adaptarlos a las características propias del problema. El algoritmo de reconstrucción que fue elegido fue el algoritmo de los Cubos Marchantes o por su nombre en inglés Marching Cubes Algorithm.

El algoritmo de los marching cubes fue desarrollado por William E. Lorensen y Harvey E. Cline y fue publicado en SIGGRAPH 1987, este algoritmo fue patentado dos años antes de su presentación en SIGGRAPH en el año de 1985, esta patente expiró en 2005 y el usó comercial de este algoritmo se ha vuelto abierto. Actualmente este algoritmo es extensivamente usado en video juegos, aplicaciones científicas, pero sobre todo es usado en la medicina para generar modelos 3D partiendo de imágenes de tomografía o resonancia magnética

Este algoritmo fue diseñado en un principio para generar modelos 3D de imágenes médicas, como lo indica el resumen del artículo original. “Nosotros presentamos un algoritmo llamado marching cubes, el cual genera modelos triangulares de superficie de densidad constante a partir de datos médicos en 3D. Usando una técnica de divide y vencerás para generar la interconexión entre los diferentes planos hemos creado una tabla que define la topología de los triángulos, Este algoritmo procesa toda la información médica en 3D escaneándola de manera lineal y calcula los vértices de los triángulos usando una interpolación lineal. Nosotros encontramos el gradiente de los datos originales, normalizándolos y usándolos como base para la iluminación del modelo. Los detalles del modelo generado son el resultado de mantener la conectividad entre

los planos, los datos de la superficie y la información gradiente presentes en los datos 3D originales. Los resultados de una tomografía computarizada, una resonancia magnética o de una tomografía computarizada de emisión de fotones, muestran la funcionalidad y la calidad del algoritmo marching cubes. También discutimos las mejoras que disminuyen los tiempos de procesamiento y que le dan capacidades sólidas de modelaje." (Lorensen y Cline - SIGGRAPH 87)

Este algoritmo fue elegido por varias razones, la primera fue su rapidez ya que el diseño del algoritmo toma en cuenta algunos aspectos de optimización que elevan su rendimiento y ayudan a que su implementación sea más sencilla y factible.

La segunda razón por la cual se eligió este algoritmo fue por que implícitamente cuenta con una función que permite diferenciar las regiones que se encuentran en foco, ya que utiliza un valor de superficie para diferenciar la estructura a reconstruir del fondo.

Y el último motivo fue porque el algoritmo toma en cuenta el hecho de que la información en tercera dimensión subyace en capaz separadas, que en este caso serían las diferentes imágenes que conforman el paquete y este es capaz de crear el modelo tomando en cuenta este hecho.

2. ANALISIS

El presente capítulo describirá a grandes rasgos el trabajo desarrollado en el Laboratorio de Neuropsiquiatría y la problemática que éste departamento de investigación enfrenta. También se enunciarán las razones por las cuales este sistema fue desarrollado, presentando las ventajas que representa usar el sistema de software desarrollado en contra del método manual de análisis de morfología-neuronal usado actualmente por el laboratorio. Además se mencionará el material con el que cuenta el laboratorio para realizar el proceso de captura de imágenes.

2.1 Laboratorio de Neuropsiquiatría

En el laboratorio de Neuropsiquiatría del Instituto de Fisiología de la BUAP, se realizan estudios especializados sobre trastornos psiquiátricos como la esquizofrenia y la depresión. La investigación que aquí se realiza busca entender estos trastornos desde un punto de vista neurofisiológico, es decir busca entender cómo cambia el tamaño, la forma y la estructura de las neuronas del sujeto de investigación cuando este sufre de alguno de los trastornos antes mencionados.

También se debe mencionar que aparte del tamaño, la forma y la estructura, se analizan los cambios de tamaño, tipo o cantidad en las espinas dendríticas de las neuronas dentro del Sistema Nervioso Central del Paciente. Estos estudios principalmente se efectúan utilizando animales, en general pueden utilizarse diferentes tipos de animales como ratas o chimpancés, en el caso del Laboratorio de Neuropsiquiatría sus estudios los realizan utilizando ratas blancas; cuya cepa se produce en el bioterio de la BUAP llamado “Claude Bernard”.

Para provocar que el animal en cuestión llegue a sufrir del trastorno psiquiátrico que se desea estudiar, se utilizan diversas técnicas como provocar lesiones en su sistema nervioso mediante alguna cirugía, administrando de manera crónica un psicofármaco o también se puede alterar el medio ambiente del animal forzándolo a vivir en un ambiente estresante que provoque el trastorno requerido.

En este laboratorio también se estudia el comportamiento y la estructura neuronal en animales que a temprana edad han sido separados de la madre o que se les ha enfrentado a cierto trauma lo cual puede llegar a provocar en el animal adulto un comportamiento anti-social que se ve reflejado en la morfología de sus neuronas.

Una vez que se tiene un modelo animal que sufra algún trastorno psiquiátrico se debe de tomar una muestra de su tejido nervioso, que por lo general requiere el sacrificio del espécimen. La técnica que se utiliza para extraer la muestra depende del tipo de investigación y de las regiones del cerebro que se estén estudiando.

Una muestra de tejido neuronal solo puede ser analizada bajo el microscopio si esta ha sido teñida usando la técnica de Golgi-Cox, la cual tiñe las neuronas para que estas puedan ser apreciadas individualmente. La técnica de tinción de Golgi-Cox solo tiñe el 1% de las neuronas de la muestra, lo cual permite su estudio morfológico individual. De otra manera sería muy difícil la identificación de una sola neurona, ya que el tejido neuronal suele ser bastante denso.

Ya que el investigador tiene la muestra tratada con Golgi-Cox, esta puede ser apreciada en el microscopio de luz simple, usando lápiz, papel y la cámara de dibujo del microscopio el investigador realiza

el boceto de una neurona registrando su estructura dendrítica, para después hacer un estudio estadístico de cientos de bocetos. Generalmente en cada boceto debe aparecer la forma del soma junto con sus dendritas, las cuales son dibujadas con diferentes colores que representan en nivel de ramificación de cada una de ellas. Opcionalmente en el boceto puede ser dibujada la cantidad de espinas que posee una de las dendritas de la neurona (véase **Figura 2.1**).

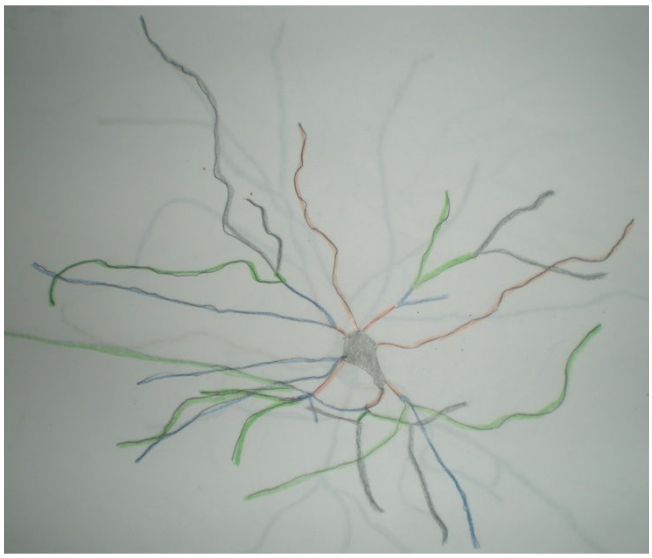

Figura 2.1 Boceto manual de una neurona usando una cámara lucida.

Para comprobar la validez de la hipótesis planteada como mínimo se deben de realizar los bocetos de 10 neuronas por cada región cerebral que se esté estudiando, en donde cada grupo debe de constar por lo menos de 8 especímenes. Suponiendo que se esté haciendo un estudio morfológico de 4 regiones cerebrales diferentes, teniendo 3 grupos de 10 especímenes, entonces se tendrán que realizar en total 1600 bocetos. Generalmente este tipo de estudios son llevados a cabo por un conjunto de personas que trabajan en equipo para dibujar los bocetos necesarios. Estos dibujos son utilizados para llevar a cabo un análisis estadístico de la información contenida en ellos y así determinar la validez de la hipótesis inicialmente planteada.

2.2 Ventajas

Para evitar que el estudio tenga que realizarse manualmente existen diversos sistemas de software que usando un conjunto de fotografías de una célula de manera automática generan un modelo 3D y obtienen las longitudes de las ramificaciones neuronales. Estos sistemas suelen ser propietarios lo cual los hace sumamente caros y además se necesita un microscopio especialmente adaptado para que el sistema funcione de manera correcta. Entre la lista de programas de este tipo se encuentran: Neurolucida, Bitplane, NeuroImage, AutoNeuron, etc. También existen herramientas gratuitas como Neuromantic, las cuales no poseen todas las características requeridas por el Laboratorio y su uso requiere capacitación.

Desafortunadamente el Laboratorio de Neuropsiquiatría no cuenta con los recursos económicos necesarios como para adquirir el equipo adecuado, ya que cualquiera de los paquetes de software propietarios mencionados anteriormente, pueden llegar a ser muy caros. Además estas soluciones de software requieren que la captura de las imágenes se realice usando cierta marca y tipo de microscopio, junto con un ocular y una platina motorizada de la misma marca y del mismo tipo, lo cual encarase aún más la solución.

Para comprobar la hipótesis que la investigación plantea, es necesario realizar el análisis estadístico de los datos obtenidos analizando cientos de bocetos, lo cual puede llegar a requerir de todo un equipo de

trabajo para realizar los dibujos necesarios para llevar a cabo los cálculos estadísticos. Es por eso que el llevar a cabo una investigación de estudios morfológicos puede llevar bastante tiempo y llegar a ser un proceso engorroso, debido a que todo se debe de hacer manualmente

Por lo anterior el Laboratorio de Neuropsiquiatría ha pedido la asistencia de la Facultad en Ciencias de la Computación, para desarrollar una herramienta de software propia que ayude a automatizar los estudios neuro-morfológicos del laboratorio. Esta herramienta está diseñada bajo los parámetros establecidos por los miembros del laboratorio y además su uso se ha adaptado al equipo que el laboratorio actualmente posee.

Una de las ventajas primordiales que se tiene en una reconstrucción en 3D al método manual de dibujo de neuronas, es que se pueden apreciar detalles que sería imposible apreciar en un dibujo hecho a mano o en una fotografía. Además en un modelo 3D se puede medir la longitud de las estructuras dendríticas en tres dimensiones, mientras que en el boceto a lápiz esta longitud se estima en dos dimensiones.

Otra de las ventajas que representa implementar una herramienta como esta en la misma universidad, será que el Laboratorio de Neuropsiquiatría contará con un paquete de software hecho a la medida que se ajustará a las necesidades y al equipo con el que cuenta el Laboratorio. A este software se le podrán hacer mejoras o cambiar características conforme así se requiera. Además el Laboratorio de Neuropsiquiatría no tuvo que afrontar la importante inversión de adquirir una solución propietaria, ya que el costo del desarrollo del paquete es casi nulo debido a que está siendo desarrollado por la misma universidad.

Por otra parte el uso de un paquete de software para el análisis morfológico de las neuronas ayudará a guardar el registro de cada neurona en un medio electrónico, evitando tener que guardar cientos de bocetos en papel para tener registro de cada análisis.

Existen otras características que en un futuro podrían incluirse y que serían de gran ayuda, pero que desafortunadamente rebasan los alcances de este trabajo de tesis. Como por ejemplo implementar la capacidad de analizar las longitudes dendríticas de manera automática y que el mismo sistema sea capaz de llevar la estadística de todas esas mediciones. De esta forma el único trabajo que se tendría que hacer sería proveer al software de varias series de imágenes con neuronas fotografiadas, para que este realice el análisis y la estadística automáticamente.

2.3 Equipo

A continuación se mencionará la marca y modelo de los equipos utilizados en el Laboratorio de Neuropsiquiatría, para capturar el conjunto de fotografías que sirvieron como base para el desarrollo de este trabajo de tesis.

Microscopía

Microscopio # 1

- Microscopio de luz simple Leica DM 2000 con cámara lucida de la misma marca.
- Adaptadores ópticos marca Cannon.
- Cámara Fotográfica Cannon
- Platina motorizada marca OptiScan.

En la **Figura 2.2** se muestra una fotografía del Microscopio #1 con todo el equipo de captura conectado.

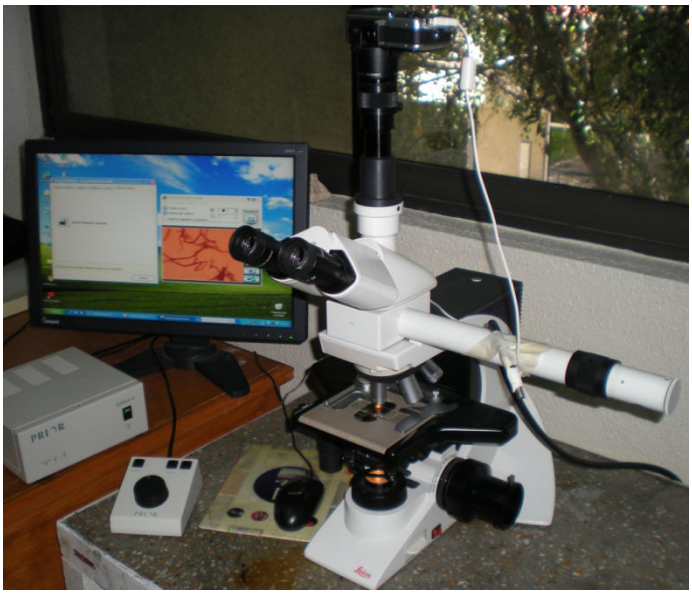

Figura 2.2 Fotografía del Microscopio #1 junto con los sistemas utilizados para realizar la captura.

Microscopio # 2

- Microscopio de luz simple Leica DMLS con cámara lucida de la misma marca.
- Cámara de video a color JVC TK-C1380

En la **Figura 2.3** se muestra una fotografía del Microscopio #2 con todo el equipo de captura conectado.

Figura 2.3 Fotografía del Microscopio #2 junto con los sistemas utilizados para realizar la captura.

Equipo de Cómputo

Ensamble Intel 2 Core Duo a 3.16 GHz con 3.23 Gbytes de RAM.

En la **Figura 2.2** se muestra el equipo de cómputo descrito anteriormente.

Es importante mencionar el equipo con el que cuenta el Laboratorio de Neuropsiquiatría, debido a que el software fue desarrollado usando como base las imágenes generadas por este conjunto de equipos. A pesar de esto la implementación del sistema ha contado siempre con un enfoque general, lo cual le permite al software trabajar de manera normal a pesar de que las imágenes hayan sido capturadas usando otro tipo de equipo.

3. PLANTEAMIENTO DEL PROBLEMA

En este capítulo se analizará en qué consiste el problema, además se mencionarán las diferentes etapas del proceso de reconstrucción estudiando a detalle cuáles son sus respectivas problemáticas y cuál fue la estrategia planteada para resolverlas.

3.1.1 Planteamiento general del problema.

El problema en general consiste en lo siguiente, generar el modelo tridimensional de una célula nerviosa, teñida con el método de Golgi-Cox, a partir de un conjunto de imágenes axiales tomadas usando una cámara digital adaptada a un microscopio de luz clara. Para después tener la posibilidad de determinar la longitud en escala real de cada una de las ramificaciones de la neurona (véase **Figura 3.1**).

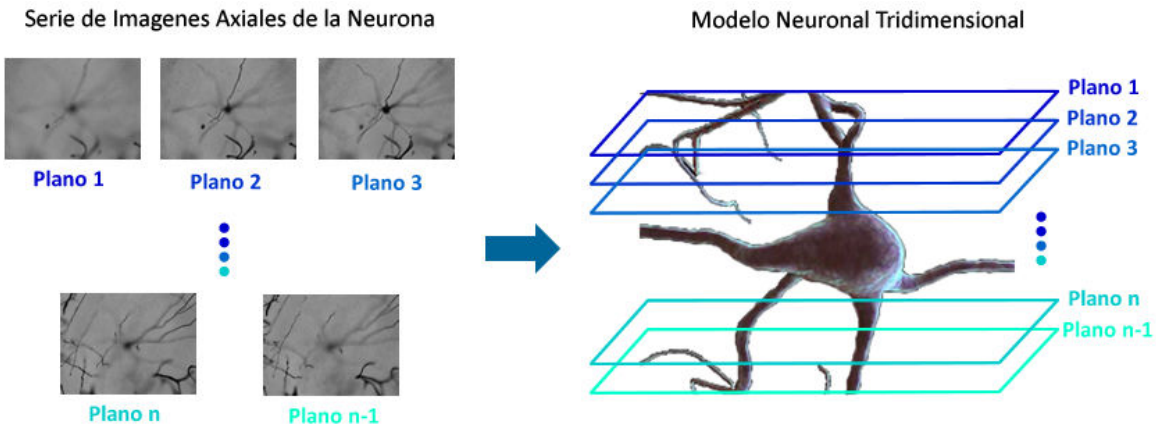

Figura 3.1 Esquema de fotografía axial de una neurona.

Entonces el problema en general consiste en generar un modelo tridimensional apegado a las dimensiones y a la morfología real de la neurona utilizando un arreglo de imágenes tomadas con un microscopio de luz clara.

Los pasos necesarios para generar la reconstrucción 3D final, se muestran en la **Figura 3.2**. Primero se debe tener una muestra de tejido neuronal teñida con el método de Golgi-Cox para poder visualizar con un microscopio de luz clara las neuronas que se encuentran dentro de la muestra. Usando el microscopio se localiza una neurona relativamente aislada, la cual cumpla con las características morfológicas que el investigador busca; una vez hecho esto le neurona es fotografiada mediante una cámara digital especialmente adaptada al microscopio. Utilizando las fotografías se determinan los parámetros de calibración para que el algoritmo de los marching cubes pueda generar un modelo tridimensional apegado a las fotografías, el cual podrá ser visualizado desde cualquier ángulo y cuyas dimensiones reales se podrán estimar.

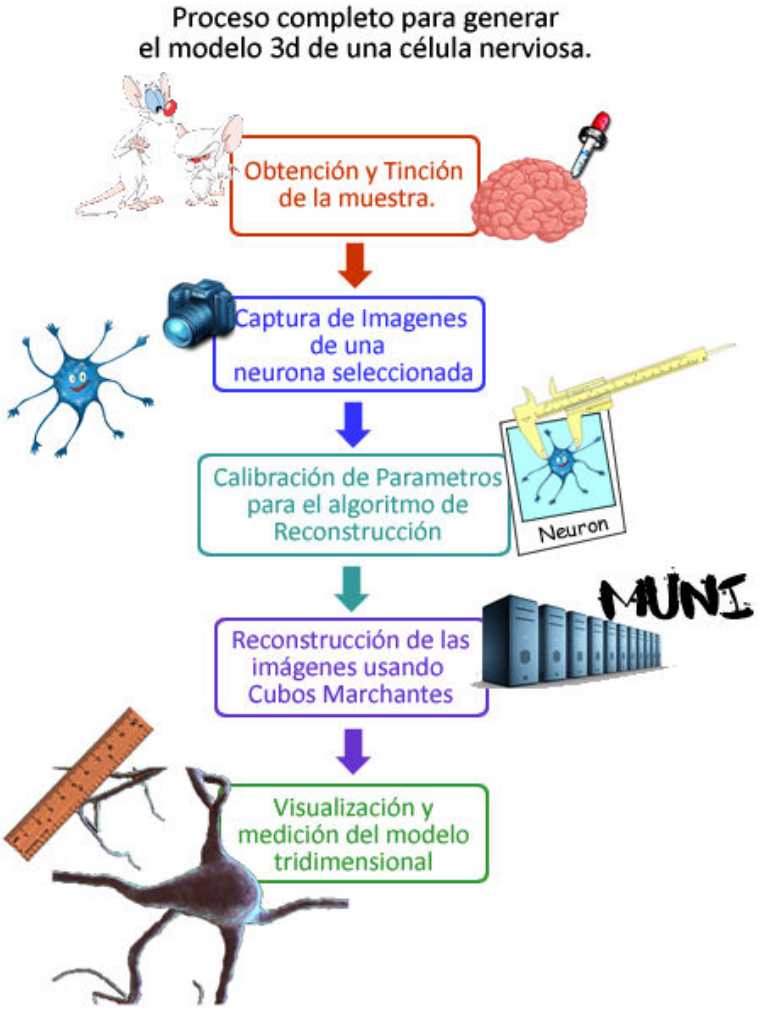

Figura 3.2 Diagrama de flujo que describe de manera general los pasos que constituyen el problema a resolver.

3.1.2 Obtención y Tinción de la Muestra

El investigador encargado debe tomar una muestra de tejido cerebral del modelo animal que se desea estudiar. La técnica utilizada para obtener la muestra varía dependiendo del tipo de enfermedad o la región cerebral que se esté estudiando. En casi todos los casos se requiere el sacrificio del animal, ya que la mayoría de las investigaciones requiere que la muestra sea extraída del sistema nervioso central del espécimen.

La muestra tomada del espécimen debe de ser tratada usando la Técnica de Tinción de Golgi-Cox, esta técnica es absolutamente necesaria para poder realizar el estudio morfológico de la neurona, ya que el tejido nervioso suele ser tan denso que sería muy difícil visualizar; para estudiar la morfología de una sola neurona.

La técnica de tinción de Golgi-Cox hace posible poder estudiar la morfología de neuronas individuales, ya que solo son teñidas el 1% de todas las neuronas que la muestra contiene; lo cual permite poder visualizarlas y estudiarlas de manera individual. Cabe recalcar que solo es posible realizar la reconstrucción de neuronas que han sido teñidas usando esta técnica, si la muestra no fue tratada usando esta

técnica la reconstrucción no se podrá realizar, debido a que los algoritmos de reconstrucción han sido adaptados para trabajar con imágenes de neuronas teñidas.

Para que la reconstrucción se lleve a cabo con éxito es necesario que el método de tinción de Golgi-Cox sea aplicado de manera correcta, de lo contrario las estructuras dendríticas aparecerán traslúcidas y difusas. El que las estructuras dendríticas no se aprecien de manera correcta puede provocar fallas durante el proceso de reconstrucción, teniendo como resultado un modelo erróneo que no se apegue a la realidad.

Este problema es muy fácil de evitar, lo único que hay que hacer es tomar las fotografías de muestras bien teñidas en donde el tinte aún siga adherido a la esta.

3.1.3 Captura de las Imágenes de la Neurona

Cuando una muestra de tejido nervioso teñida con Golgi-Cox se observa con el microscopio, se pueden contar cientos de neuronas de los miles o millones que en total se hallan en la muestra. De esos cientos de células el investigador debe de seleccionar las que tengan las características deseadas, lo cual depende del tipo de investigación. El usuario usando el microscopio explora la muestra en busca de una neurona cuya morfología se apegue a los parámetros de su investigación.

La neurona que haya sido seleccionada debe de ser fotografiada en su totalidad usando una cámara digital, adaptada al microscopio. La captura de estas imágenes puede realizarse de forma manual o de manera automática mediante el uso de una platina motorizada.

En el caso de una captura manual el usuario debe de ubicarse en el plano focal más alto o más bajo de la célula, una vez allí deberá mover la platina del microscopio manualmente una longitud constante hacia arriba o hacia abajo para tomar una fotografía, este proceso se repite secuencialmente hasta haber recorrido toda la célula. En general se requiere tomar de 80 a 200 imágenes para fotografiar una célula nerviosa completa, a este conjunto de imágenes le nombraremos paquete de imágenes.

En el caso de una captura automática el usuario debe de ubicar el foco del microscopio en el plano focal más alto o más bajo de la neurona, después usando el software de automatización se indican cuantas fotografías se van a tomar y se comienza el proceso. La computadora envía una señal a la platina moviéndola una longitud constante y después toma una fotografía de dicho plano, el proceso anterior es automáticamente controlado por la computadora hasta que toma el número de fotografías indicado inicialmente por el usuario.

Ya sea que la captura de las imágenes se haga de manera manual o automática es posible generar una reconstrucción fiable de la célula, solo se deben de tomar en cuenta varios aspectos cuando se realice la captura, para que los resultados de la reconstrucción sean congruentes con lo que se muestra en las fotografías.

El primer aspecto que se debe tomar en cuenta es que las imágenes deben de ser capturadas siempre de manera secuencial, no se debe de mover la platina en sentido contrario al que se está fotografiando. Es decir si se empezó a fotografiar a la célula desde el plano focal más alto la platina se debe de ir moviendo siempre hacia abajo para evitar tomar una fotografía de un plano anterior, lo cual afectaría la secuencia de las imágenes.

Otro aspecto que se ha de tomar en cuenta, es ni que la muestra ni el microscopio se muevan durante el proceso de captura. Si durante el proceso de captura la muestra llegará a moverse, la posición de la neurona en el plano que se movió, no coincidirá con la posición que tiene en los planos fotográficos anteriores. Esto provocará que el algoritmo de reconstrucción genere deformidades o saltos en el modelo.

Por último se recomienda al usuario nombrar las fotografías, que conforman al paquete de imágenes, de manera secuencial para que el proceso de reconstrucción sea más sencillo. No es específicamente necesario tener las imágenes nombradas de manera secuencial pero de no hacerlo así el usuario tendrá que ponerlas en orden antes de generar el modelo 3D. Si el paquete de imágenes estuviera en desorden entonces el modelo generado tendrá deformidades o discontinuidades en los planos que se encuentran en desorden.

3.1.4 Calibración de parámetros de reconstrucción

Existen ciertos parámetros que son indispensables tanto para generar la reconstrucción como para tomar mediciones a partir de ella. Para ser capaces de tomar mediciones del modelo se debe de contar con la resolución en micras de las imágenes, es decir es necesario saber la relación pixel/micra que tiene el paquete de imágenes para así poder relacionar ambas referencias y obtener una relación. Mientras que para generar la reconstrucción es necesario determinar el valor de superficie cuya correcta elección es indispensable para obtener una buena reconstrucción.

Para poder medir la longitud de las estructuras dendríticas del modelo final, es necesario saber cuántos pixeles equivalen una micra o cuantas micras equivale un pixel. Esta relación entre pixel/micra es necesaria para obtener las longitudes de las estructuras dendríticas del modelo 3D de la neurona, sin estos datos sería muy difícil determinar las dimensiones reales del modelo. Esta relación varía dependiendo del tipo de microscopio, la resolución de la cámara y el objetivo utilizado, esto hace que sea un dato difícil de determinar ya que depende de muchas variables.

Existen algunas cámaras de video especializadas para microscopios, en donde la relación pixeles/micras puede ser consultada en el software de captura de la misma cámara. Pero existen otras cámaras que no poseen esta capacidad y que el dato de relación entre pixeles y micras debe de ser estimado usando mediciones indirectas.

En el caso de que no se cuente con el dato de relación pixel/micra, este puede ser estimado usando una técnica ideada por el equipo de investigación. Para que esta técnica funcione primero el usuario debe de elegir una dendrita del paquete de imágenes e indicar el diámetro en micras de esta, una vez hecho esto se mide el diámetro de la dendrita en pixeles y comparando estas dos medidas se obtiene la relación pixel/micra para ese set de imágenes. Cabe la pena resaltar que esta técnica solo es una estimación de ninguna manera debe de ser considerado como una medida real, debido a que la relación es estimada usando una medición indirecta; por lo tanto las mediciones tomadas usando esta relación también serán solo estimaciones.

También es necesario determinar el valor de la superficie a partir de las imágenes de la neurona, el cual es un tono de gris que funciona como un margen para determinar las regiones de la imagen que pertenecen o no a la célula. El valor de superficie es un valor que se encuentra implícito en cada paquete de imágenes y que por lo general es un tono de gris que se encuentra en el margen de las secciones de la dendrita que se encuentran en foco (véase **Figura 3.3**). La correcta elección de este parámetro es crucial para que la reconstrucción sea coherente con la realidad.

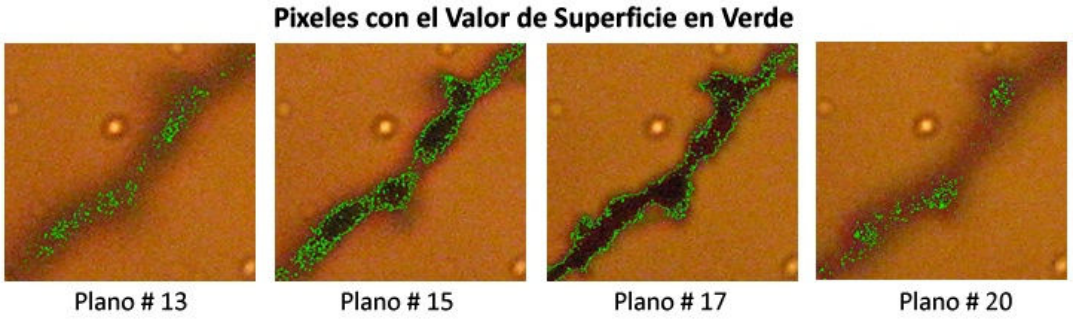

Figura 3.3 Diferentes planos de un mismo paquete de imágenes donde los pixeles con el tono de gris del valor de superficie están coloreados en verde

El valor de superficie es un parámetro muy difícil de determinar ya que depende de la cantidad de luz con la que se tomó la fotografía y de la cantidad de tinte fijado en la neurona. A pesar de esto este parámetro puede determinarse sin que el usuario se percate de ello, utilizando como base, los tonos de gris del fondo y los tonos de gris de las regiones en foco. Utilizando la ayuda del usuario se requiere que este seleccione al menos tres regiones donde solo haya fondo y tres regiones donde haya una dendrita en foco, usando esta información se calcula el valor de superficie que por lo general se encuentra a la mitad entre el valor mínimo de las regiones en foco y el valor mínimo de las regiones con fondo.

Como el valor de superficie es estimado usando una técnica indirecta, al finalizar el cálculo puede que no sea el adecuado es por ello que se permite al usuario cambiar el valor de superficie dentro de un rango seguro; con el propósito de mejorar el resultado de la reconstrucción.

3.1.5 Reconstrucción del paquete de Imágenes

Se investigaron e implementaron diversas estrategias para generar la reconstrucción tridimensional de la célula, de las cuales finalmente solo se utilizó una. Se abordó el problema de diferentes maneras, en un principio se pensó en utilizar un algoritmo de seguimiento de huellas digitales en 2D y adaptarlo a 3D; para que este identificara las estructuras dendríticas de la neurona, como si se tratara de las crestas de una huella dactilar. Esta estrategia fue presentada en el Encuentro Nacional de Ciencias de la Computación (CCMC-ENC 08).

Desafortunadamente esta aproximación no tuvo los resultados esperados, debido a que el algoritmo requería que las imágenes fueran binarizadas. Para binarizar las imágenes era necesario tratar las imágenes con una serie de filtros para imágenes digitales, cuyos resultados dependían de variables como la iluminación y la cantidad de tinte fijado en la neurona. El controlar dichos factores es poco práctico y difícil de llevar a cabo.

Después la investigación se enfocó a estudiar los algoritmos de reconstrucción utilizados en la tomografía computarizada y la resonancia magnética. El algoritmo más utilizado para generar reconstrucciones 3D a partir de datos de tomografía o resonancia era el Algoritmo de los Marching Cubes, ya que fue diseñado especialmente para generar reconstrucciones 3D a partir de este tipo de datos.

Cabe resaltar que las imágenes resultantes de una tomografía o una resonancia magnética no son más que una representación numérica en tonos de gris, en donde el tono de gris representa un nivel de absorción de radiación o la resonancia característica de un tejido en especial. Teniendo en consideración el origen numérico de las imágenes generadas por un tomógrafo o por máquina de resonancia magnética queda claro

que el algoritmo de los Marching Cubes fue diseñado para generar reconstrucciones a partir de tablas de medición axiales más no sobre imágenes axiales.

El Algoritmo de los Marching Cubes recibe ese nombre porque los datos tridimensionales son analizados usando un cubo que va recorriendo los datos y va evaluando cuáles se encuentran dentro de la estructura a graficar y cuáles no. Para realizar esto se evalúa si alguno de los vértices del cubo tiene un valor menor o igual al valor de superficie, lo que indica que esa parte del cubo se encuentra dentro de la estructura a graficar.

Se optó entonces por utilizar una versión modificada del algoritmo de los Marching Cubes, la cual a partir de un conjunto de imágenes binarizadas de la estructura celular utilizaba el concepto de cubo marchante para determinar los puntos donde la imagen binaria cambiaba de estado; una vez obtenidos los puntos estos se conectaban para formar triángulos usando una distancia constante como parámetro. Esta estrategia generaba modelos burdos y toscos con triángulos desordenados, además al igual que la estrategia anterior dependía del control de la iluminación y la cantidad de tinte en la neurona para que la binarización de la estructura celular tuviera éxito.

Debido a que la binarización de las imágenes no era factible se eligió una nueva estrategia que no requiriera que el paquete de imágenes fuera binarizado para conseguirlo se usó el Algoritmo de los Marching Cubes sin modificaciones, solamente se adaptó un poco la implementación para que pudiera funcionar usando las fotografías en escala de grises. A diferencia de las estrategias planteadas anteriormente, esta tuvo éxito en generar un modelo con una malla triangular más definida. La principal ventaja de esta estrategia plantea es que genera una reconstrucción 3D de la célula usando las imágenes originales en escala de grises sin binarizar la estructura del fondo; lo cual permite fotografiar a la neurona sin necesidad de controlar la iluminación o la cantidad de tinte en la muestra, descartando a estas variables como factores en la reconstrucción del modelo.

Evaluando qué vértices se encuentran dentro de la estructura y qué vértices no, se puede saber cuáles son las orillas del cubo que entran en colisión con la superficie, ya que uno de sus vértices estará en colisión mientras que el otro no. Usando interpolación sobre, todas las orillas que se encuentren en colisión, se genera un punto cuya posición concuerde con el de la superficie de la estructura. Y por último usando la configuración que poseen los vértices del cubo se determina la manera en la que deben de ser formados los triángulos, cuyos vértices serán los puntos que se han interpolado.

El Algoritmo de los marching cubes solo se adaptó para que en vez de leer datos de mediciones flotantes lo hiciera de los pixeles de las fotografías que son números enteros. Las reconstrucciones generadas a partir de datos de tomografía tienen una mayor calidad que las reconstrucciones hechas usando fotografías digitales, debido a que en una sección topográfica no existen planos fuera de foco mientras que en una fotografía es inevitable tener regiones fuera de foco.

Si el valor de superficie es elegido de manera correcta el algoritmo de los marching cubes es capaz de discernir entre las regiones que se encuentran en foco y las que no. Debido a que el tono de gris que se encuentra en el borde la estructura, que es el valor de superficie, va desapareciendo conforme la célula va perdiendo el foco véase **Figura 3.3**.

4. DISEÑO E IMPLEMENTACIÓN

En esta sección se analizarán los detalles sobre la implementación y el diseño del programa final. Se mencionarán aspectos técnicos sobre la implementación del algoritmo de los marching cubes responsable de generar la reconstrucción, también se mencionará la estrategia planteada para determinar el valor de superficie y la resolución real del paquete de imágenes, los cuales son parámetros necesarios para generar el modelo tridimensional y medirlo en micras. Además se mencionarán los aspectos de diseño del sistema y como es que las diversas entidades de software que lo conforman están relacionadas entre sí.

4.1 Aspectos globales de la Implementación del sistema

En un diagrama de flujo bastante simple (véase **Figura 4.1**), se pueden apreciar de manera bastante clara cuáles son los pasos que se siguen, empezando por abrir un paquete de imágenes hasta que se visualiza el modelo 3D de la neurona. Cabe señalar que a pesar de que el algoritmo de los marching cubes es parte esencial del sistema, existen otras rutinas bastante importantes como la de visualización del modelo o la rutina de obtención de parámetros que obtiene los parámetros necesarios para alimentar al algoritmo de los marching cubes. Además de la rutina para la medición a escala real del modelo.

Figura 4.1 Diagrama esquemático para la generación del modelo neuronal.

A continuación se mencionará brevemente que es lo que sucede en cada una de los procesos que se aprecian en la **Figura 4.1.**

- **Abrir Imágenes**.- Aquí el usuario indica en donde se encuentra el paquete de imágenes que desea procesar.
- **Obtención de Parámetros**.- En esta fase se leen o se calculan todos los parámetros necesarios para generar el modelo y medirlo.
- **Marching Cubes**.- Aquí se utilizan todos los parámetros obtenidos anteriormente, los cuales son usados para generar el modelo en tercera dimensión de la neurona.
- **Visualización del Modelo**.- Una vez generados los triángulos que definen al modelo, estos son enviados a la tarjeta de video como un arreglo de puntos, para que se visualicen en pantalla.
- **Medición del Modelo**.- Con el modelo en pantalla se proporciona la opción de poder obtener la longitud de cualquier área del modelo en micras.

Podemos ver claramente que el sistema tiene un diagrama de flujo lineal y simple, por lo tanto los errores que ocurran durante el proceso se detectarán de manera secuencial impidiendo al usuario continuar si el sistema no ha completado satisfactoriamente los pasos anteriores. El diseño de la interfaz del programa se apega a este flujo de datos, para tener un mejor control de los errores que pudieran ocurrir en cada fase.

Todos estos pasos son únicamente para generar el modelo, opcionalmente el usuario puede guardar el modelo generado en un archivo para su posterior visualización, sin tener que pasar de nuevo por cada uno de

estos pasos. Además el sistema cuenta con otra variedad de funciones que pueden usarse una vez que se ha generado el modelo, como lo son: medir la distancia entre dos puntos seleccionados, construir manualmente el árbol dendrítico de la neurona, sacar las longitudes de las ramas, cambiar el color de los objetos de la escena, rotar la escena, mover la cámara, etc.

4.2 Apertura de Paquetes de Imágenes.

En cuanto a la apertura de los paquetes de imágenes podemos mencionar que el usuario puede seleccionar entre una lista de imágenes o una carpeta en donde estas se encuentren. Los formatos aceptados por el programa son: BMP, JPG, TIFF y PNG, cualquier otro tipo de imagen, como GIF, que no esté soportado debe de ser convertido a un formato válido usando alguna herramienta de conversión.

El usuario puede elegir una lista de archivos usando un cuadro de diálogo para abrir archivo (*openFileDialog*) o en caso de que todos los archivos se encuentren en una carpeta se utiliza un cuadro de diálogo para abrir carpeta (*folderBrowserDialog*).

Durante esta fase solamente se extraen los nombres de los archivos de imagen y se guardan en una lista, no se realiza ninguna validación en formato de las imágenes ya que el manejo de imágenes en Visual Studio .NET tiende a ser bastante lento, incluso llega a ser mucho más lento que el mismo proceso de reconstrucción. Así que es responsabilidad del usuario garantizar que las imágenes sean del mismo tamaño y no estén dañadas. En caso de que exista algún error en la lista de imágenes el sistema controlará el error durante la fase de reconstrucción y detendrá la ejecución obligando al usuario a escoger un nuevo paquete de imágenes.

También es importante aclarar que todas las imágenes a pesar de estar a color son convertidas a escala de grises, ya que el algoritmo de reconstrucción no toma en cuenta el color para construir el modelo, además se obtienen mejores resultados si los componentes de color de las imágenes son promediados. Así que si el paquete de imágenes esta a color automáticamente las imágenes son convertidas a escala de grises cada vez que son cargadas a memoria.

Una vez que se tienen todos los nombres de las imágenes estos nombres se almacenan en una lista ligada en donde se ordenan por orden alfabético, ya que es necesario que los archivos de imagen se encuentren en orden antes de generar la reconstrucción. A pesar de que el sistema ordene la lista de imágenes en orden alfabético esto no garantiza que el contenido de las imágenes sea secuencial, es por eso que se le recomienda al usuario cuando realice la captura nombrar los archivos de manera secuencial.

4.3 Obtención de parámetros.

La siguiente fase es la de Obtención de parámetros, durante esta fase el usuario proporciona los parámetros necesarios para la reconstrucción, ya sea de manera directa o indirecta. Algunos de estos parámetros solamente son utilizados como referencia pero hay otros que son muy importantes, ya que del valor de estos parámetros depende el resultado de la reconstrucción.

Los parámetros que se necesitan para proceder con la reconstrucción del modelo son:

- La magnitud del objetivo del microscopio.
- La resolución en micras del paquete de imágenes
- El valor de superficie de ese paquete imágenes.

La magnitud del objetivo se utiliza solamente como referencia para saber con qué objetivo fueron tomadas esas imágenes. La resolución en micras se utiliza para calcular las distancias en micras que existen entre un punto y otro del modelo 3D. Por último el valor de superficie es un parámetro necesario para generar la reconstrucción de la neurona, que determina qué regiones de la fotografía forman parte de la célula y cuáles no.

4.3.1 Magnitud del Objetivo.

La magnitud del objetivo es un parámetro que debe de ser proporcionado directamente por el usuario, hace referencia a la magnitud del objetivo del microscopio con el que se obtuvieron las imágenes. El usuario debe de introducir directamente en una caja de texto, la única validación que se realiza es que el parámetro sea un número entero positivo entre 0 y 1000. El valor de este parámetro puede ser arbitrario mientras se ajuste a los parámetros de validación mencionados.

4.3.2 Valor de Superficie.

Uno de los parámetros más importantes es el valor de la superficie, debido a que es el responsable de discriminar en la fotografía qué regiones formarán parte del modelo y qué regiones no. El valor de la superficie es un valor que define que partes de la fotografía son parte de la neurona y qué regiones son fondo, partiendo del principio de que en las fotografías las neuronas siempre tienen un tono de gris más obscuro que el fondo que las rodea. Por lo tanto el valor de la superficie siempre será un tono de gris que se encuentre entre los tonos de gris del fondo y los tonos de gris de la neurona.

El valor de superficie no puede ser asignado arbitrariamente por el usuario, ya que la elección errónea de este parámetro generará problemas durante la fase de reconstrucción, generando deformidades en el modelo tridimensional. Además el valor de superficie al ser un tono de gris puede variar por un sin número de factores, como la iluminación del microscopio, la magnitud del objetivo, la resolución de la cámara, el tipo de cámara, la antigüedad de la muestra, etc. El valor de superficie debe de ser estimado tomando en cuenta todas estas variables, es por eso que se ha desarrollado una estrategia que ayude al usuario a elegir un valor de superficie adecuado para las variables específicas de ese paquete de imágenes.

La estrategia planteada para ayudar al usuario a elegir de manera correcta el valor de superficie requiere que se elijan al menos 3 regiones de la fotografía donde haya alguna dendrita en foco y 3 regiones en donde solo haya fondo, estas pueden ser elegidas de cualquier plano del paquete de imágenes.

Para seleccionar las regiones con una dendrita en foco el usuario selecciona un punto de un lado de la dendrita y otro punto que del otro lado de esta, de manera que se tenga una recta que cruce por la dendrita. En cuanto a las regiones que están fuera de foco el usuario debe de seleccionar una sección de recta de manera que no toque ninguna estructura dendrítica y marque una región solo con fondo. La manera de realizar la selección estos dos tipos de regiones se describe en la **Figura 4.2**.

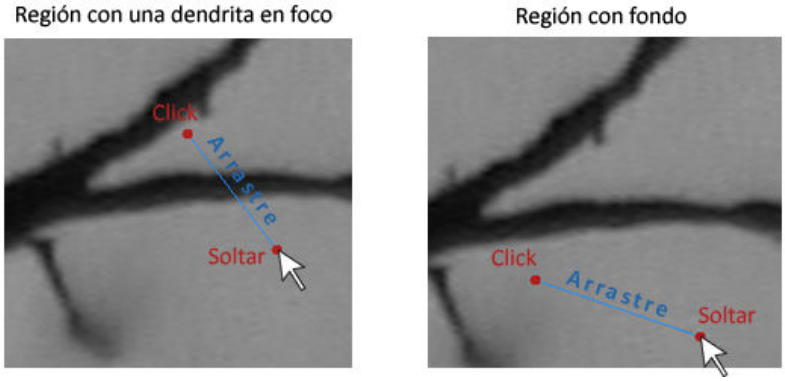

Figura 4.2 *A la izquierda* figura que ejemplifica la manera de seleccionar una dendrita en foco.
A la derecha figura que ejemplifica la manera de seleccionar una región con fondo.

Basados en el principio de que el valor de superficie se debe de encontrar en equilibrio entre los tonos de gris del fondo y los tonos de gris de la neurona, se debe de analizar qué ocurre con los tonos de gris en las regiones que fueron seleccionadas por el usuario. En la **Figura 4.3** se muestra una gráfica que representa los diversos tonos de gris que se encuentran en la línea marcada por el usuario, la cual se supone cruza a una dendrita que se encuentra en el plano de enfoque.

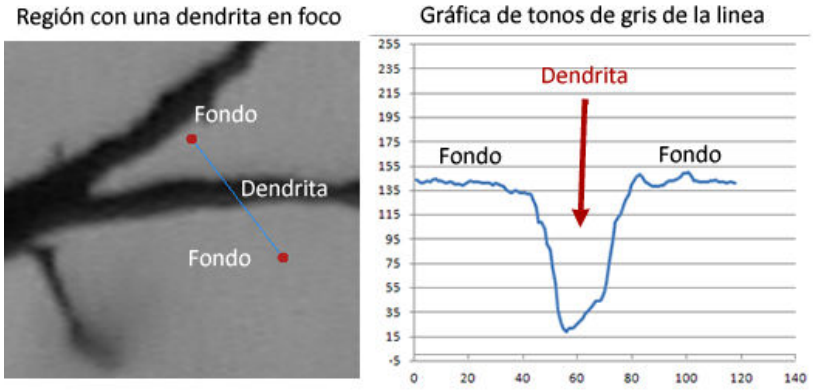

Figura 4.3 *A la izquierda* selección de una dendrita en foco.
A la derecha gráfico de los tonos de gris de la región seleccionada.

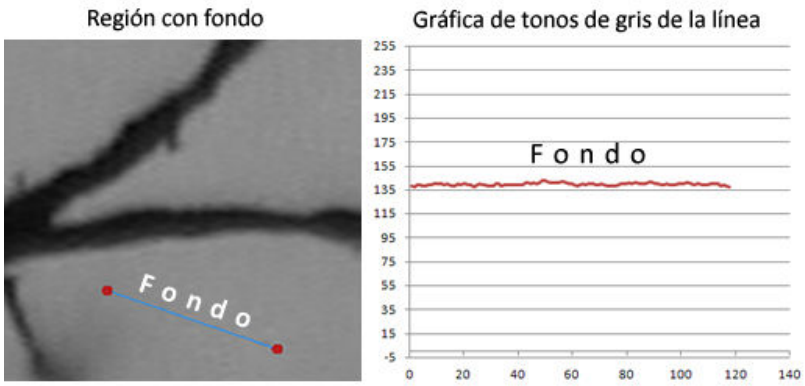

Figura 4.4 *A la izquierda* selección de una región con fondo.
A la derecha gráfico de los tonos de gris de la región seleccionada.

En la **Figura 4.4** se puede ver otra gráfica generada a partir de los tonos de gris contenidos en la línea marcada por el usuario, pero esta vez la línea no toca ninguna estructura dendrítica y solo marca una región vacía de la imagen. Comparando las dos gráficas la de la **Figura 4.3** y la de la **Figura 4.4** se aprecia que una tiene un bache de datos obscuros mientras que la otra no lo tiene, esto es debido a que en la **Figura 4.3** la línea cruza la dendrita que es un objeto más obscuro que el fondo y en la **Figura 4.4** no existe ninguna estructura neuronal alguna.

A partir de las gráficas anteriores se apreció que generalmente el valor de superficie adecuado se encuentra entre el valor más bajo de las regiones con dendrita y el valor mínimo de las regiones solo con fondo.

De lo anterior se dedujo una manera de estimar el valor de superficie. Primero se obtiene el mínimo absoluto de las tres regiones con dendrita y el mínimo de las tres regiones de fondo, así el valor de superficie es el valor que se encuentre en el medio de los mínimos. Está técnica nos ayuda a encontrar el valor de superficie que generalmente es el tono de gris que se encuentra en el margen de la dendrita, el cual se puede ver marcado en amarillo en la **Figura 4.5**.

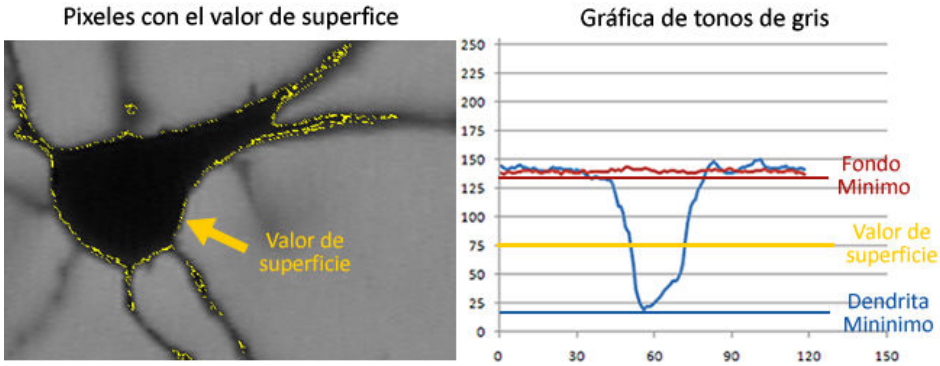

Figura 4.5 *A la izquierda* fotografía de una neurona donde los tonos de gris que coinciden con el valor de superficie se ven amarillo. *A la derecha* una gráfica que indica la posición del valor de superficie.

4.3.3 Resolución en micras.

Este parámetro es un conjunto de valores que nos indican la relación entre el pixel y la micra en los tres ejes de coordenadas, estos dependen directamente del tipo y la marca del microscopio, de la distancia que se mueve la platina entre toma y toma y también depende del objetivo usado para tomar las fotografías de la neurona.

Cómo se puede apreciar la resolución en micras de una tomografía es un parámetro bastante difícil de determinar, ya que depende de muchas variables. Además este parámetro es necesario para poder calcular las medidas en escala real del modelo en tercera dimensión, brindando al usuario una medición en micras del modelo generado.

Este es un parámetro que puede ser proporcionado directamente por el usuario indicando la relación entre pixeles y micras para ese paquete de imágenes, por ejemplo 1 pixel equivale a 0.25 micras o 150 pixeles equivalen a 50 micras. Además se debe de indicar la distancia en micras que existe entre imagen e

imagen, es decir la cantidad de micras que la platina se desplazó para después tomar la siguiente fotografía. Estos valores son introducidos en cajas de texto y validados para que sean números reales mayores a cero.

Debido a la dificultad de precisar la resolución en micras de la imagen y el desplazamiento en micras de la platina entre las diferentes tomas. Por esta razón el sistema cuenta con una forma de estimar estos parámetros, la cual se basa en el cálculo del diámetro en pixeles de una dendrita cuyo diámetro en micras se conoce o se estima. Por ejemplo el usuario nos indica la ubicación de una dendrita cuyo diámetro estima en 4 micras, entonces el sistema calcula el diámetro en pixeles de esa dendrita en 75 pixeles y un alto de 6 planos por lo tanto 4 micras equivalen a 75 pixeles y distancia de desplazamiento de la platina es de 0.66 micras.

En la **Figura 4.6** se puede ver el procedimiento usado para calcular la resolución en micras tanto en X,Y como en Z. La resolución en X,Y se obtiene midiendo el diámetro de una dendrita en pixeles determinando la distancia en la que la dendrita es más gruesa, utilizando el valor de superficie para saber dónde empieza y dónde acaba dicha dendrita. Mientras que la resolución en Z se estima contando el número de planos en los que la dendrita sigue apareciendo de manera nítida, para esto también se ocupa el valor de superficie.

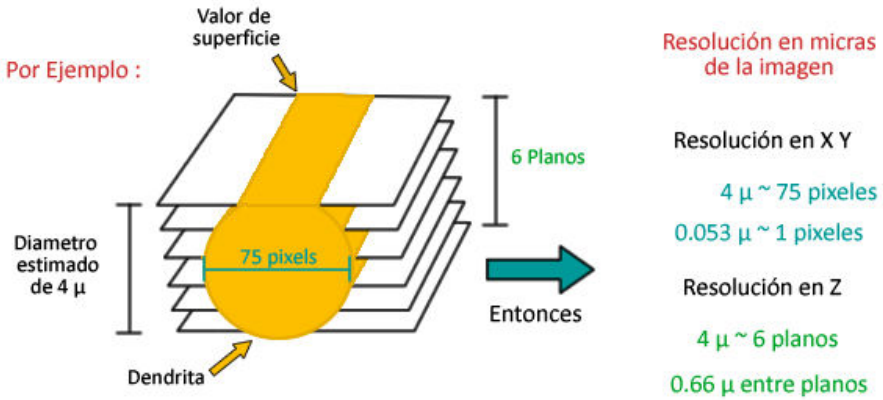

Figura 4.6 Diagrama que muestra el procedimiento para estimar la resolución en micras del paquete de imágenes

4.4 Implementación del Algoritmo de los Marching Cubes.

En general el algoritmo de los marching cubes necesita de dos datos para poder generar una reconstrucción 3D de cualquier objeto, primero necesita los datos originales que contienen al objeto que en nuestro caso son las series de imágenes del microscopio, con las cuales se cuenta desde el inicio. Y en segundo lugar necesita el valor de margen o límite del objeto que lo diferencia del fondo, el cual hay que calcular.

Partiendo del hecho que en la fotografía la neurona es mucho más obscura que el fondo y por lo tanto numéricamente posee tonos de gris mucho más bajos que el resto de la imagen, sería posible discriminar a la célula del fondo usando este criterio. El algoritmo de los marching cubes usa este criterio y establece un valor margen o límite llamado valor de superficie. Entonces cualquier dato que se encuentre por debajo del valor de superficie será considerado parte de la neurona, en caso contrario no se lo toma en cuenta, como se puede ver en la **Figura 4.7**.

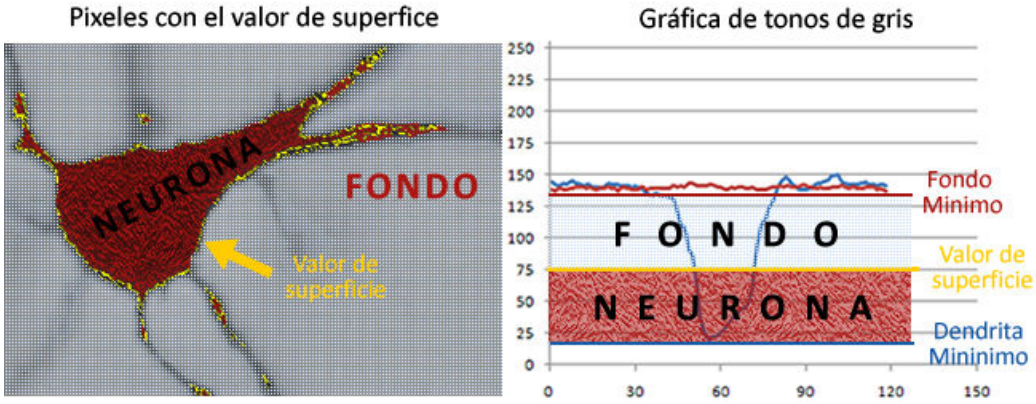

Figura 4.7 *A la izquierda* se muestra la diferenciación entre fondo y neurona usando el valor de superficie. *A la derecha* gráfica que muestra el tono de gris del valor de superficie, los tonos que están por debajo se consideran neurona mientras que los más altos se consideran fondo.

El valor de superficie es uno de los parámetros clave para que el algoritmo de los marching cubes logre una buena reconstrucción. Generalmente este valor es muy sencillo de determinar en datos de tomografía o resonancia magnética ya que cada tipo de tejido tiene un valor de resonancia o un índice de absorción de radiación específica y estos no varían entre individuos. En estos casos en particular el valor de superficie es muy fácil de determinar solo se debe de tomar un valor cercano al índice de absorción o al valor de resonancia.

Pero en nuestro caso en particular el obtener el valor de superficie es algo más delicado, porque se debe de escoger un tono de gris adecuado que se encuentre en el margen de la neurona en un plano nítido, para que sea de ayuda para eliminar los planos en los que la neurona está difusa. El valor de superficie puede variar por muchas razones como la iluminación, la resolución de la cámara, el microscopio, etc. Es por eso que el valor de superficie se obtiene usando la estrategia mencionada en la **Sección 4.3.2**.

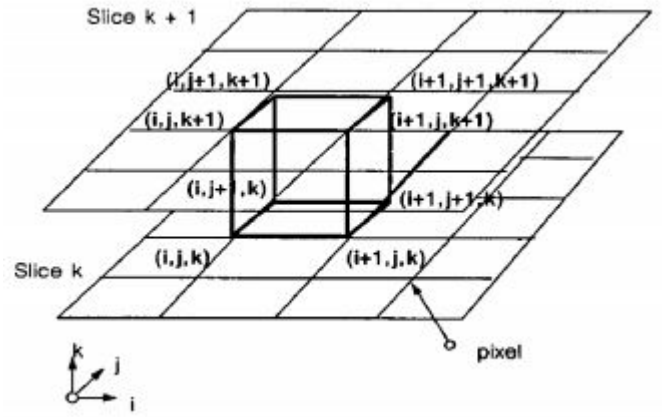

Figura 4.8 Diagrama del movimiento de un marching cube a través del paquete de imágenes

El algoritmo de los marching cubes usando un cubo imaginario de dimensiones variables va recorriendo los diferentes planos que contienen al objeto a modelar, como se puede ver en la **Figura 4.8**. De allí viene el nombre de marching cubes ya que este se va moviendo a través de los planos de manera secuencial, interpolando la posición de los triángulos que conforman el modelo.

Figura 4.9 Diagrama de flujo del Algoritmo de los Marching Cubes

El diagrama de flujo que se ve en la **Figura 4.9**, describe el algoritmo de los marching cubes. A grandes rasgos podemos ver que el algoritmo tiene 6 fases cada vez que el cubo se mueve, primero se evalúa si existen vértices dentro de la superficie a modelar, si es así dependiendo de los vértices que se encuentren dentro se obtienen los lados que interceptan a la superficie, después se interpolan los vértices de los triángulos y sus normales sobre los lados correspondientes y por último se guarda esta información en memoria o en archivo y se continúa con el siguiente cubo hasta terminar de recorrer el paquete de imágenes.

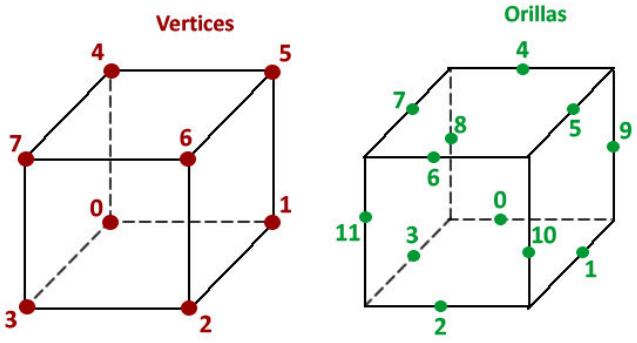

Figura 4.10 Índices de los vértices y las orillas del cubo

Cada uno de los vértices y las orillas del cubo son numeradas como se puede ver en la **Figura 4.10**, esta numeración nos ayuda a determinar que vértices se encuentran adentro y cuales afuera de la superficie. El algoritmo de los marching cubes dependiendo de cuales vértices se encuentren dentro y fuera de la superficie determina una configuración de triángulos específica para ese caso en particular, para facilitar la implementación del algoritmo se definen 16 casos base dependiendo de la configuración de los vértices

(véase **Figura 4.11**). El número de combinaciones posibles dependiendo si el vértice está afuera o adentro, es bastante grande. ($2^8 = 256$). Para codificar todas estas combinaciones se utilizan dos arreglos que tienen codificada la configuración de las orillas y de los triángulos, para cada configuración de vértices.

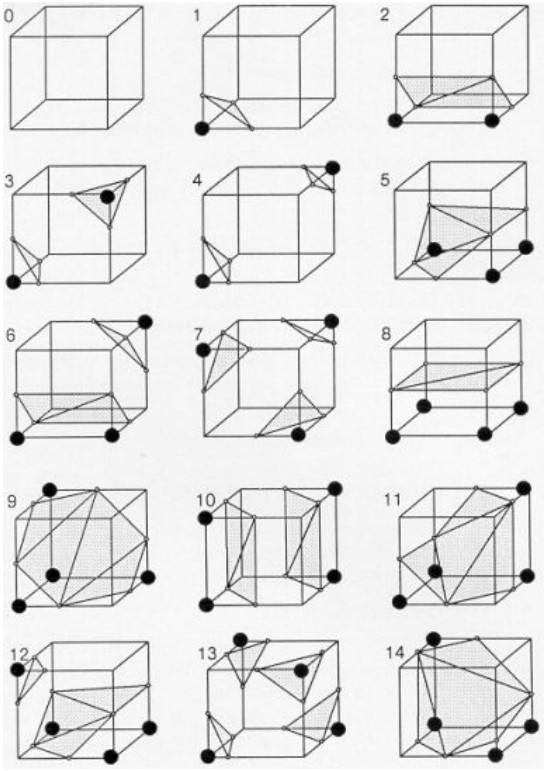

Figura 4.11 Diagrama que muestra los 16 tipos de facetas principales, cuya alineación depende de los vértices en colisión, el caso faltante es cuando todos los vértices se encuentran dentro la superficie.

Para el caso que se ve en la **Figura 4.13** tenemos que el vértice 3 se encuentra dentro de la superficie y por lo tanto las orillas 11, 3 y 2 son interceptadas por esta. Para evaluar si el vértice se encuentra o no dentro de la estructura se utiliza el valor de superficie, si el tono de gris para en el vértice del cubo está por debajo del valor de superficie, se considera que el vértice esta dentro de la estructura.

Cada vez que se determina que un vértice se encuentra dentro de la superficie dependiendo del índice del vértice se enciende un bit en una variable entera de 8 bits, a la cual llamaremos variable de índice. La variable de índice se utilizará para consultar la información codificada en los arreglos de orillas y de triángulos, ya que usando esta variable como el índice en los arreglos de orillas o de triángulos se puede consultar la información de las orillas y de las facetas de los triángulos, ya que toda la información fue ordenada de esa manera.

En el algoritmo de los marching cubes se usan dos arreglos, el arreglo de orillas y el arreglo de triángulos. Estos arreglos son declarados al inicio del programa y se usan como constantes, su objetivo es facilitar la implementación del algoritmo evitando el manejo convencional de las 256 configuraciones de facetas diferentes. El arreglo de orillas es un arreglo se enteros de 12 bits en donde cada variable tiene encendidos ciertos bits, que dependiendo de su posición representan las orillas sobre las cuales se debe de interpolar. Como en total existen 256 posibles combinaciones de orillas este arreglo tiene una longitud de 256, en donde cada registro representa una combinación específica de orillas de las 256 posibles (véase **Figura 4.12**).

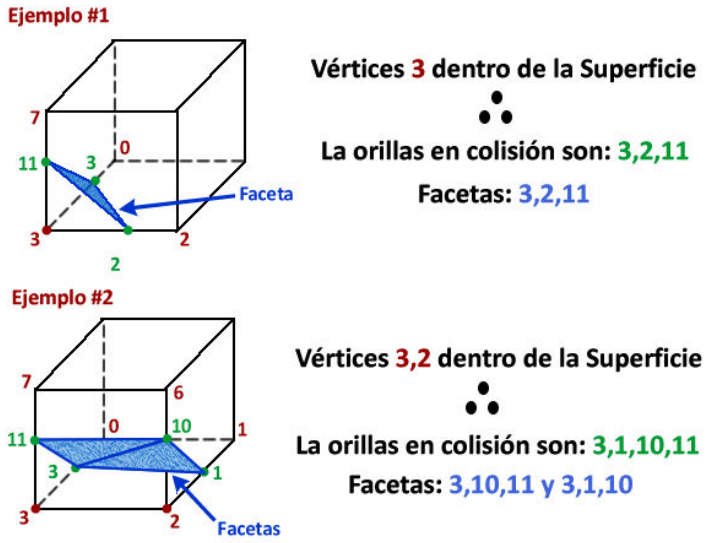

Figura 4.12 Dos ejemplos de marching cubes que demuestran que dependiendo de la configuración de los vértices dentro de la superficie, se tiene una configuración orillas y facetas diferentes.

En cuanto al arreglo de triángulos este es declarado como una matriz de variables enteras con 256 x 16 registros, en cada uno de los 256 registros se encuentra un arreglo de 16 enteros en donde cada variable entera representa el índice de una orilla del cubo sobre la cual se encuentra el vértice de un triángulo, estos se encuentran definidos en orden usando la regla de la mano derecha.[1]

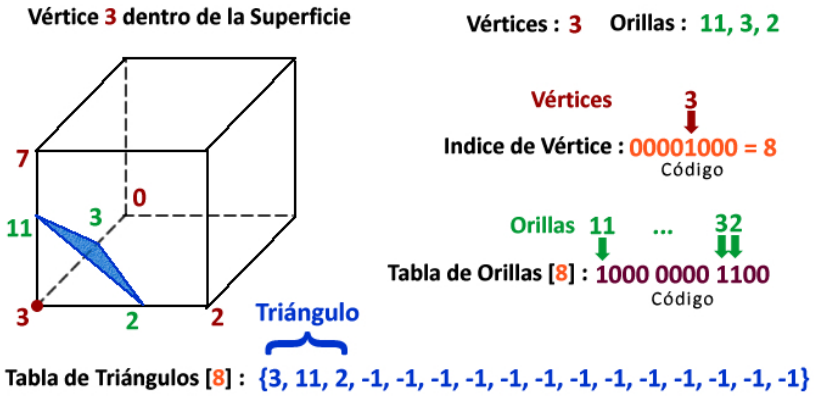

Figura 4.13 Diagrama de una corrida de un cubo marchante.

Ya que se han determinado qué vértices del cubo están dentro del modelo y se ha guardado esa información en la variable de índice. Después se utiliza el arreglo de orillas para obtener los índices de las orillas sobre las cuales estarán los vértices de los triángulos, para esto se usa la variable de índice como índice en el arreglo de triángulos para obtener la configuración de facetas triangulares para ese caso en particular. (Véase **Figura 4.13**).

[1] Para consultar las tablas de orillas y triángulos refiérase al Apéndice B

$$v = v_1 + \mu * (v_2 - v_1)$$ **Ecuación 4.1**

dónde:

v Es la posición del vértice interpolado
v_1 Es la posición del primer vértice de la orilla
v_2 La posición del segundo vértice de la orilla

$$\mu = \frac{isolevel - tono|v_1|}{tono|v_2| - tono|v_1|}$$ **Ecuación 4.2**

$isolevel$ Es la variable de superficie
$tono$ Tono de gris del vértice correspondiente

Conociendo las orillas que son interceptadas por la superficie se realiza una interpolación sobre estas, para así obtener los vértices de los triángulos correspondientes. Esta interpolación se realiza usando la posición de los vértices del cubo, los tonos de gris de la imagen y las dimensiones del cubo, usando la fórmula que se ve en la **Ecuación 4.1** y que a su vez hace uso de la **Ecuación 4.2**. Y usando los mismos parámetros se realiza la interpolación de las normales de estos puntos, como se ve en las **Ecuaciones 4.3** y **4.4**, las cuales son necesarias para el manejo de iluminación del modelo.

$$G_x(i,j,k) = \frac{tono(i+1,j,k) - (i-1,j,k)}{\Delta x}$$

$$G_x(i,j,k) = \frac{tono(i,j+1,k) - (i,j+1,k)}{\Delta y}$$ **Ecuación 4.3**

$$G_x(i,j,k) = \frac{tono(i,j,k+1) - (i,j,k+1)}{\Delta z}$$

dónde:

$G(i,j,k)$ Es la normal del vértice en (i, j, k)
$\Delta x\ \Delta \mathrm{y}\ \Delta \mathrm{z}$ Son las dimensiones del cubo.
$tono$ Tono de gris del vértice correspondiente

Fórmula de interpolación:

$$n = \overrightarrow{g_1} + \mu(\overrightarrow{g_2} - \overrightarrow{g_1})$$ **Ecuación 4.4**

El método para obtener la posición de los vértices es una simple interpolación lineal que usa la posición de los vértices y sus tonos de gris para calcular las coordenadas de un punto sobre todas las orillas que atraviesan la superficie. Se pueden utilizar interpolaciones de mayor grado esperando mejores resultados, pero en base al trabajo reportado por Lorensen y Cline se sabe que utilizar métodos de mayor grado solo

mejora un poco los resultados obtenidos, así pues se optó por usar una estrategia lineal para promediar la posición de los vértices.

El método para interpolar las normales se basa en el hecho de que una superficie con gradiente constante también tendría una densidad o un tono de pixel constante, usando este hecho podemos estimar el gradiente del vértice usando una resta centralizada con la densidad del tono de gris de la imagen, en los vértices posteriores y anteriores al vértice del cual deseamos conocer la normal. Una vez que se tiene el valor de las normales de los dos vértices de la orilla, se usan para interpolar la normal del vértice que se encuentra en dicha orilla. Este método de interpolación obtiene la normal de la superficie tal como lo haría una derivada $\vec{g}(x, y, z) = \nabla \text{tono}(x, y, z)$. Lo cual mejora considerablemente la calidad en la iluminación del modelo (véase **Figura 4.14**).

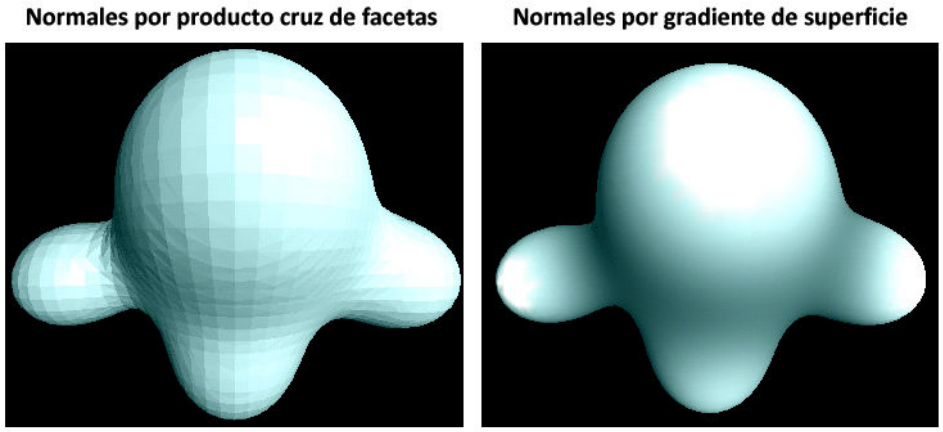

Figura 4.14 Comparación de resultados entre el método implementado *derecha* el cual calcula la normal sobre la superficie y el método usual para calcular normales *izquierda,* el cual calcula la normal sobre los vértices de las facetas.

Una vez interpolados todos los vértices sobre las orillas señaladas por el arreglo de orillas, estos se deben de ligar y ordenar de manera que formen los triángulos correspondientes a esa faceta. Para esto se usa la tabla de triángulos que posee el orden y la configuración de los triángulos para ese caso en particular. El primer índice de la matriz, el de 256 registros, contiene los 256 casos posibles, mientras que el segundo índice contiene el número que identifica a cada orilla. Es decir que si el registro tiene la forma: `tabla_triangular[3][] = {1, 8, 3, 9, 8, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}`, esto significa que los vértices de las orillas `(1, 8, 3)` forman un triangulo, y las de `(9, 8, 1)` forman otro, la ausencia de más vértices se indica con `-1` (véase **Figura 4.13**).

Cada vez que el cubo se mueve a través de la muestra, la información obtenida de los vértices de los triángulos y sus normales es almacenada en un archivo de texto, para que después esta pueda ser visualizada. La principal razón por la cual las coordenadas de las facetas y sus normales son almacenadas en un archivo, es para ahorrar el consumo de memoria RAM, lo que también permite generar modelos a partir de imágenes digitales grandes, de más de 8 megapixeles.

4.5 Graficado de vértices y normales.

Como el modelo puede llegar a estar formado por millones de vértices es necesario utilizar técnicas especializadas para la graficación de los datos. Existen varias técnicas para realizar este tipo de trabajo como lo son: las listas de despliegue, buffer de vértices de objetos o arreglo de vértices. Cada una de estas técnicas

presenta sus ventajas y sus desventajas, se optó por usar arreglos de vértices ya que permite generar las coordenadas de los vértices en tiempo de ejecución y además se obtiene un rendimiento adecuado para la visualización fluida de modelos con una gran cantidad de polígonos.

Durante el ciclo cada vez que el cubo se mueve los datos con la posición de los vértices y los valores de sus normales se deben de almacenar ya sea en archivo o en memoria, para que después puedan ser graficados usando OpenGL. Debido a la gran cantidad de puntos y normales que se pueden generar esta información se va guardando en un archivo, para evitar un uso excesivo de memoria RAM. El formato en el que se guardan estos puntos depende totalmente del programador, en este caso en particular las tres coordenadas junto con las tres coordenadas de cada normal se guardan en una línea de un archivo de texto, así tres líneas consecutivas indican las coordenadas y normales de un triangulo.

Una vez que se ha finalizado de recorrer toda la muestra se tendrá un archivo con los vértices de cada uno de los triángulos junto con sus respectivas normales. Esta información se grafica usando una funcionalidad de OpenGL llamada "Arreglos de Vértices", esta propiedad de OpenGL permite enviar una arreglo de vértices directamente a la memoria de la tarjeta de video, evitando así la continua transferencia de datos entre la memoria RAM y la memoria de video; provocando un mejor rendimiento en la visualización en pantalla de los datos.

Para implementar los arreglos de vértices primero se deben de declarar los arreglos usando un tipo de datos aceptado por OpenGL, en este caso tanto el arreglo de vértices como el de normales son declarados como flotantes. Para indicar a OpenGL que vamos a utilizar arreglos de vértices debemos de activar el servicio llamando a **`glEnableClientState()`** y al terminar de usar el servicio se debe de desactivar con **`glDisableClientState()`**, tal como se muestra a continuación.

```
Gl.glEnableClientState ( Gl.GL_VERTEX_ARRAY );
Gl.glEnableClientState ( Gl.GL_NORMAL_ARRAY );

//Código para graficar el arreglo de vértices y de normales.

Gl.glEnableClientState ( Gl.GL_VERTEX_ARRAY );
Gl.glEnableClientState ( Gl.GL_NORMAL_ARRAY );
```

A continuación se indican las dimensiones y la localización de los arreglos llamando a las funciones **`glVertexPointer()`** y **`glNormalPointer(),`** tal como se ve más abajo. En el primer argumento de **`glVertexPointer()`** se indica la cantidad de coordenadas que definen al vértice que puede ser un número de 2 hasta 4, el segundo argumento indica el tipo del arreglo, el tercer argumento indica el índice de partida a partir del primer elemento del arreglo y el último argumento es el apuntador al primer elemento del arreglo; en este caso como se está usando C# se indica con el objeto en sí.

```
Gl.glNormalPointer ( Gl.GL_FLOAT, 0, normales );
Gl.glVertexPointer ( 3, Gl.GL_FLOAT, 0, puntos );
Gl.glDrawArrays ( Gl.GL_TRIANGLES, 0, max * 3 );
```

En cuanto a la función **`glNormalPointer()`**, se utiliza para indicar la localización y las dimensiones del arreglo de normales. El primer argumento indica el tipo del conjunto de datos, el segundo argumento es el índice de partida a partir del primer elemento del arreglo y por último el tercer argumento es el apuntador al arreglo de normales, que en este caso en particular por estar usando C# se indica con el objeto en sí, como se ve arriba.

Por último la **glDrawArrays()** es la responsable de enviar los arreglos de vértices y de normales a la memoria de video para su graficación. En el primer argumento de esta función se indica el tipo de polígono a trazar, en el segundo el índice del primer elemento a graficar y por último en el tercer argumento se indica el número de primitivas que se van a graficar.

```
for (int i=0; i < trian.Count; i++)
{
      Gl.glBegin(Gl.GL_TRIANGLES);
      Gl.glVertex3f(train[i].v1x, train[i].v1y, train[i].v1z);
      Gl.glVertex3f(train[i].v2x, train[i].v2y, train[i].v2z);
      Gl.glVertex3f(train[i].v3x, train[i].v3y, train[i].v3z);
}
```

Puede apreciarse que al utilizar esta funcionalidad ya no es necesario mandar cada primitiva una a una a la memoria de video como se ve arriba, sino que en un solo paso se envía toda la información necesaria para graficar el modelo lo cual aumenta de manera considerable el rendimiento incrementando la cantidad de cuadros por segundo que son presentados en pantalla.

Una vez que el modelo ha sido desplegado en pantalla el usuario tiene la opción de girar el modelo sobre cualquiera de sus ejes, de cambiar su tamaño en cualquiera de sus dimensiones y de cambiar el color con el que se muestra el modelo.

4.5.1 Cámara.

También es importante mencionar la manera en la que se manejó la cámara para visualizar el modelo. El tipo de cámara implementada está basada en un modelo de primera persona, en donde se trata de simular el campo de visión de una persona, ubicando la cámara en la cabeza de un personaje que va recorriendo la escena. Este modelo de visión también es conocido por el nombre de cámara sintética y es muy utilizado en la industria de los videojuegos; para simular el campo de visión de un personaje virtual.

Este tipo de cámara tiene 5 grados de libertad por lo tanto existen 5 parámetros que se deben de controlar para este modelo de visión, los cuales son la posición en *X,Y* y *Z,* el ángulo de rotación plano al cual llamaremos *yaw* y por último el ángulo de rotación alto al cual llamaremos *yawh.* Los primeros tres parámetros definen la posición en la que se encuentra la cámara, mientras que el *yaw* determina hacia donde se encuentra mirando la cámara sobre los ejes *X,Z* mientras que el *yawh* determina hacía donde apunta en el eje *Y*. Las matemáticas necesarias para controlar este tipo de visión no son muy complejas, debido a que solo se usan cuestiones básicas de trigonometría para calcular el ángulo de visión de la cámara y el punto hacia donde ve (véase **Figura 4.15**).

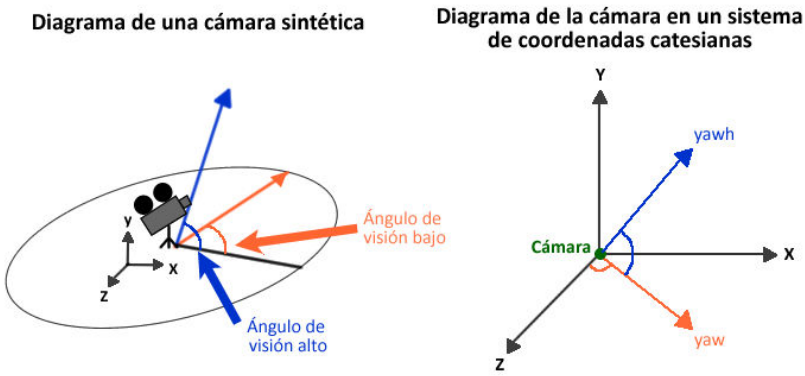

Figura 4.15 *A la izquierda* diagrama que muestra las diversas funciones de una cámara de primera persona. *A la derecha* diagrama en el espacio cartesiano que muestra los 5 grados de libertad de la cámara.

Primero se necesita aclarar que esta cámara no puede ser implementada usando un modelo de visión ortogonal, es necesario que el modelo de visión sea de proyección, ya que se utiliza la instrucción **`gluLookAt()`** que en OpenGL solo se encuentra disponible en el modelo de visión por proyección. Mediante la instrucción **`gluLookAt()`** se modifican los parámetros de posición, visión y el vector de orientación de la cámara. Los cuales deben de ser calculados y refrescados cada vez que el usuario desencadena un evento para moverla.

Para mover la cámara se usan las flechas del teclado arriba y abajo que trasladarán la cámara hacia adelante y hacia atrás, mientras que las teclas derecha e izquierda rotarán el ángulo de visión plano; la rueda del ratón se utilizará para mover la cámara hacia arriba o hacia abajo sobre el eje *Y*. Por último usando el arrastre del ratón a través de la ventana de renderizado se pueden definir tanto el ángulo de visión plano como el ángulo visión alto, para redefinir el punto de visión de la cámara (look-at-point).

Para precisar la velocidad con la que se mueve la cámara es necesario definir varias constantes que controlen la velocidad de traslación y rotación de la cámara. A estas constantes las que llamaremos SPEED, ROTSPEED, MOUSEROTSPEED estas ayudarán a definir una tasa de cambio cada vez que el usuario desencadena un evento que mueva la cámara ya sea en su posición o en su ángulo de visión.

Como se puede apreciar en las **Ecuaciones 4.5, 4.6, 4.7** y **4.8** cada vez que el usuario realiza un evento que cambie el campo de visión de la cámara se incrementan o decrementan varios contadores, a un ritmo definido por las constantes mencionadas anteriormente. Estos contadores son necesarios para registrar la posición de la cámara y el valor de su ángulo de visión plano (*yaw*).

$$\boldsymbol{Adelante} \quad dz = SPEED * 1.0 \qquad \textbf{Ecuación 4.5}$$

$$\boldsymbol{Atras} \quad dz = SPEED * -1.0 \qquad \textbf{Ecuación 4.6}$$

$$\boldsymbol{Dererecha} \quad yaw -= ROOTSPEED * 1.0 \qquad \textbf{Ecuación 4.7}$$

$$\boldsymbol{Izquierda} \quad yaw += ROOTSPEED * 1.0 \qquad \textbf{Ecuación 4.8}$$

Para definir tanto el ángulo de giro como la posición de la cámara es necesario conocer el valor del *yaw* o ángulo de rotación plano. Este ángulo de visión plano nos ayuda a determinar hacia donde se encuentra mirando la cámara y hacia adonde tiene que avanzar la cámara sobre las coordenadas *x* y *z*. Para entender cómo se calcula el *yaw* es necesario usar un diagrama básico de trigonometría, véase **Figura 4.16**.

Figura 4.16 Trigonometría básica para determinar el incremento en los componentes *x* y *z*, tanto para la posición de la cámara como para determinar su punto de visión.

En el triángulo que se muestra en la figura lo único que se conoce es el ángulo de apertura del triángulo, que en este caso sería el *yaw*. Lo que se necesita calcular es la longitud de los lados del triángulo que determinarán el incremento en *x* y *z,* lo cual es trivial y consiste en multiplicar la constante de velocidad por el seno o coseno de la coordenada en cuestión como se muestra en las **Ecuaciones 4.9** y **4.10**; suponiendo que la hipotenusa del triangulo es 1.

Posición sobre los ejes *x* y *z*:

$$pos_x += dz * \cos(yaw) \qquad \textbf{Ecuación 4.9}$$

$$pos_z += dz * \sin(yaw) \qquad \textbf{Ecuación 4.10}$$

También es posible cambiar la posición de la cámara sobre el eje *y*. El cambio de posición de la cámara sobre este eje no depende de *yaw* ni de *yawh*, así que para definirlo simplemente se incrementa su valor con otro contador; pero usando la misma tasa de cambio para la posiciones en *x* y *z*. (véanse **Ecuaciones 4.11, 4.12** y **4.13**)

$$\textbf{\textit{Arriba}} \quad dzh = SPEED * 1.0 \qquad \textbf{Ecuación 4.11}$$

$$\textbf{\textit{Abajo}} \quad dzh = SPEED * -1.0 \qquad \textbf{Ecuación 4.12}$$

Posición sobre el eje *y*:

$$pos_y = dzh \qquad \textbf{Ecuación 4.13}$$

Una vez calculada la posición de la cámara solo resta determinar hacia qué punto se encuentra mirando (look-at-point), para determinar este punto se usan los conceptos trigonométricos mencionados anteriormente véase **Figura 4.16**. Intuitivamente la cámara siempre debería de estar mirando hacia al frente sobre el ángulo de visión plano. Si se conoce la posición de la cámara simplemente se tiene que definir un punto al frente y sobre el *yaw* (véanse **Ecuaciones 4.14** y **4.15**)

Punto de Visión en ejes *x* y *z*:

$$look_x = posx + \cos(yaw) \qquad \textbf{Ecuación 4.14}$$

$$look_z = posz + \sin(yaw) \qquad \textbf{Ecuación 4.15}$$

Lo anterior solamente describe los procedimientos para implementar una cámara con tan solo 4 grados de libertad. Para implementar el grado faltante es necesario definir el valor del ángulo de visión alto, para incrementar o decrementar el valor de este ángulo se utiliza el arrastre del mouse si el arrastre del mouse es hacia arriba se incrementa el valor de la variable y se decrementa en caso contrario (véanse **Ecuaciones 4.16** y **4.17**).

$$\textbf{\textit{Arriba}} \quad yawh \mathrel{+}= MOUSEROOTSPEED * 1.0 \qquad \textbf{Ecuación 4.16}$$

$$\textbf{\textit{Abajo}} \quad yawh \mathrel{-}= MOUSEROOTSPEED * 1.0 \qquad \textbf{Ecuación 4.17}$$

El ángulo de visión alto define la coordenada *y* del punto de visión, lo cual permite que apuntar la cámara hacia arriba o hacia abajo. El cálculo del *yawh* se basa en el mismo principió trigonométrico mencionado en la **Figura 4.16**, sumando la posición de la cámara con respecto al eje *y* con el resultado del *cos(yawh)* (véase **Ecuación 4.18**).

Punto de Visión en el eje *y*

$$look_y = pos_y + cos(yawh) \qquad \textbf{Ecuación 4.18}$$

Ya que se han calculado tanto la posición de la cámara con respecto al ángulo de visión plano y las tres coordenadas del punto de visión (look-at-point), se introducen cómo parámetros de la función **`gluLookAt(),`** como se ve abajo. Esa instrucción redefinirá la transformación de visión equivalente conforme a los parámetros suministrados

```
Glu.glulookat(posx, posy, posz, lookx, looky, lookz, 0.0f, 1.0f, 0.0f);
```

La implementación de esta cámara con 5 grados de libertad permitirá al usuario visualizar cualquier aspecto del modelo 3D a cualquier distancia y en cualquier ángulo.

4.6 Medición del modelo

Uno de los requisitos planteados por el laboratorio, fue el de medir las ramificaciones de la estructura dendrítica de la neurona hasta un tercer nivel de profundidad, ya que dicha información es utilizada para comprobar o refutar la hipótesis de la investigación que se esté llevando a cabo. Desafortunadamente la implementación de esta funcionalidad va más allá de los alcances planteados inicialmente para este trabajo de tesis. Pero a pesar de ello se ha implementado una herramienta, parecida a una regla, que le permite al investigador medir el modelo tridimensional a escala real, es decir se pueden tomar mediciones de cualquier región del modelo y recibir un resultado de su escala real en micras.

Es necesario aclarar que la herramienta de medición desarrollada es bastante básica parecida una regla común. Lo único que se debe hacer para usarla es posicionar dos esferas en cualquier posición del

espacio tridimensional, lógicamente una esfera indica la posición inicial y la otra el final de dicha medición. Dichas esferas pueden ser seleccionadas por el usuario haciendo click sobre ellas, para poder moverlas de posición y así ubicar la regla en la posición deseada.

Para realizar la selección de las esferas usando un click del ratón se necesitan utilizar varias funcionalidades que la API de OpenGL proporciona para ello. Esta funcionalidad de OpenGL es usualmente llamada picking o pitch.

El seleccionar un objeto dentro de una escena de OpenGL es una operación que se realiza en varios pasos. Previamente se identifican al objeto dentro de la escena usando el stack de nombres, una vez que el usuario hizo click se obtienen las coordenadas del click dentro de la ventana, entra en modo de selección y se redefine el campo visión a un cuadro pequeño alrededor del cursor, después se renderiza la escena entera y por último se sale del modo de selección y se identifican los objetos que se renderizaron en ese pequeño campo de visión, consultado el buffer de selección. La **Figura** 4.17 muestra la secuencia de pasos para realizar la selección.

Figura 4.17 Diagrama que muestra en proceso completo para llevar a cabo la selección de un objeto en una escena de OpenGL, usando el modo de renderizado de selección.

Antes que nada debemos de identificar con un nombre único a cada elemento de la escena que deseemos seleccionar; para esto se utiliza el stack de nombres de OpenGL. Primero la pila de objetos debe de ser inicializado usando la función **`glInitNames()`**, para después introducir un número entero al stack que identifique de manera única al objeto definido. Abajo se muestra un código de ejemplo para identificar un conjunto de primitivas usando la pila de nombres.

```
Gl.glInitNames();

Gl.glPushName(1);
 dibujarCuerpo();
Gl.glPopName();

Gl.glPushName(2);
 dibujarCabeza();
 dibujarOjos();
Gl.glPopName();
```

Una vez identificadas las primitivas que deseamos seleccionar, debemos de capturar el evento de click sobre la ventana. Al producirse el evento se debe de inicializar el buffer de selección usando la instrucción **`glInitNames().`** Después se debe de guardar el estado actual de la matriz, usando la función **`glGetIntegerv(GL_VIEWPORT, view)`**, para poder redefinir el campo de visión en el modo de

selección. Posteriormente se pasa al modo de selección usando la instrucción **`glRenderMode(GL_SELECT)`** y se define un campo de visión nuevo como un cuadrado pequeño al alrededor de la región donde el usuario hizo click.

Ya en el modo de selección y con un campo de visión reducido, se renderiza nuevamente la escena completa. Si el objeto aparece renderizado en este pequeño campo de visión, significa que el usuario hizo click sobre este, cuando esto pasa OpenGL internamente guarda en el buffer de selección el nombre asociado al objeto y sus niveles de profundidad; los cuales indican que tan cerca o lejos se encuentra el objeto del campo de visión. El buffer de selección tiene una estructura como la que se muestra en la **Tabla** 4.1.

Buffer de Selección

Contenido	Descripción
0	Ningún nombre almacenado para el primer click.
4.2822e+009	Profundidad mínima del primer click
4.28436e+009	Profundidad máxima del primer click
1	Número de nombres almacenado para el segundo click
4.2732e+009	Profundidad mínima del segundo click
4.27334e+009	Profundidad máxima del segundo click
6	Nombre del objeto registrado para el segundo click
2	Número de nombres almacenado para el tercer click
4.27138e+009	Profundidad mínima del tercer click
4.27155e+009	Profundidad máxima del tercer click
2	Nombre del primer objeto registrado para el tercer click
5	Nombre del segundo objeto registrado para el tercer click

Tabla 4.1 Describe el contenido del Buffer de Selección que contiene en su el registro de 3 clicks. La información correspondiente a cada click se muestra con un color diferente.

Finalizado el renderizado de la escena nuevamente es restaurado el modo de renderizado normal usando la función **`hits = glRenderMode(GL_RENDER),`** la cual nos devolverá un valor entero que indica el número de elementos que se agregaron al buffer de selección, mientras el modo de selección estuvo activado. Posteriormente la información que se halla dentro del buffer de selección debe de ser analizada para determinar sobre cuales objetos se hizo click.

Las esferas que componen la regla de medición son seleccionadas con un click de ratón mediante la técnica mencionada anteriormente. Posteriormente la posición de la esfera seleccionada puede modificarse usando las flechas del teclado. La selección de las esferas pudo haberse implementado usando un método más sencillo, pero se decidió hacerlo así para que la herramienta fuese más intuitiva.

Las esferas indican el inició y el final de la medición, por lo tanto si se conoce la posición de esas esferas en la escena se puede calcular la distancia en pixeles que existe entre ellas, la cual se calcula como se muestra en la **Ecuación 4.19**. Una vez que se tienen esas distancias se las debe de comparar con los factores de escala real que fueron calculados anteriormente, usando una conversión de proporcionalidad lineal (regla de tres), como se muestra en la **Ecuación 4.20**.

$$dist_x = |esf1_x - esf2_x|$$
$$\boldsymbol{dist_y = |esf1_y - esf2_y|} \quad \textbf{Ecuación 4.19}$$
$$\boldsymbol{dist_z = |esf1_z - esf2_z|}$$

$$dist\mu_x = \frac{(dist_x * micras_x)}{(pixeles_x)}$$
$$\boldsymbol{dist\mu_y = \frac{(dist_y * micras_y)}{(pixeles_y)}} \quad \textbf{Ecuación 4.20}$$
$$dist\mu_z = \frac{(dist_z * micras_z)}{(pixeles_z)}$$

Una vez que se efectúa la conversión, de pixeles a micras, de la distancia entre las dos esferas; cálculo que fue realizado usando los factores de resolución de las fotografías. Se calcula la distancia a escala real existente entre los dos marcadores usando las longitudes ya convertidas a micras, como se aprecia en la **Ecuación 4.21**.

$$dist\mu = \sqrt{(dist\mu_x)^2 + \left(dist\mu_y\right)^2 + (dist\mu_z)^2} \quad \textbf{Ecuación 4.21}$$

El resultado le es presentado al usuario en una cada de texto, y el cálculo es realizado cada vez que el usuario mueve alguna de esferas a través de la escena.

5. RESULTADOS

En el presente se mencionarán todas las características que ofrece el software desarrollado. También se presentarán los resultados de las reconstrucciones obtenidas mencionando el tiempo de procesamiento, las características de las imágenes originales, las características del modelo junto con su respectiva captura de pantalla. También se presentarán diagramas descriptivos de la interface del programa y los resultados arrojados por la herramienta de medición desarrollada.

5.1 Muni

El programa desarrollado recibe el nombre de Muni, la cual es una palabra en sanscrito que quiere decir monje, gurú, ermitaño o sabio. El software recibe este nombre debido a que los monjes tibetanos conocen los secretos de la mente humana desde hace más de 4000 años, a pesar de que no saben que es una neurona; mientras que los científicos occidentales siguen tratando de descifrar su funcionamiento.

Usemos esta analogía si la mente humana fuera un auto de Formula 1 los científicos serían los mecánicos sabrían como darle mantenimiento y arreglar el auto, mientras que un gurú tibetano sería el piloto capaz de obtener un máximo rendimiento; experimentando la sensación de llevar el auto hasta el límite. Es por eso que el programa se llama Muni rindiendo honor a todos esos hombres sabios, que a pesar de no entender el funcionamiento del cerebro humano son maestros en el control de la mente. En la **Figura 5.1** se puede ver la ventana de introducción del software que tiene un diseño alusivo al nombre del programa.

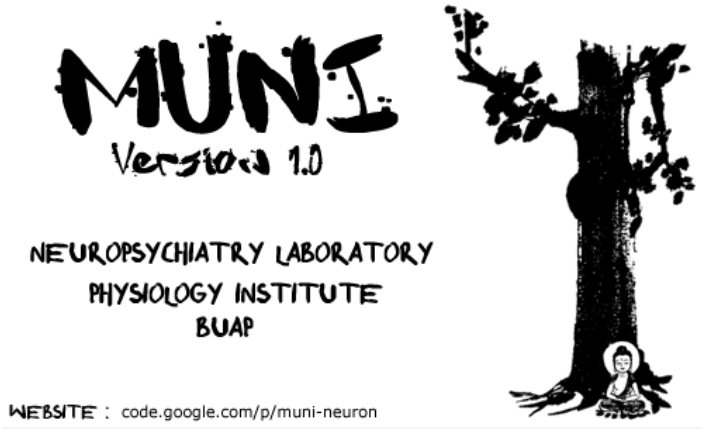

Figura 5.1 Ventana de introducción de Muni, el cual es el nombre del programa desarrollado.

5.2 Características

En cuanto a las características, establecidas por los miembros de laboratorio, que el sistema desarrollado debería cumplir se mencionan:

- Generar una reconstrucción 3D de una neurona a partir de un conjunto de fotografías axiales.
- Visualizar el modelo generado desde cualquier perspectiva.
- Medir la longitud de las ramificaciones dendríticas hasta en un tercer nivel.

- Que sea amigable con el usuario y que requiera poca capacitación.

Este trabajo de tesis se enfoca en la reconstrucción del modelo en tercera dimensión y su visualización, ofreciendo al usuario una interfaz con un diseño sencillo y fácil de manejar. En cuanto a la medición del modelo se ha desarrollado una herramienta, parecida a una regla, la cual permite al usuario medir la longitud en micras del modelo de manea manual.

Además de las características anteriormente mencionadas se ofrecen una serie de características extras entre las cuales se pueden mencionar:

- Abrir y guardar la información del modelo.
- Rotar y Escalar el modelo.
- Cambiar el color de todos los elementos de la escena
- Mostar y ocultar cualquier elemento de la escena.

Estos servicios no fueron específicamente mencionados por el usuario, pero su implementación era necesaria, ya que estas características hacen práctico y rápido el uso del sistema. En la **Figura 5.2** se muestra un diagrama de todas las características hasta ahora implementadas en Muni.

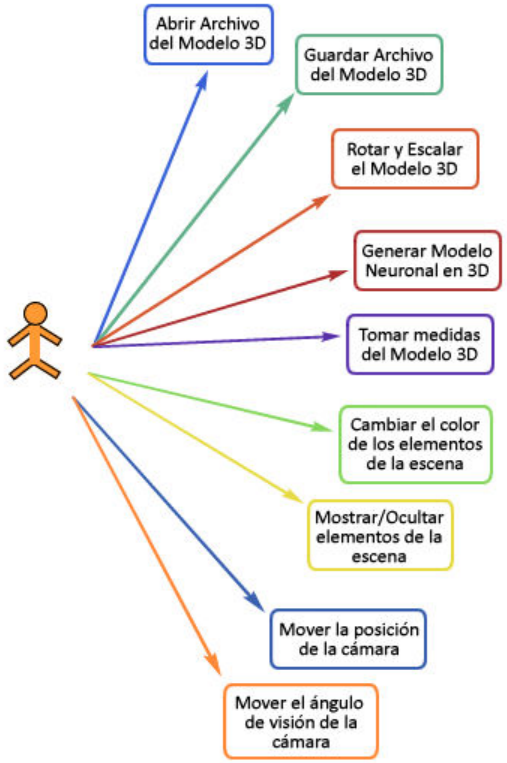

Figura 5.2 Diagrama de los servicios proporcionados por Muni

5.3 Descripción general de la Interfaz

A continuación se mostrarán algunas capturas de pantalla de las interfaces del programa que describen de manera general la funcionalidad de los objetos que las conforman.

En la **Figura 5.3** se ve puede apreciar una captura de pantalla de la ventana principal del programa, con la descripción de sus diferentes regiones. En esta ventana el usuario tiene la posibilidad de acceder a cualquier característica del sistema. Además en esta ventana se realiza la visualización de la reconstrucción, por lo cual en esta área se presentan los controles necesarios para manipular, medir o visualizar el modelo.

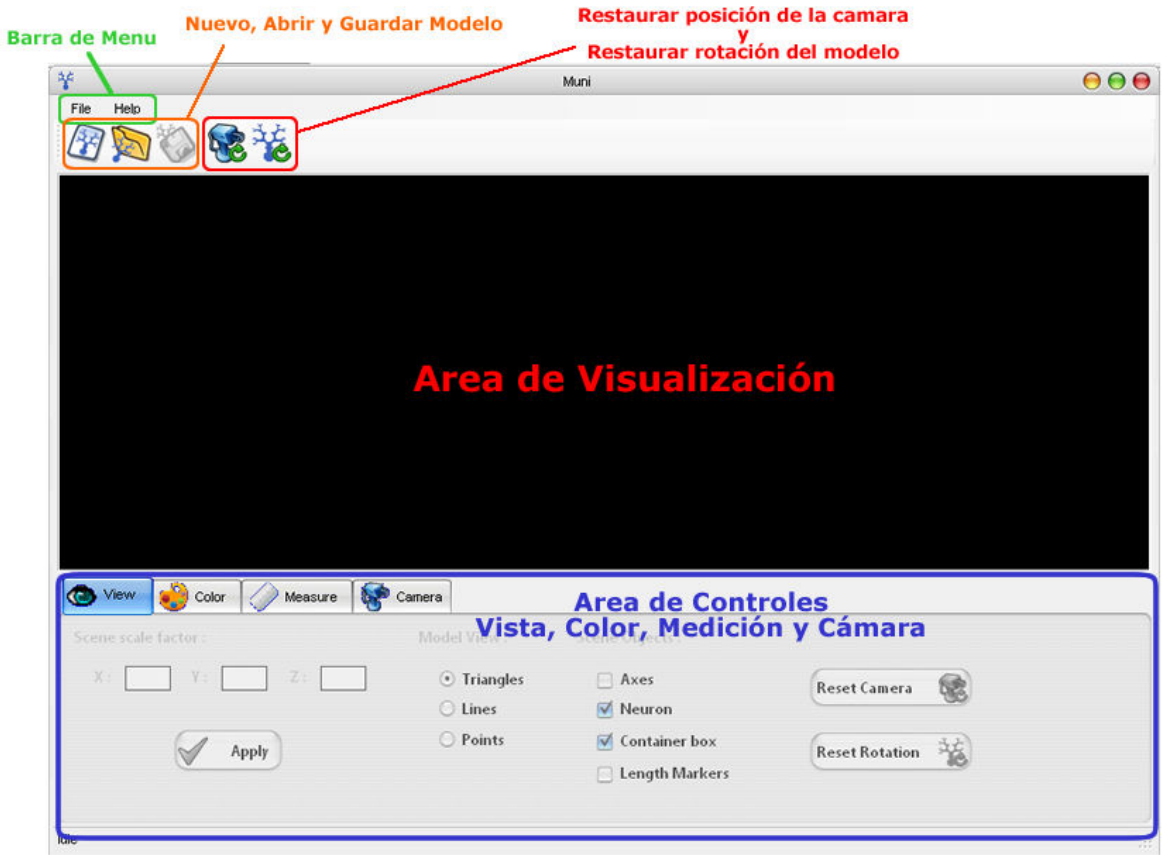

Figura 5.3 Captura de pantalla de la ventana principal del sistema junto la descripción de los componentes.

Se ha desarrollado una interfaz especial para tomar los parámetros necesarios para generar el modelo. El diseño de esta interfaz está dividida en pasos, impidiendo al usuario continuar con el proceso si es que ha cometido un error. Esto ayuda a evitar en lo posible que el usuario cometa errores a la hora de generar el modelo.

En las **Figura 5.4, Figura** 5.5 y **Figura** 5.6 se ven las capturas de todas las fases en las que se encuentra dividida la interfaz para la creación de un nuevo modelo. En la **Figura 5.4** se ve el primer paso del proceso que consiste en la selección de las imágenes, al usuario se le otorga la libertad de escoger los archivos por lista o por carpeta, dependiendo de la forma en la que se encuentren organizadas las fotografías.

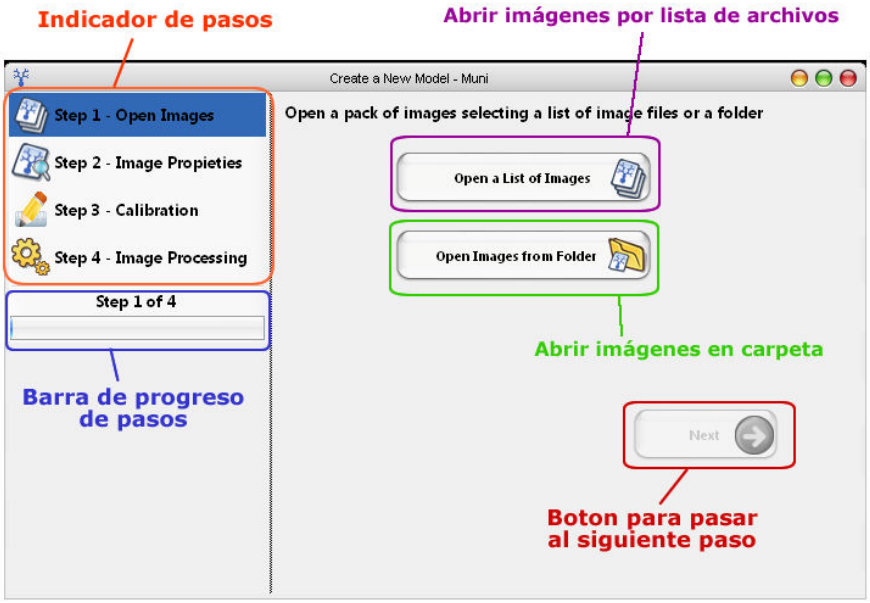

Figura 5.4 Captura de pantalla del primer paso, de la interfaz para la creación de un nuevo modelo.

La **Figura 5.5** muestra la segunda fase de las ventanas para la reconstrucción, esta interface cuenta con campos para introducir la magnitud el objetivo del microscopio utilizado y para indicar la resolución en micras del paquete de imágenes. El usuario puede indicar la resolución real del paquete de imágenes indicando la cantidad de pixeles que equivalen a tantas micras.

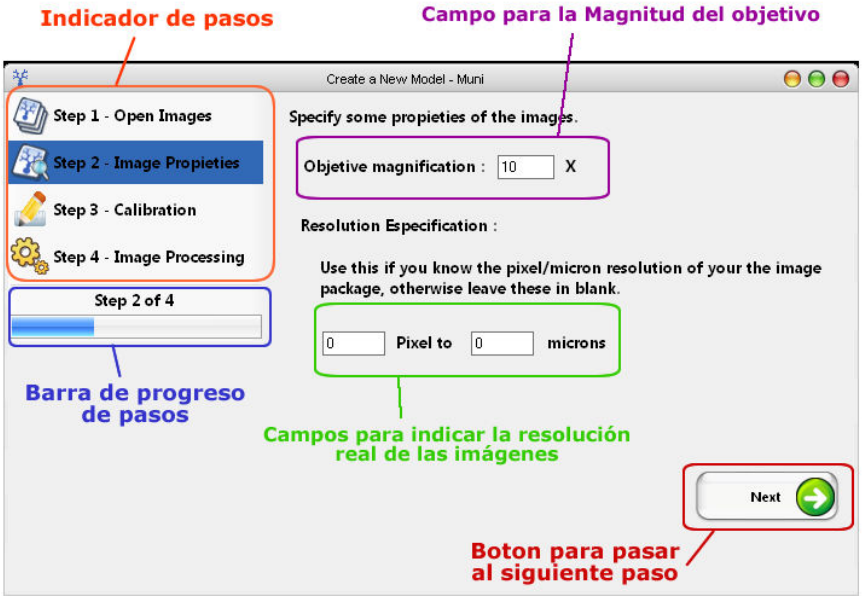

Figura 5.5 Captura de pantalla del segundo paso, de la interfaz para la creación de un nuevo modelo.

Durante la siguiente fase el usuario debe de seleccionar al menos tres regiones en foco y tres regiones con fondo; para calcular el valor de superficie. Además debe de indicar la posición de una dendrita en foco y estimar su diámetro en micras, para poder calcular la resolución real del paquete de imágenes, si es que no ha sido indicado en la fase anterior. Como se puede ver en la **Figura 5.6** esta interfaz permite al usuario visualizar y mover de posición cualquier imagen del paquete. Además para facilitar la selección de las

regiones, el usuario puede llegar a cualquier posición de la imagen dando un click sobre el thumbnail que se encuentra a la derecha del área de visualización.

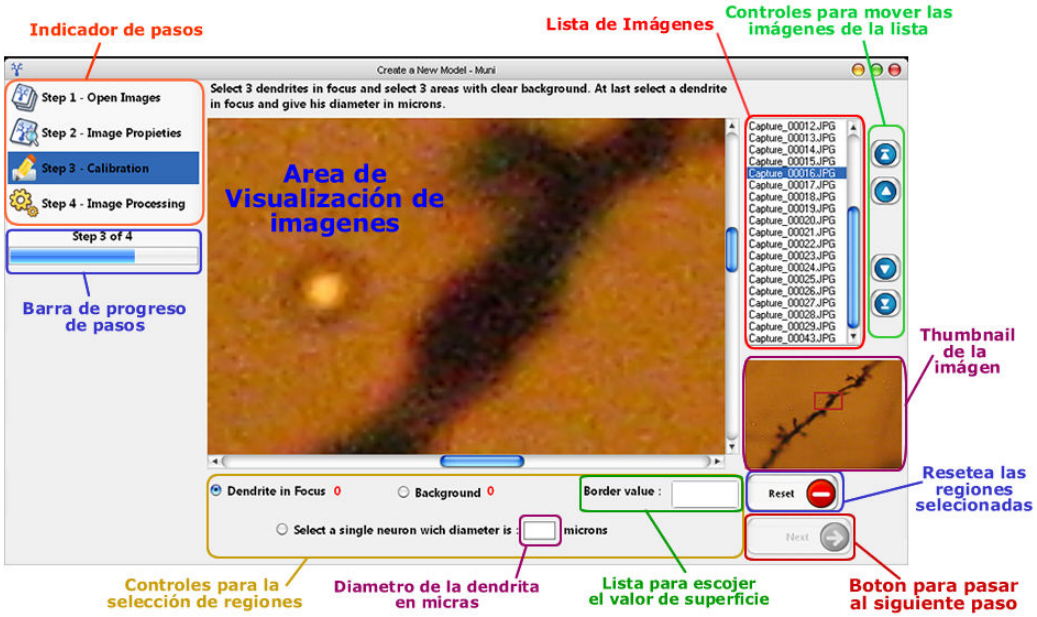

Figura 5.6 Captura de pantalla del tercer paso, de la interfaz para la creación de un nuevo modelo.

En la última fase de la reconstrucción aparecen varios campos de texto, para indicar las dimensiones para el marching cube, las cuales entre más pequeñas sean mayor resolución tendrá el modelo y por lo tanto será mucho más grande y pesado. También hay cuadros texto para ajustar los factores de escalamiento del modelo sobre los tres ejes de coordenadas. Además si las imágenes originales son pequeñas se da la opción de aplicar un Zoom bilineal del 300%, con el propósito de mejorar los resultados de la reconstrucción, lo cual se indica con un caja de selección en la parte superior de la ventana, (véase **Figura 5.7**).

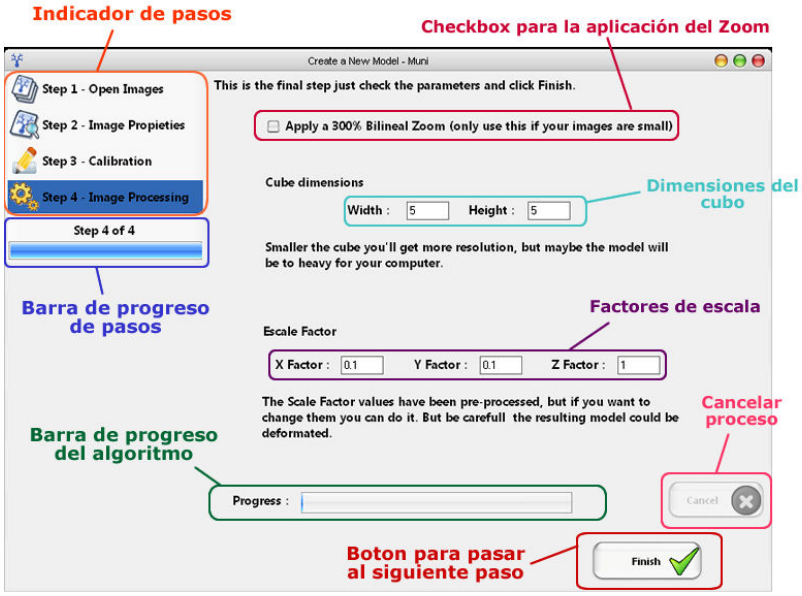

Figura 5.7 Captura de pantalla del cuarto paso, de la interfaz para la creación de un nuevo modelo.

5.4 Análisis de los Resultados

En esta sección se mencionarán las características de las imágenes y de los modelos tridimensionales generados usando la estrategia planteada en el Capitulo 4. Se especificarán las características más sobresalientes de los paquetes de imágenes como sus dimensiones, su resolución si se les aplico algún filtro o no y la cámara con la que se realizo la captura. Además se presentarán capturas de pantalla de los diferentes modelos neuronales generados, mencionando la cantidad de triángulos utilizados, sus factores de escala, sus dimensiones y el tiempo que tardo el software en generar la reconstrucción.

Se tomaran como ejemplo los resultados de cuatro reconstrucciones diferentes, que fueron generadas usando paquetes de imágenes de diferentes neuronas. En el primer y segundo caso tenemos una neurona y una dendrita cuyas imágenes fueron tomadas usando la cámara JVC TK-C1380. Mientras que en el tercer y cuarto caso tenemos igualmente una neurona y una dendrita fotografiadas usando una cámara Canon A470. Para identificar a cada modelo se numerará de manera sucesiva a cada uno de ellos.

Debido a que los tiempos de procesamiento de las imágenes pueden variar de una maquina a otra, dependiendo de sus características, es necesario mencionar que todos los resultados aquí presentados fueron realizados en una laptop Pentium (R) Dual Core a 2.0 GHz, con 1.99 Gbytes de memoria RAM sin tarjeta aceleradora de gráficos.

5.4.1 Modelo #1 – Neurona completa

Este modelo se generó usando un paquete de imágenes de una neurona completa. En la **Figura 5.8** vemos una serie representativa del paquete de imágenes. En la **Tabla 5.1** se muestran las propiedades del paquete de imágenes mientras que en la **Tabla 5.2** se muestran las propiedades del modelo generado. En la **Figura 5.9** Captura de pantalla del modelo #1 vemos el modelo tridimensional #1 generado a partir del paquete de imágenes #1.

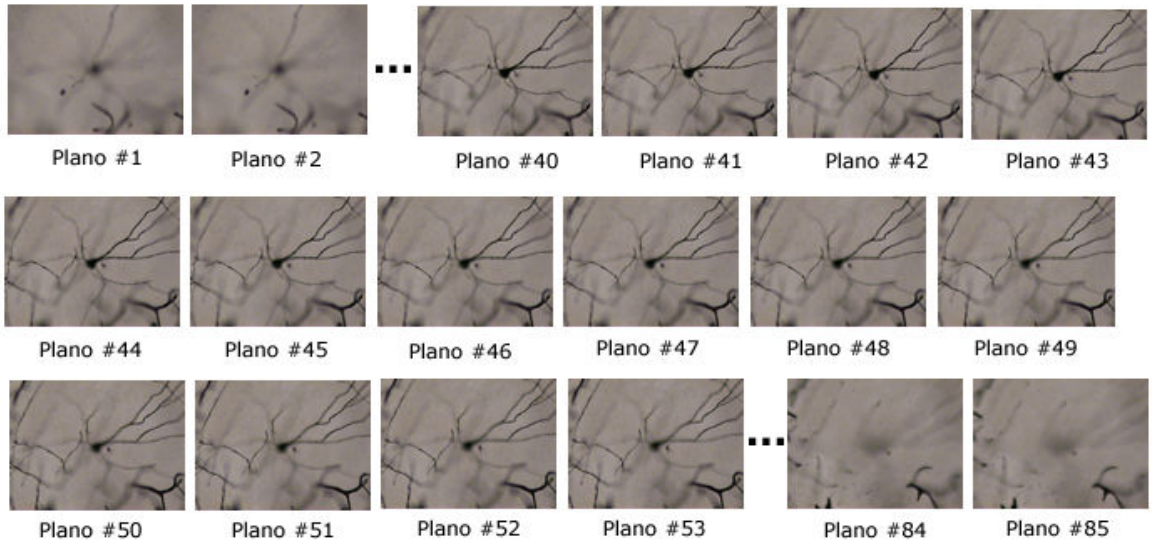

Figura 5.8 Una serie representativa del paquete de imágenes #1

Cámara	JVC TK-C1380
Objetivo	40x
Zoom Bilineal	Si
Conversión a Escala de Grises	Si
Dimensiones Originales	764 x 574 pixeles
Dimensiones después del Zoom	2292 x 1722 pixeles
Número de planos	85 planos
Tamaño de cada archivo	1.25 Mb
Tamaño después del Zoom	1.75 Mb

Tabla 5.1 Propiedades del paquete de imágenes #1

Cantidad de triángulos	349974
Dimensiones del cubo	5 x 5
Valor de Superficie	65
Factores de Escala	0.1, 0.1, 0.4
Dimensiones del modelo	2292, 1722, 82
Tiempo de procesamiento (aplicando Zoom)	138 segundos
Tiempo de procesamiento (sin aplicar Zoom)	51 segundos

Tabla 5.2 Propiedades del modelo tridimensional #1

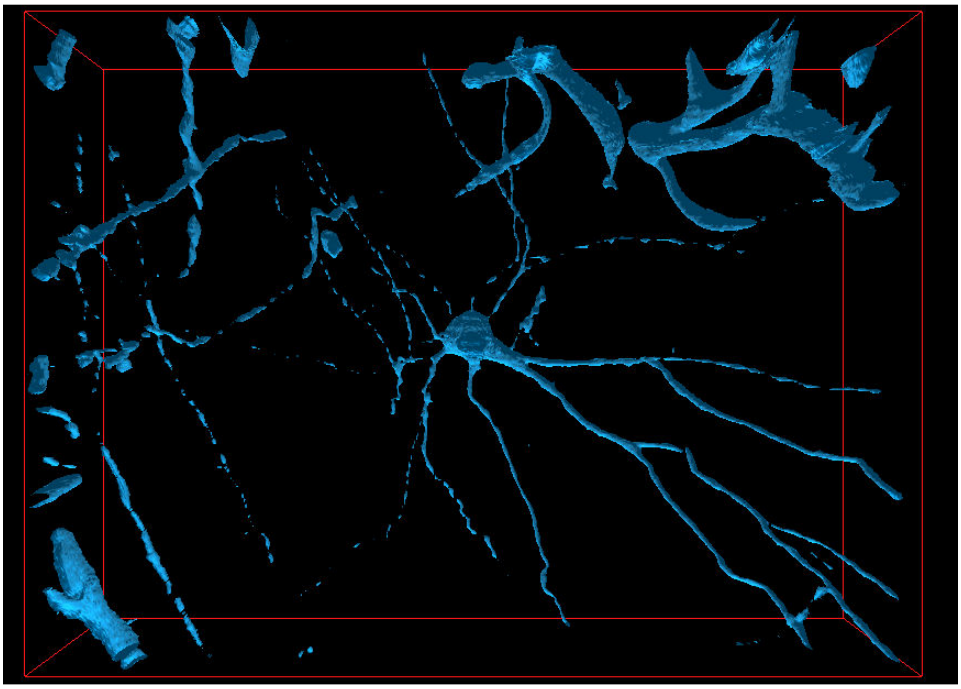

Figura 5.9 Captura de pantalla del modelo #1

5.4.2 Modelo #2 – Sección de dendrita con espinas

Este modelo se generó usando un paquete de imágenes de sección de una dendrita. En la **Figura 5.10** vemos una serie representativa del paquete de imágenes. En la **Tabla 5.3** se muestran las propiedades del paquete de imágenes mientras que en la **Tabla 5.4** se muestran las propiedades del modelo generado. En la **Figura 5.11** vemos el modelo tridimensional #2.

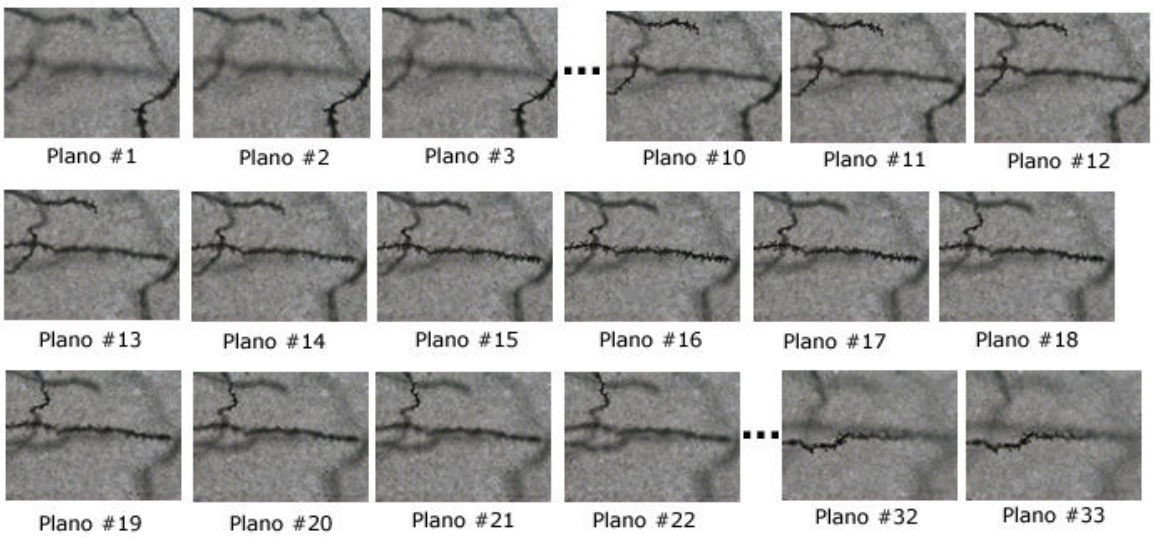

Figura 5.10 Serie representativa del paquete de imágenes #2

Cámara	JVC TK-C1380
Objetivo	100x
Zoom Bilineal	Si
Conversión a Escala de Grises	Si
Dimensiones Originales	764 x 574 pixeles
Dimensiones después del Zoom	2292 x 1722 pixeles
Número de planos	33 planos
Tamaño de cada archivo	1.25 Mb
Tamaño después del Zoom	2.35 Mb

Tabla 5.3 Propiedades del paquete de imágenes número #2

Cantidad de triángulos	232607
Dimensiones del cubo	5 x 5
Valor de Superficie	64
Factores de Escala	0.1, 0.1, 0.4
Dimensiones del modelo	2292, 1722, 32
Tiempo de procesamiento (aplicando Zoom)	120 segundos
Tiempo de procesamiento (sin aplicar Zoom)	21 segundos

Tabla 5.4 Propiedades del modelo tridimensional #2

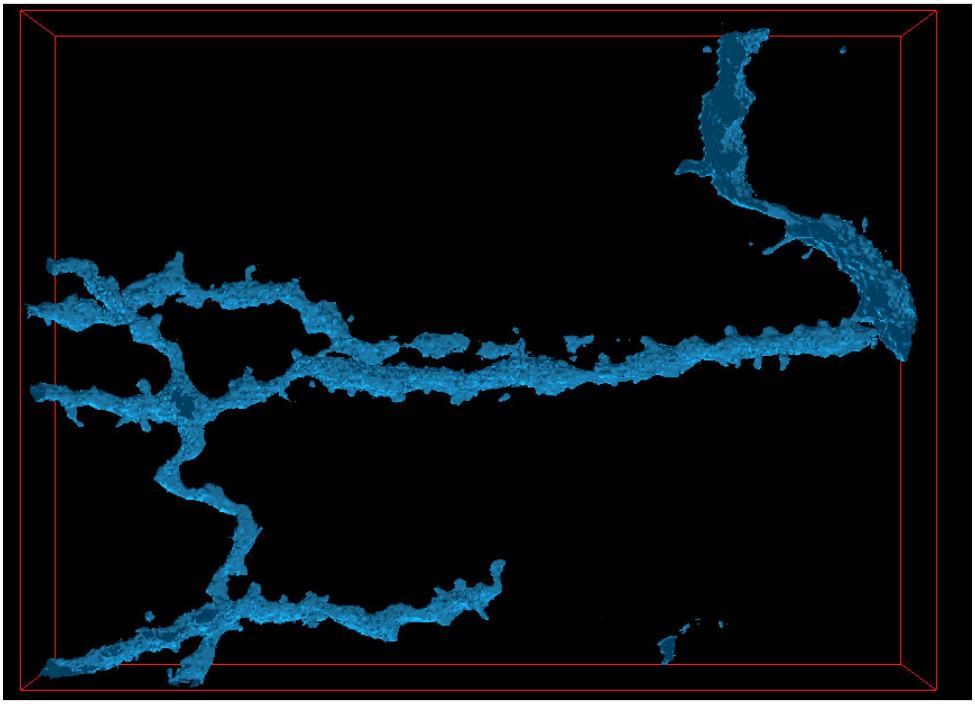

Figura 5.11 Figura que muestra el modelo tridimensional #2 generado a partir del paquete de imágenes #2. En el modelo se pueden apreciar algunas espinas dendríticas.

5.4.3 Modelo #3 – Neurona completa

Se generó usando un paquete de imágenes de una neurona completa. En la **Figura 5.12** vemos una serie representativa del paquete de imágenes. En la **Tabla 5.5** se muestran las propiedades del paquete de imágenes mientras que en la **Tabla 5.6** se muestran las propiedades del modelo generado. En la **Figura 5.13** vemos el modelo tridimensional #3.

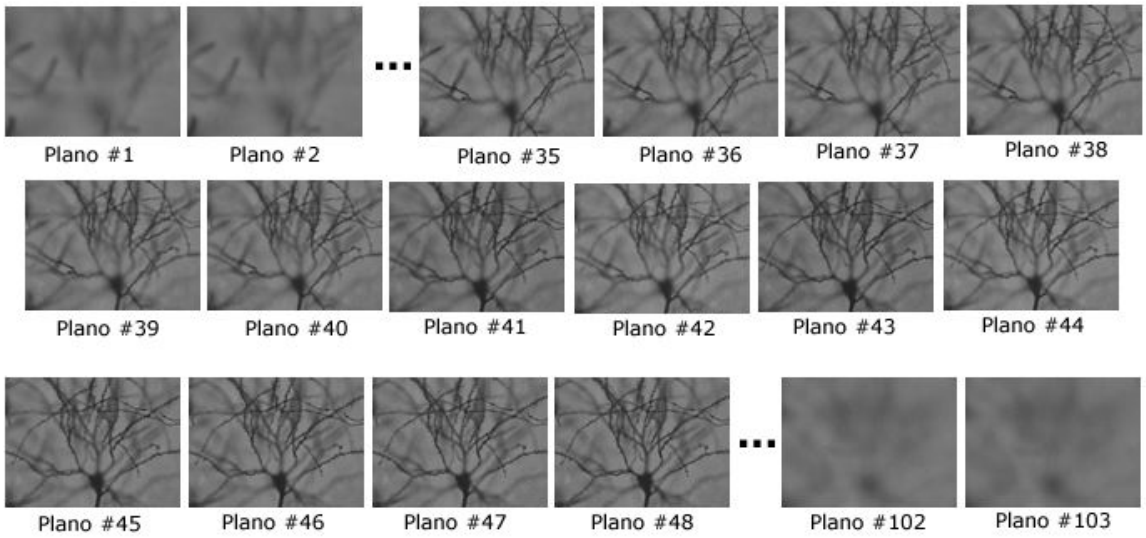

Figura 5.12 Figura con una serie representativa del paquete de imágenes #3.

Cámara	Canon a470
Objetivo	40x
Zoom Bilineal	No
Conversión a Escala de Grises	Si
Dimensiones Originales	3264 x 2448 pixeles
Número de planos	103 planos
Tamaño de cada archivo	3.00 Mb

Tabla 5.5 Propiedades del paquete de imágenes #3

Cantidad de triángulos	2374895
Dimensiones del cubo	5 x 5
Valor de Superficie	55
Factores de Escala	0.1, 0.1, 0.4
Dimensiones del modelo	3264, 2448, 103
Tiempo de procesamiento (sin aplicar Zoom)	198 segundos

Tabla 5.6 Propiedades del modelo tridimensional #3

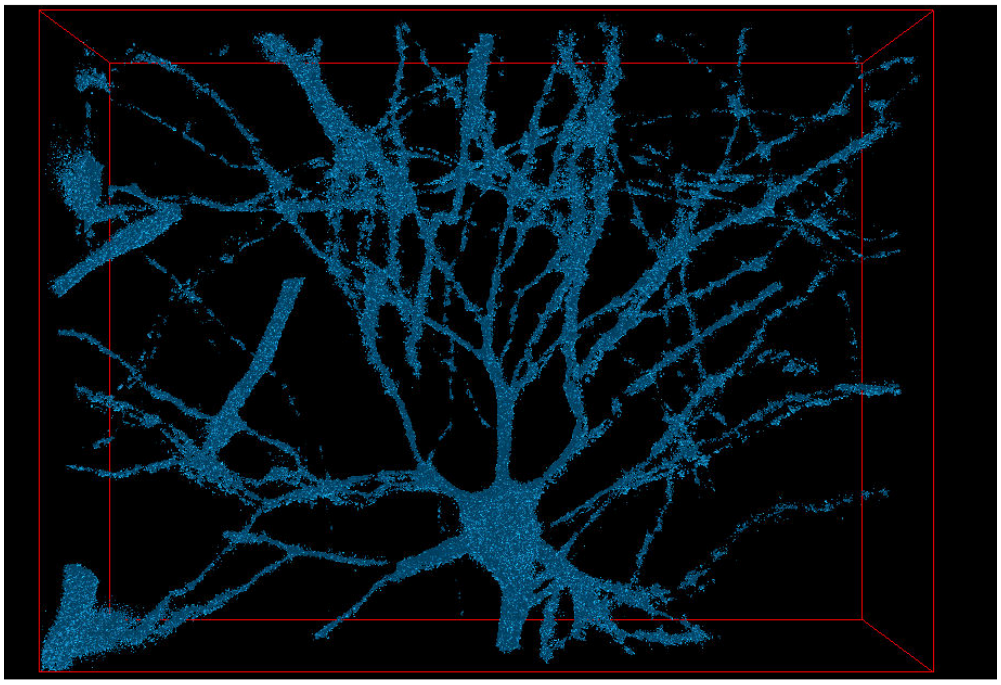

Figura 5.13 Figura que muestra el modelo tridimensional #3.

5.4.4 Modelo #4 – Sección de dendrita con espinas

Este se generó usando un paquete de imágenes de una dendrita. En la **Figura 5.14** vemos una serie representativa del paquete de imágenes. En la **Tabla 5.7** se muestran las propiedades del paquete de imágenes mientras que en la **Tabla 5.8** se muestran las propiedades del modelo generado. En la **Figura 5.15** vemos el modelo tridimensional #4.

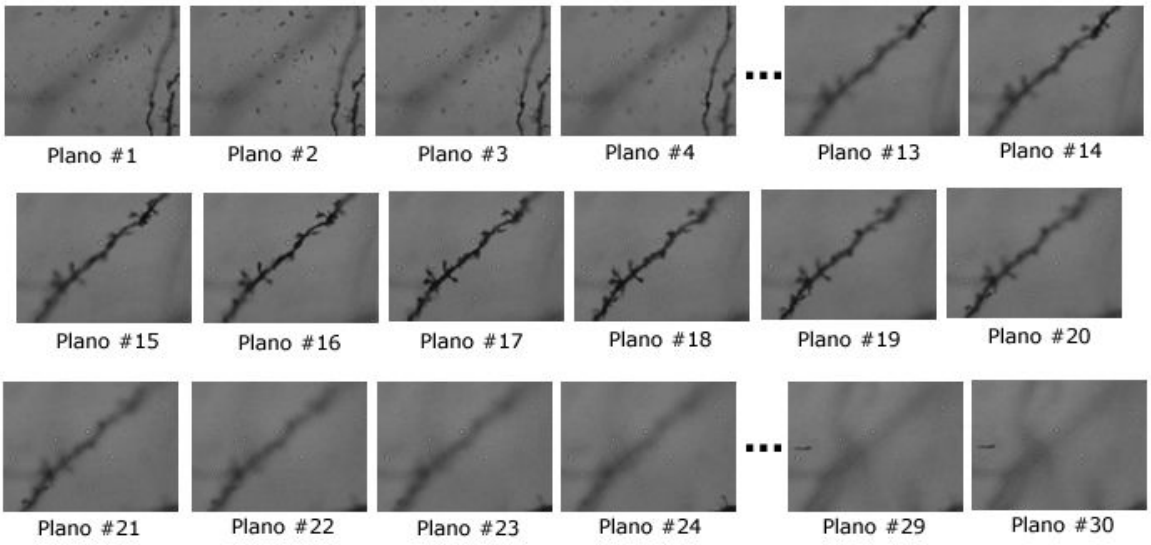

Figura 5.14 Diagrama con una serie representativa de fotografías del paquete de imágenes #4.

Cámara	Canon a470
Objetivo	100x
Zoom Bilineal	No
Conversión a Escala de Grises	Si
Dimensiones Originales	3264 x 2448 pixeles
Número de planos	103 planos
Tamaño de cada archivo	3.00 Mb

Tabla 5.7 Propiedades del paquete de imágenes #4

Cantidad de triángulos	171714
Dimensiones del cubo	5 x 5
Valor de Superficie	45
Factores de Escala	0.1, 0.1, 0.4
Dimensiones del modelo	3264,2448,30
Tiempo de procesamiento (sin aplicar Zoom)	42 segundos

Tabla 5.8 Propiedades del modelo tridimensional #4

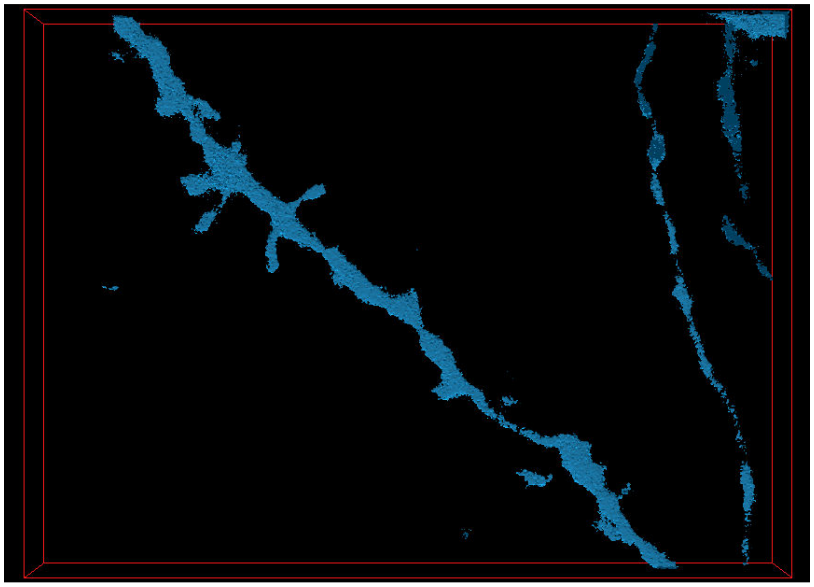

Figura 5.15 Captura de pantalla del modelo tridimensional #4, en el cual se pueden apreciar algunas espinas dendríticas.

5.5 Medición del modelo

El sistema de medición desarrollado es capaz de medir la distancia en micras de un punto a otro del modelo, usando como marcadores dos esferas que se pueden mover a través de la escena. Esta medición se lleva a cabo usando la resolución real del paquete de imágenes, la cual ya ha sido previamente calculada.

Es entonces que usando la relación entre la cantidad de pixeles que representan cierta cantidad de micras es posible saber la longitud que existe entre dos dentro de la escena, y para esto se utilizan los dos marcadores esféricos que indican el inicio y el final de la medición.

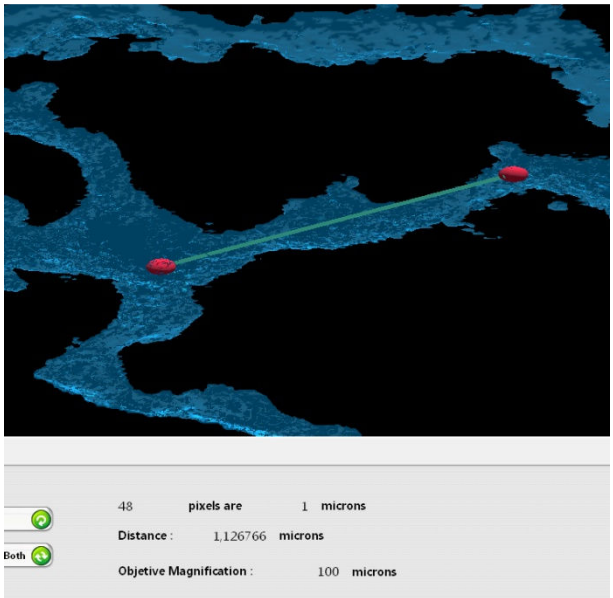

Figura 5.16 Figura que muestra la herramienta de medición, la cual calcula la distancia entre dos puntos del modelo en micras.

En la **Figura 5.16** se pueden apreciar los marcadores esféricos que sirven como indicadores de la medición, en la parte baja de la ventana en la pestaña de Medición, en el Área de Controles se puede ver que cada vez que las esferas se mueven, se calcula la longitud entre ellas, dando el resultado siempre en micras. Para el caso que se muestra en la **Figura 5.16** la distancia que existente entre las dos esferas es de 1.33 micras.

A parte de indicar al usuario la distancia en micras también se indica la relación pixel/micra y la magnitud del objetivo utilizado para ese modelo en particular.

5.6 Requisitos del sistema

El sistema es capaz de crear un modelo tridimensional en cualquier tipo que cuente con un sistema operativo Windows XP o superior y que posea como mínimo un 1 Gbyte de memoria RAM. Por otra parte para que la visualización se de de manera fluida es necesario que la computadora cuente con un tarjeta aceleradora de video compatible con OpenGL. Es por eso que se recomienda ejecutar Muni en sistema con las siguientes características.

- **Sistema Operativo**: Windows XP o superior.
- **Procesador:** Procesador Pentium 4 2.5 GHz, equivalente o superior.
- **Memoria:** 1 Gbyte de memoria RAM
- **Video:** Tarjeta aceleradora compatible con OpenGL con 64 Mb de memoria o más.

Los requisitos mínimos mencionados anteriormente permitirán que el sistema procese los datos de manera adecuada y la visualización del modelo se realice de manera fluida.

CONCLUSIONES

Detrás del software desarrollado existe un proceso de investigación que ha explorado diversas técnicas para generar una reconstrucción tridimensional a partir de información médica o biológica. De todas las técnicas exploradas se ha llegado a tomar la que mejor se ajusta al problema en particular y la que mejores resultados arrojaba; la que finalmente se usó fue el Algoritmo de los Marching Cubes. Este algoritmo se implemento y adapto para que pudiera funcionar con los tipos de datos generados por los microscopios del Laboratorio de Neuropsiquiatría, pero bien podría funcionar con imágenes capturadas usando cualquier tipo de microscopio de luz simple.

A partir de esta investigación finalmente se desarrolló un sistema que cumple satisfactoriamente con los objetivos inicialmente propuestos. El sistema de software implementado es sencillo y fácil de usar permitiendo al usuario generar la reconstrucción en tercera de dimensión de una neurona, ver el modelo en cualquier ángulo y también le permite tomar mediciones a partir de este.

Además de las características planteadas inicialmente por el usuario se incluyen características extras como cambiar los colores de cualquier elemento de la escena, cambiar los factores de escala del modelo sobre cualquier eje y también permite ocultar o mostrar cualquier elemento de la escena, características que facilitan la visualización y la medición del modelo.

Existen muchas posibles mejoras que se pueden alcanzar y que permitirán al Laboratorio de Neuropsiquiatría agilizar aún más su trabajo de investigación, las cuales se mencionarán en orden de importancia:

- Afinar el método para obtener la resolución real del paquete de imágenes.
- Implementación de una herramienta manual que permita construir un modelo discreto de la neurona, para poder medir la longitud de sus dendritas.
- Herramienta que mida de manera automática las longitudes de las estructuras dendríticas hasta en un tercer nivel.
- Llevar un control sobre los modelos generados, para que el mismo sistema lleve la estadística de la investigación.
- Incluir en el mismo sistema de software un modulo de control para la captura automática de las fotografías.
- Exportar el modelo generado en formatos de tercera dimensión comerciales, como 3D Studio Max (*.3ds)

Este software fue desarrollado pensado en su continuo mejoramiento y mantenimiento; es por eso que tanto el software como su respectivo código fuente estarán disponibles de manera gratuita para todo el público, que así lo desee, en el sitio de Google Code. `http://code.google.com/p/muni-neuron/`.

El programa desarrollado se liberará bajo la licencia GNU v3.0 fomentando a otras personas a unirse al proyecto generando una mejor herramienta libre y gratuita que sirva a todos los laboratorios de neurofisiología a lo largo del mundo.

BIBLIOGRAFIA

Qiang Wu, Fatima Merchant, Kenneth Castleman (2008). “Microscope Image Processing”, Academic Press, Cap 2, pags. 11-12.

Chris Guy, Dominic Ffytche, (2005-06), “Introduction to the Principles of Medical Imaging”, Imperial College Press, Introduction, pags. xl-xli.

Gordon S. Kino, Timothy R. Corle, (1996), “Confocal Scanning Optical Microscopy and Related Imaging Systems”, Academic Press, Cap 1, pags 31-33.

Robert A. Wilson, Frank C. Keil, (1999), “The MIT Encyclopedia of the Cognitive Sciences”, The MIT Press, Introduction, pags liii, liv.

Douglas B. Murphy, (2001), “Fundamentals of Light Microscopy and Electronic Imaging”, Cap 14, pags 359-361.

Mark Mumenthaler, Heinrich Mattle, Ethan Taub, (2006), “Fundamantal of Neurology”, Thieme, Cap 1, pags 1-2.

Stramotios V. Kartalopoulos, (1996), “Understanding Neural Networks and Fussy Logic”, IEEE Press, Cap 1, pags 10-14.

William E. Lorensen, Harvey E. Cline, “Marching Cubes a High Resolution 3D Surface Construction Algorithm”, SIGGRAPH 87, Computer Graphics Volume 21, Number 4.

OpenGL Architecture Review Board, “OpenGL Reference Manual”, Addison-Wesley, Caps 3, 4.

Richard S. Wright, Benjamin Lipchak, Nicholas Haemel, (2007), “OpenGl Super Bible”, Addison-Wesley, Cap 11.

Daniel Sánchez, Crespo Dalmau, (2003), “Core Techniques and Algorithms in Game Programing”, New Riders Publishing, Cap 16, Pags 486-490

A P E N D I C E S

APENDICE A: FUNCIONES DE OPENGL

En este anexo se describirán las funciones de la API de OpenGL mencionadas en este trabajo, en caso de que el lector no esté relacionado con la notación o los parámetros de estas.

Se mencionarán las funciones que han sido presentadas en este trabajo por orden alfabético, donde se incluirá una breve reseña de su funcionamiento y la descripción de sus parámetros y su retorno; en caso de que lo tuvieran.

glDrawArrays

Descripción:

glDrawArrays.- Renderiza todas las primitivas en el arreglo de datos o de vértices.

Encabezado:

```
void glDrawArrays( GLenum mode,
                   GLint first,
                   GLsizei count )
```

Parámetros:

mode	Especifica qué tipo de primitivas se van a renderizar. Las constantes simbolicas **GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_TRIANGLES, GL_QUAD_STRIP. GL_QUADS** y **GL_POLYGON** son aceptádas.
first	Especifica el índice de inició de los arreglos habilitados.
count	Especifica el número de índices a ser renderizados.

glEnableClientState, glDisableClientState

Descripción:

glEnableClientState, glDisableClientState.- Activa las capacidades del lado del cliente, las cuales por default se encuentran desactivadas.

Encabezado:

```
void glEnableClientState( GLenum cap )
void glDisableClientState( GLenum cap )
```

Parámetros:

cap	Especifica la capacidad que se desea activar, puede asumir los siguientes valores: **GL_COLOR_ARRAY, GL_EDGE_FLAG_ARRAY, GL_INDEX_ARRAY, GL_NORMAL_ARRAY, GL_TEXTURE_COORD_ARRAY** y

GL_VERTEX_ARRAY.

En este caso para *cap* solo se utilizó **GL_VERTEX_ARRAY** y **GL_NORMAL_ARRAY**.

GL_VERTEX_ARRAY	Si es activado, permite el uso de los arreglos de vértices en modo de escritura, cada vez que las funciones **glDrawArrays** o **glDrawElements** se ha llamada durante tiempo de renderizado.
GL_NORMAL_ARRAY	Si es activado, permite el uso de los arreglos de normales en modo de escritura, cada vez que las funciones **glDrawArrays** o **glDrawElements** se ha llamada durante tiempo de renderizado.

glGetIntegerv (glGet)

Descripción:

glGetIntegerv.- Regresa el valor o valores de parámetro seleccionado.

Encabezado:

```
void glGetIntegerv(     GLenum      pname,
                        GLint *     params);
```

Parámetros:

pname	Especifica el parámetro a ser retornado. Usando constantes simbólicas, la constante simbólica GL_MODELVIEW_MATRIX es aceptada.
params	Especifica la posición sobre el eje *Y* del ojo de la cámara.

glInitNames

Descripción:

glInitNames.- Inicializa el stack de nombres

Encabezado:

```
void glInitNames( void )
```

gluLookAt

Descripción:

gluLookAt.- Define una transformación de visión.

Encabezado:

```
void gluLookAt( GLdouble eyeX,
                GLdouble eyeY,
                GLdouble eyeZ,
                GLdouble centerX,
                GLdouble centerY,
                GLdouble centerZ,
                GLdouble upX,
```

```
                GLdouble upY,
                GLdouble upZ )
```

Parámetros:

eyeX	Especifica la posición sobre el eje *X* del ojo de la cámara.
eyeY	Especifica la posición sobre el eje *Y* del ojo de la cámara.
eyeZ	Especifica la posición sobre el eje *Z* del ojo de la cámara.
centerX	Especifica la posición del punto de referencia, sobre el eje *X*.
centerY	Especifica la posición del punto de referencia, sobre el eje *Y*.
centerZ	Especifica la posición del punto de referencia, sobre el eje *Z*.
upX	Especifica la dirección del vector de orientación, sobre el eje *X*.
upY	Especifica la dirección del vector de orientación, sobre el eje *Y*.
upZ	Especifica la dirección del vector de orientación, sobre el eje *Z*.

glNormalPointer

Descripción:

glNormalPointer.- Define un arreglo para un conjunto de normales de datos

Encabezado:

```
void glNormalPointer(    GLenum type,
                           GLsizei       stride,
                           const GLvoid *pointer )
```

Parámetros:

type	Especifica el tipo de dato de cada coordenada en el arreglo. Las constantes simbólicas **GL_BYTE, GL_SHORT, GL_INT, GL_FLOAT** y **GL_DOUBLE** son aceptadas. El valor inicial es **GL_FLOAT.**
stride	Especifica el margen en bytes entre normales consecutivas dentro del arreglo. Si el margen es 0, entonces se entiende que la información de los vértices está almacenada consecutivamente. El valor inicial es 0.
pointer	Especifica el apuntador a la primera coordenada de la primera normal del arreglo.

glRenderMode

Descripción:

glRenderMode.- Determina el método de rasterización de la escena

Encabezado:

```
GLint     glRenderMode( GLenum mode )
```

Parámetros:

mode	Especifica qué tipo de rasterización se va a usar. Tres valores son aceptados; **GL_RENDER, GL_SELECT, GL_FEEDBACK.** El valor inicial es

GL_RENDER.

Retorno:

El valor de retorno de **glRenderMode** está determinado por el modo de renderizado al tiempo de **glRenderMode** es llamado, en vez del parámetro *mode*. Los valores de retorno para estos tres modos de renderizado son como sigue:

GL_RENDER	0
GL_SELECT	El número de clicks dados son transferidos al Buffer de Selección.
GL_FEEDBACK	El número de valores (no de vértices) son transferidos al Buffer de Feedback.

glVertexPointer

Descripción:

glVertexPointer.- Define un arreglo para un conjunto de vértices de datos

Encabezado:

```
void glVertexPointer(    GLint size,
                         GLenum type,
                         GLsizei stride,
                         const GLvoid *pointer )
```

Parámetros:

size	Especifica el número de coordenadas por vértice; debe de ser 2, 3 o 4. El valor inicial es 4.
type	Especifica el tipo de dato de cada coordenada in el arreglo. Constantes simbólicas **GL_SHORT, GL_INT, GL_FLOAT, y GL_DOUBLE** son aceptadas. El valor inicial es **GL_FLOAT**.
stride	Especifica el margen en bytes entre vértices dentro del arreglo. Si el margen es 0, entonces se entiende que la información de los vértices está almacenada consecutivamente. El valor inicial es 0.
pointer	Especifica el apuntador a la primera coordenada del primer vértice dentro del arreglo

APENDICE B: ARREGLOS DE ORILLAS Y TRIANGULOS

Declaración y contenido de la tabla de orillas, como cada elemento del arreglo tiene una codificación que ocupa 12 bits, para reducir la notación de cada uno de los valores, es necesario utilizar una notación en hexadecimal.

```
private int[] tabla_orillas = new int[256]
 {
   0x0  , 0x109, 0x203, 0x30a, 0x406, 0x50f, 0x605, 0x70c,
   0x80c, 0x905, 0xa0f, 0xb06, 0xc0a, 0xd03, 0xe09, 0xf00,
   0x190, 0x99 , 0x393, 0x29a, 0x596, 0x49f, 0x795, 0x69c,
   0x99c, 0x895, 0xb9f, 0xa96, 0xd9a, 0xc93, 0xf99, 0xe90,
   0x230, 0x339, 0x33 , 0x13a, 0x636, 0x73f, 0x435, 0x53c,
   0xa3c, 0xb35, 0x83f, 0x936, 0xe3a, 0xf33, 0xc39, 0xd30,
   0x3a0, 0x2a9, 0x1a3, 0xaa , 0x7a6, 0x6af, 0x5a5, 0x4ac,
   0xbac, 0xaa5, 0x9af, 0x8a6, 0xfaa, 0xea3, 0xda9, 0xca0,
   0x460, 0x569, 0x663, 0x76a, 0x66 , 0x16f, 0x265, 0x36c,
   0xc6c, 0xd65, 0xe6f, 0xf66, 0x86a, 0x963, 0xa69, 0xb60,
   0x5f0, 0x4f9, 0x7f3, 0x6fa, 0x1f6, 0xff , 0x3f5, 0x2fc,
   0xdfc, 0xcf5, 0xfff, 0xef6, 0x9fa, 0x8f3, 0xbf9, 0xaf0,
   0x650, 0x759, 0x453, 0x55a, 0x256, 0x35f, 0x55 , 0x15c,
   0xe5c, 0xf55, 0xc5f, 0xd56, 0xa5a, 0xb53, 0x859, 0x950,
   0x7c0, 0x6c9, 0x5c3, 0x4ca, 0x3c6, 0x2cf, 0x1c5, 0xcc ,
   0xfcc, 0xec5, 0xdcf, 0xcc6, 0xbca, 0xac3, 0x9c9, 0x8c0,
   0x8c0, 0x9c9, 0xac3, 0xbca, 0xcc6, 0xdcf, 0xec5, 0xfcc,
   0xcc , 0x1c5, 0x2cf, 0x3c6, 0x4ca, 0x5c3, 0x6c9, 0x7c0,
   0x950, 0x859, 0xb53, 0xa5a, 0xd56, 0xc5f, 0xf55, 0xe5c,
   0x15c, 0x55 , 0x35f, 0x256, 0x55a, 0x453, 0x759, 0x650,
   0xaf0, 0xbf9, 0x8f3, 0x9fa, 0xef6, 0xfff, 0xcf5, 0xdfc,
   0x2fc, 0x3f5, 0xff , 0x1f6, 0x6fa, 0x7f3, 0x4f9, 0x5f0,
   0xb60, 0xa69, 0x963, 0x86a, 0xf66, 0xe6f, 0xd65, 0xc6c,
   0x36c, 0x265, 0x16f, 0x66 , 0x76a, 0x663, 0x569, 0x460,
   0xca0, 0xda9, 0xea3, 0xfaa, 0x8a6, 0x9af, 0xaa5, 0xbac,
   0x4ac, 0x5a5, 0x6af, 0x7a6, 0xaa , 0x1a3, 0x2a9, 0x3a0,
   0xd30, 0xc39, 0xf33, 0xe3a, 0x936, 0x83f, 0xb35, 0xa3c,
   0x53c, 0x435, 0x73f, 0x636, 0x13a, 0x33 , 0x339, 0x230,
   0xe90, 0xf99, 0xc93, 0xd9a, 0xa96, 0xb9f, 0x895, 0x99c,
   0x69c, 0x795, 0x49f, 0x596, 0x29a, 0x393, 0x99 , 0x190,
   0xf00, 0xe09, 0xd03, 0xc0a, 0xb06, 0xa0f, 0x905, 0x80c,
   0x70c, 0x605, 0x50f, 0x406, 0x30a, 0x203, 0x109, 0x0
};
```

Declaración y contenido del arreglo de triángulos. Como se puede apreciar aquí se encuentran codificadas las 256 combinaciones posibles, dependiendo de la configuración de los vértices dentro y fuera de la superficie.

Cada faceta definida en la tabla tiene una orientación en sentido de las manecillas del reloj (CW), definido desde el punto de vista de los vértices en colisión.

```
private int[,] tabla_triangulos = new int[256, 16]
{
        {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {0, 1, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {1, 8, 3 , 9, 8, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {0, 8, 3, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {9, 2, 10, 0, 2, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {2, 8, 3, 2, 10, 8, 10, 9, 8, -1, -1, -1, -1, -1, -1, -1},
        {3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {0, 11, 2, 8, 11, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {1, 9, 0, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
```

```
{1, 11, 2, 1, 9, 11, 9, 8, 11, -1, -1, -1, -1, -1, -1, -1},
{3, 10, 1, 11, 10, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 10, 1, 0, 8, 10, 8, 11, 10, -1, -1, -1, -1, -1, -1, -1},
{3, 9, 0, 3, 11, 9, 11, 10, 9, -1, -1, -1, -1, -1, -1, -1},
{9, 8, 10, 10, 8, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{4, 3, 0, 7, 3, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 1, 9, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{4, 1, 9, 4, 7, 1, 7, 3, 1, -1, -1, -1, -1, -1, -1, -1},
{1, 2, 10, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{3, 4, 7, 3, 0, 4, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1},
{9, 2, 10, 9, 0, 2, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1},
{2, 10, 9, 2, 9, 7, 2, 7, 3, 7, 9, 4, -1, -1, -1, -1},
{8, 4, 7, 3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{11, 4, 7, 11, 2, 4, 2, 0, 4, -1, -1, -1, -1, -1, -1, -1},
{9, 0, 1, 8, 4, 7, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1},
{4, 7, 11, 9, 4, 11, 9, 11, 2, 9, 2, 1, -1, -1, -1, -1},
{3, 10, 1, 3, 11, 10, 7, 8, 4, -1, -1, -1, -1, -1, -1, -1},
{1, 11, 10, 1, 4, 11, 1, 0, 4, 7, 11, 4, -1, -1, -1, -1},
{4, 7, 8, 9, 0, 11, 9, 11, 10, 11, 0, 3, -1, -1, -1, -1},
{4, 7, 11, 4, 11, 9, 9, 11, 10, -1, -1, -1, -1, -1, -1, -1},
{9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{9, 5, 4, 0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 5, 4, 1, 5, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{8, 5, 4, 8, 3, 5, 3, 1, 5, -1, -1, -1, -1, -1, -1, -1},
{1, 2, 10, 9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{3, 0, 8, 1, 2, 10, 4, 9, 5, -1, -1, -1, -1, -1, -1, -1},
{5, 2, 10, 5, 4, 2, 4, 0, 2, -1, -1, -1, -1, -1, -1, -1},
{2, 10, 5, 3, 2, 5, 3, 5, 4, 3, 4, 8, -1, -1, -1, -1},
{9, 5, 4, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 11, 2, 0, 8, 11, 4, 9, 5, -1, -1, -1, -1, -1, -1, -1},
{0, 5, 4, 0, 1, 5, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1},
{2, 1, 5, 2, 5, 8, 2, 8, 11, 4, 8, 5, -1, -1, -1, -1},
{10, 3, 11, 10, 1, 3, 9, 5, 4, -1, -1, -1, -1, -1, -1, -1},
{4, 9, 5, 0, 8, 1, 8, 10, 1, 8, 11, 10, -1, -1, -1, -1},
{5, 4, 0, 5, 0, 11, 5, 11, 10, 11, 0, 3, -1, -1, -1, -1},
{5, 4, 8, 5, 8, 10, 10, 8, 11, -1, -1, -1, -1, -1, -1, -1},
{9, 7, 8, 5, 7, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{9, 3, 0, 9, 5, 3, 5, 7, 3, -1, -1, -1, -1, -1, -1, -1},
{0, 7, 8, 0, 1, 7, 1, 5, 7, -1, -1, -1, -1, -1, -1, -1},
{1, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{9, 7, 8, 9, 5, 7, 10, 1, 2, -1, -1, -1, -1, -1, -1, -1},
{10, 1, 2, 9, 5, 0, 5, 3, 0, 5, 7, 3, -1, -1, -1, -1},
{8, 0, 2, 8, 2, 5, 8, 5, 7, 10, 5, 2, -1, -1, -1, -1},
{2, 10, 5, 2, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1, -1},
{7, 9, 5, 7, 8, 9, 3, 11, 2, -1, -1, -1, -1, -1, -1, -1},
{9, 5, 7, 9, 7, 2, 9, 2, 0, 2, 7, 11, -1, -1, -1, -1},
{2, 3, 11, 0, 1, 8, 1, 7, 8, 1, 5, 7, -1, -1, -1, -1},
{11, 2, 1, 11, 1, 7, 7, 1, 5, -1, -1, -1, -1, -1, -1, -1},
{9, 5, 8, 8, 5, 7, 10, 1, 3, 10, 3, 11, -1, -1, -1, -1},
{5, 7, 0, 5, 0, 9, 7, 11, 0, 1, 0, 10, 11, 10, 0, -1},
{11, 10, 0, 11, 0, 3, 10, 5, 0, 8, 0, 7, 5, 7, 0, -1},
{11, 10, 5, 7, 11, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{10, 6, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 8, 3, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{9, 0, 1, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{1, 8, 3, 1, 9, 8, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1},
{1, 6, 5, 2, 6, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{1, 6, 5, 1, 2, 6, 3, 0, 8, -1, -1, -1, -1, -1, -1, -1},
{9, 6, 5, 9, 0, 6, 0, 2, 6, -1, -1, -1, -1, -1, -1, -1},
{5, 9, 8, 5, 8, 2, 5, 2, 6, 3, 2, 8, -1, -1, -1, -1},
{2, 3, 11, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{11, 0, 8, 11, 2, 0, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1},
{0, 1, 9, 2, 3, 11, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1},
{5, 10, 6, 1, 9, 2, 9, 11, 2, 9, 8, 11, -1, -1, -1, -1},
{6, 3, 11, 6, 5, 3, 5, 1, 3, -1, -1, -1, -1, -1, -1, -1},
{0, 8, 11, 0, 11, 5, 0, 5, 1, 5, 11, 6, -1, -1, -1, -1},
{3, 11, 6, 0, 3, 6, 0, 6, 5, 0, 5, 9, -1, -1, -1, -1},
{6, 5, 9, 6, 9, 11, 11, 9, 8, -1, -1, -1, -1, -1, -1, -1},
{5, 10, 6, 4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{4, 3, 0, 4, 7, 3, 6, 5, 10, -1, -1, -1, -1, -1, -1, -1},
```

```
{1, 9, 0, 5, 10, 6, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1},
{10, 6, 5, 1, 9, 7, 1, 7, 3, 7, 9, 4, -1, -1, -1, -1},
{6, 1, 2, 6, 5, 1, 4, 7, 8, -1, -1, -1, -1, -1, -1, -1},
{1, 2, 5, 5, 2, 6, 3, 0, 4, 3, 4, 7, -1, -1, -1, -1},
{8, 4, 7, 9, 0, 5, 0, 6, 5, 0, 2, 6, -1, -1, -1, -1},
{7, 3, 9, 7, 9, 4, 3, 2, 9, 5, 9, 6, 2, 6, 9, -1},
{3, 11, 2, 7, 8, 4, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1},
{5, 10, 6, 4, 7, 2, 4, 2, 0, 2, 7, 11, -1, -1, -1, -1},
{0, 1, 9, 4, 7, 8, 2, 3, 11, 5, 10, 6, -1, -1, -1, -1},
{9, 2, 1, 9, 11, 2, 9, 4, 11, 7, 11, 4, 5, 10, 6, -1},
{8, 4, 7, 3, 11, 5, 3, 5, 1, 5, 11, 6, -1, -1, -1, -1},
{5, 1, 11, 5, 11, 6, 1, 0, 11, 7, 11, 4, 0, 4, 11, -1},
{0, 5, 9, 0, 6, 5, 0, 3, 6, 11, 6, 3, 8, 4, 7, -1},
{6, 5, 9, 6, 9, 11, 4, 7, 9, 7, 11, 9, -1, -1, -1, -1},
{10, 4, 9, 6, 4, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{4, 10, 6, 4, 9, 10, 0, 8, 3, -1, -1, -1, -1, -1, -1, -1},
{10, 0, 1, 10, 6, 0, 6, 4, 0, -1, -1, -1, -1, -1, -1, -1},
{8, 3, 1, 8, 1, 6, 8, 6, 4, 6, 1, 10, -1, -1, -1, -1},
{1, 4, 9, 1, 2, 4, 2, 6, 4, -1, -1, -1, -1, -1, -1, -1},
{3, 0, 8, 1, 2, 9, 2, 4, 9, 2, 6, 4, -1, -1, -1, -1},
{0, 2, 4, 4, 2, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{8, 3, 2, 8, 2, 4, 4, 2, 6, -1, -1, -1, -1, -1, -1, -1},
{10, 4, 9, 10, 6, 4, 11, 2, 3, -1, -1, -1, -1, -1, -1, -1},
{0, 8, 2, 2, 8, 11, 4, 9, 10, 4, 10, 6, -1, -1, -1, -1},
{3, 11, 2, 0, 1, 6, 0, 6, 4, 6, 1, 10, -1, -1, -1, -1},
{6, 4, 1, 6, 1, 10, 4, 8, 1, 2, 1, 11, 8, 11, 1, -1},
{9, 6, 4, 9, 3, 6, 9, 1, 3, 11, 6, 3, -1, -1, -1, -1},
{8, 11, 1, 8, 1, 0, 11, 6, 1, 9, 1, 4, 6, 4, 1, -1},
{3, 11, 6, 3, 6, 0, 0, 6, 4, -1, -1, -1, -1, -1, -1, -1},
{6, 4, 8, 11, 6, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{7, 10, 6, 7, 8, 10, 8, 9, 10, -1, -1, -1, -1, -1, -1, -1},
{0, 7, 3, 0, 10, 7, 0, 9, 10, 6, 7, 10, -1, -1, -1, -1},
{10, 6, 7, 1, 10, 7, 1, 7, 8, 1, 8, 0, -1, -1, -1, -1},
{10, 6, 7, 10, 7, 1, 1, 7, 3, -1, -1, -1, -1, -1, -1, -1},
{1, 2, 6, 1, 6, 8, 1, 8, 9, 8, 6, 7, -1, -1, -1, -1},
{2, 6, 9, 2, 9, 1, 6, 7, 9, 0, 9, 3, 7, 3, 9, -1},
{7, 8, 0, 7, 0, 6, 6, 0, 2, -1, -1, -1, -1, -1, -1, -1},
{7, 3, 2, 6, 7, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{2, 3, 11, 10, 6, 8, 10, 8, 9, 8, 6, 7, -1, -1, -1, -1},
{2, 0, 7, 2, 7, 11, 0, 9, 7, 6, 7, 10, 9, 10, 7, -1},
{1, 8, 0, 1, 7, 8, 1, 10, 7, 6, 7, 10, 2, 3, 11, -1},
{11, 2, 1, 11, 1, 7, 10, 6, 1, 6, 7, 1, -1, -1, -1, -1},
{8, 9, 6, 8, 6, 7, 9, 1, 6, 11, 6, 3, 1, 3, 6, -1},
{0, 9, 1, 11, 6, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{7, 8, 0, 7, 0, 6, 3, 11, 0, 11, 6, 0, -1, -1, -1, -1},
{7, 11, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{7, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{3, 0, 8, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 1, 9, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{8, 1, 9, 8, 3, 1, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1},
{10, 1, 2, 6, 11, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{1, 2, 10, 3, 0, 8, 6, 11, 7, -1, -1, -1, -1, -1, -1, -1},
{2, 9, 0, 2, 10, 9, 6, 11, 7, -1, -1, -1, -1, -1, -1, -1},
{6, 11, 7, 2, 10, 3, 10, 8, 3, 10, 9, 8, -1, -1, -1, -1},
{7, 2, 3, 6, 2, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{7, 0, 8, 7, 6, 0, 6, 2, 0, -1, -1, -1, -1, -1, -1, -1},
{2, 7, 6, 2, 3, 7, 0, 1, 9, -1, -1, -1, -1, -1, -1, -1},
{1, 6, 2, 1, 8, 6, 1, 9, 8, 8, 7, 6, -1, -1, -1, -1},
{10, 7, 6, 10, 1, 7, 1, 3, 7, -1, -1, -1, -1, -1, -1, -1},
{10, 7, 6, 1, 7, 10, 1, 8, 7, 1, 0, 8, -1, -1, -1, -1},
{0, 3, 7, 0, 7, 10, 0, 10, 9, 6, 10, 7, -1, -1, -1, -1},
{7, 6, 10, 7, 10, 8, 8, 10, 9, -1, -1, -1, -1, -1, -1, -1},
{6, 8, 4, 11, 8, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{3, 6, 11, 3, 0, 6, 0, 4, 6, -1, -1, -1, -1, -1, -1, -1},
{8, 6, 11, 8, 4, 6, 9, 0, 1, -1, -1, -1, -1, -1, -1, -1},
{9, 4, 6, 9, 6, 3, 9, 3, 1, 11, 3, 6, -1, -1, -1, -1},
{6, 8, 4, 6, 11, 8, 2, 10, 1, -1, -1, -1, -1, -1, -1, -1},
{1, 2, 10, 3, 0, 11, 0, 6, 11, 0, 4, 6, -1, -1, -1, -1},
{4, 11, 8, 4, 6, 11, 0, 2, 9, 2, 10, 9, -1, -1, -1, -1},
{10, 9, 3, 10, 3, 2, 9, 4, 3, 11, 3, 6, 4, 6, 3, -1},
{8, 2, 3, 8, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1, -1},
```

```
{0, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{1, 9, 0, 2, 3, 4, 2, 4, 6, 4, 3, 8, -1, -1, -1, -1},
{1, 9, 4, 1, 4, 2, 2, 4, 6, -1, -1, -1, -1, -1, -1, -1},
{8, 1, 3, 8, 6, 1, 8, 4, 6, 6, 10, 1, -1, -1, -1, -1},
{10, 1, 0, 10, 0, 6, 6, 0, 4, -1, -1, -1, -1, -1, -1, -1},
{4, 6, 3, 4, 3, 8, 6, 10, 3, 0, 3, 9, 10, 9, 3, -1},
{10, 9, 4, 6, 10, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{4, 9, 5, 7, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 8, 3, 4, 9, 5, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1},
{5, 0, 1, 5, 4, 0, 7, 6, 11, -1, -1, -1, -1, -1, -1, -1},
{11, 7, 6, 8, 3, 4, 3, 5, 4, 3, 1, 5, -1, -1, -1, -1},
{9, 5, 4, 10, 1, 2, 7, 6, 11, -1, -1, -1, -1, -1, -1, -1},
{6, 11, 7, 1, 2, 10, 0, 8, 3, 4, 9, 5, -1, -1, -1, -1},
{7, 6, 11, 5, 4, 10, 4, 2, 10, 4, 0, 2, -1, -1, -1, -1},
{3, 4, 8, 3, 5, 4, 3, 2, 5, 10, 5, 2, 11, 7, 6, -1},
{7, 2, 3, 7, 6, 2, 5, 4, 9, -1, -1, -1, -1, -1, -1, -1},
{9, 5, 4, 0, 8, 6, 0, 6, 2, 6, 8, 7, -1, -1, -1, -1},
{3, 6, 2, 3, 7, 6, 1, 5, 0, 5, 4, 0, -1, -1, -1, -1},
{6, 2, 8, 6, 8, 7, 2, 1, 8, 4, 8, 5, 1, 5, 8, -1},
{9, 5, 4, 10, 1, 6, 1, 7, 6, 1, 3, 7, -1, -1, -1, -1},
{1, 6, 10, 1, 7, 6, 1, 0, 7, 8, 7, 0, 9, 5, 4, -1},
{4, 0, 10, 4, 10, 5, 0, 3, 10, 6, 10, 7, 3, 7, 10, -1},
{7, 6, 10, 7, 10, 8, 5, 4, 10, 4, 8, 10, -1, -1, -1, -1},
{6, 9, 5, 6, 11, 9, 11, 8, 9, -1, -1, -1, -1, -1, -1, -1},
{3, 6, 11, 0, 6, 3, 0, 5, 6, 0, 9, 5, -1, -1, -1, -1},
{0, 11, 8, 0, 5, 11, 0, 1, 5, 5, 6, 11, -1, -1, -1, -1},
{6, 11, 3, 6, 3, 5, 5, 3, 1, -1, -1, -1, -1, -1, -1, -1},
{1, 2, 10, 9, 5, 11, 9, 11, 8, 11, 5, 6, -1, -1, -1, -1},
{0, 11, 3, 0, 6, 11, 0, 9, 6, 5, 6, 9, 1, 2, 10, -1},
{11, 8, 5, 11, 5, 6, 8, 0, 5, 10, 5, 2, 0, 2, 5, -1},
{6, 11, 3, 6, 3, 5, 2, 10, 3, 10, 5, 3, -1, -1, -1, -1},
{5, 8, 9, 5, 2, 8, 5, 6, 2, 3, 8, 2, -1, -1, -1, -1},
{9, 5, 6, 9, 6, 0, 0, 6, 2, -1, -1, -1, -1, -1, -1, -1},
{1, 5, 8, 1, 8, 0, 5, 6, 8, 3, 8, 2, 6, 2, 8, -1},
{1, 5, 6, 2, 1, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{1, 3, 6, 1, 6, 10, 3, 8, 6, 5, 6, 9, 8, 9, 6, -1},
{10, 1, 0, 10, 0, 6, 9, 5, 0, 5, 6, 0, -1, -1, -1, -1},
{0, 3, 8, 5, 6, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{10, 5, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{11, 5, 10, 7, 5, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{11, 5, 10, 11, 7, 5, 8, 3, 0, -1, -1, -1, -1, -1, -1, -1},
{5, 11, 7, 5, 10, 11, 1, 9, 0, -1, -1, -1, -1, -1, -1, -1},
{10, 7, 5, 10, 11, 7, 9, 8, 1, 8, 3, 1, -1, -1, -1, -1},
{11, 1, 2, 11, 7, 1, 7, 5, 1, -1, -1, -1, -1, -1, -1, -1},
{0, 8, 3, 1, 2, 7, 1, 7, 5, 7, 2, 11, -1, -1, -1, -1},
{9, 7, 5, 9, 2, 7, 9, 0, 2, 2, 11, 7, -1, -1, -1, -1},
{7, 5, 2, 7, 2, 11, 5, 9, 2, 3, 2, 8, 9, 8, 2, -1},
{2, 5, 10, 2, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1, -1},
{8, 2, 0, 8, 5, 2, 8, 7, 5, 10, 2, 5, -1, -1, -1, -1},
{9, 0, 1, 5, 10, 3, 5, 3, 7, 3, 10, 2, -1, -1, -1, -1},
{9, 8, 2, 9, 2, 1, 8, 7, 2, 10, 2, 5, 7, 5, 2, -1},
{1, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{0, 8, 7, 0, 7, 1, 1, 7, 5, -1, -1, -1, -1, -1, -1, -1},
{9, 0, 3, 9, 3, 5, 5, 3, 7, -1, -1, -1, -1, -1, -1, -1},
{9, 8, 7, 5, 9, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{5, 8, 4, 5, 10, 8, 10, 11, 8, -1, -1, -1, -1, -1, -1, -1},
{5, 0, 4, 5, 11, 0, 5, 10, 11, 11, 3, 0, -1, -1, -1, -1},
{0, 1, 9, 8, 4, 10, 8, 10, 11, 10, 4, 5, -1, -1, -1, -1},
{10, 11, 4, 10, 4, 5, 11, 3, 4, 9, 4, 1, 3, 1, 4, -1},
{2, 5, 1, 2, 8, 5, 2, 11, 8, 4, 5, 8, -1, -1, -1, -1},
{0, 4, 11, 0, 11, 3, 4, 5, 11, 2, 11, 1, 5, 1, 11, -1},
{0, 2, 5, 0, 5, 9, 2, 11, 5, 4, 5, 8, 11, 8, 5, -1},
{9, 4, 5, 2, 11, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{2, 5, 10, 3, 5, 2, 3, 4, 5, 3, 8, 4, -1, -1, -1, -1},
{5, 10, 2, 5, 2, 4, 4, 2, 0, -1, -1, -1, -1, -1, -1, -1},
{3, 10, 2, 3, 5, 10, 3, 8, 5, 4, 5, 8, 0, 1, 9, -1},
{5, 10, 2, 5, 2, 4, 1, 9, 2, 9, 4, 2, -1, -1, -1, -1},
{8, 4, 5, 8, 5, 3, 3, 5, 1, -1, -1, -1, -1, -1, -1, -1},
{0, 4, 5, 1, 0, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{8, 4, 5, 8, 5, 3, 9, 0, 5, 0, 3, 5, -1, -1, -1, -1},
{9, 4, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
```

```
        {4, 11, 7, 4, 9, 11, 9, 10, 11, -1, -1, -1, -1, -1, -1, -1},
        {0, 8, 3, 4, 9, 7, 9, 11, 7, 9, 10, 11, -1, -1, -1, -1},
        {1, 10, 11, 1, 11, 4, 1, 4, 0, 7, 4, 11, -1, -1, -1, -1},
        {3, 1, 4, 3, 4, 8, 1, 10, 4, 7, 4, 11, 10, 11, 4, -1},
        {4, 11, 7, 9, 11, 4, 9, 2, 11, 9, 1, 2, -1, -1, -1, -1},
        {9, 7, 4, 9, 11, 7, 9, 1, 11, 2, 11, 1, 0, 8, 3, -1},
        {11, 7, 4, 11, 4, 2, 2, 4, 0, -1, -1, -1, -1, -1, -1, -1},
        {11, 7, 4, 11, 4, 2, 8, 3, 4, 3, 2, 4, -1, -1, -1, -1},
        {2, 9, 10, 2, 7, 9, 2, 3, 7, 7, 4, 9, -1, -1, -1, -1},
        {9, 10, 7, 9, 7, 4, 10, 2, 7, 8, 7, 0, 2, 0, 7, -1},
        {3, 7, 10, 3, 10, 2, 7, 4, 10, 1, 10, 0, 4, 0, 10, -1},
        {1, 10, 2, 8, 7, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {4, 9, 1, 4, 1, 7, 7, 1, 3, -1, -1, -1, -1, -1, -1, -1},
        {4, 9, 1, 4, 1, 7, 0, 8, 1, 8, 7, 1, -1, -1, -1, -1},
        {4, 0, 3, 7, 4, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {4, 8, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {9, 10, 8, 10, 11, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {3, 0, 9, 3, 9, 11, 11, 9, 10, -1, -1, -1, -1, -1, -1, -1},
        {0, 1, 10, 0, 10, 8, 8, 10, 11, -1, -1, -1, -1, -1, -1, -1},
        {3, 1, 10, 11, 3, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {1, 2, 11, 1, 11, 9, 9, 11, 8, -1, -1, -1, -1, -1, -1, -1},
        {3, 0, 9, 3, 9, 11, 1, 2, 9, 2, 11, 9, -1, -1, -1, -1},
        {0, 2, 11, 8, 0, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {3, 2, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {2, 3, 8, 2, 8, 10, 10, 8, 9, -1, -1, -1, -1, -1, -1, -1},
        {9, 10, 2, 0, 9, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {2, 3, 8, 2, 8, 10, 0, 1, 8, 1, 10, 8, -1, -1, -1, -1},
        {1, 10, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {1, 3, 8, 9, 1, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {0, 9, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {0, 3, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,-1}

};
```