UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

CENTRO DE NEUROBIOLOGÍA
CAMPUS UNAM-UAQ, JURIQUILLA, QRO.

ELABORACIÓN DE UNA TÉCNICA DE ANÁLISIS DE IMÁGENES PARA LA VISUALIZACIÓN DE ESTRUCTURAS CONTENIDAS EN CORTES SERIADOS

T E S I S

QUE PARA OBTENER EL GRADO DE

M A E S T R O E N C I E N C I A S

P R E S E N T A

GERARDO RAFAEL FAVILA HUMARA

TESIS DIRIGIDA POR
FERNANDO A. BARRIOS ÁLVAREZ

JURIQUILLA, QUERÉTARO 2002

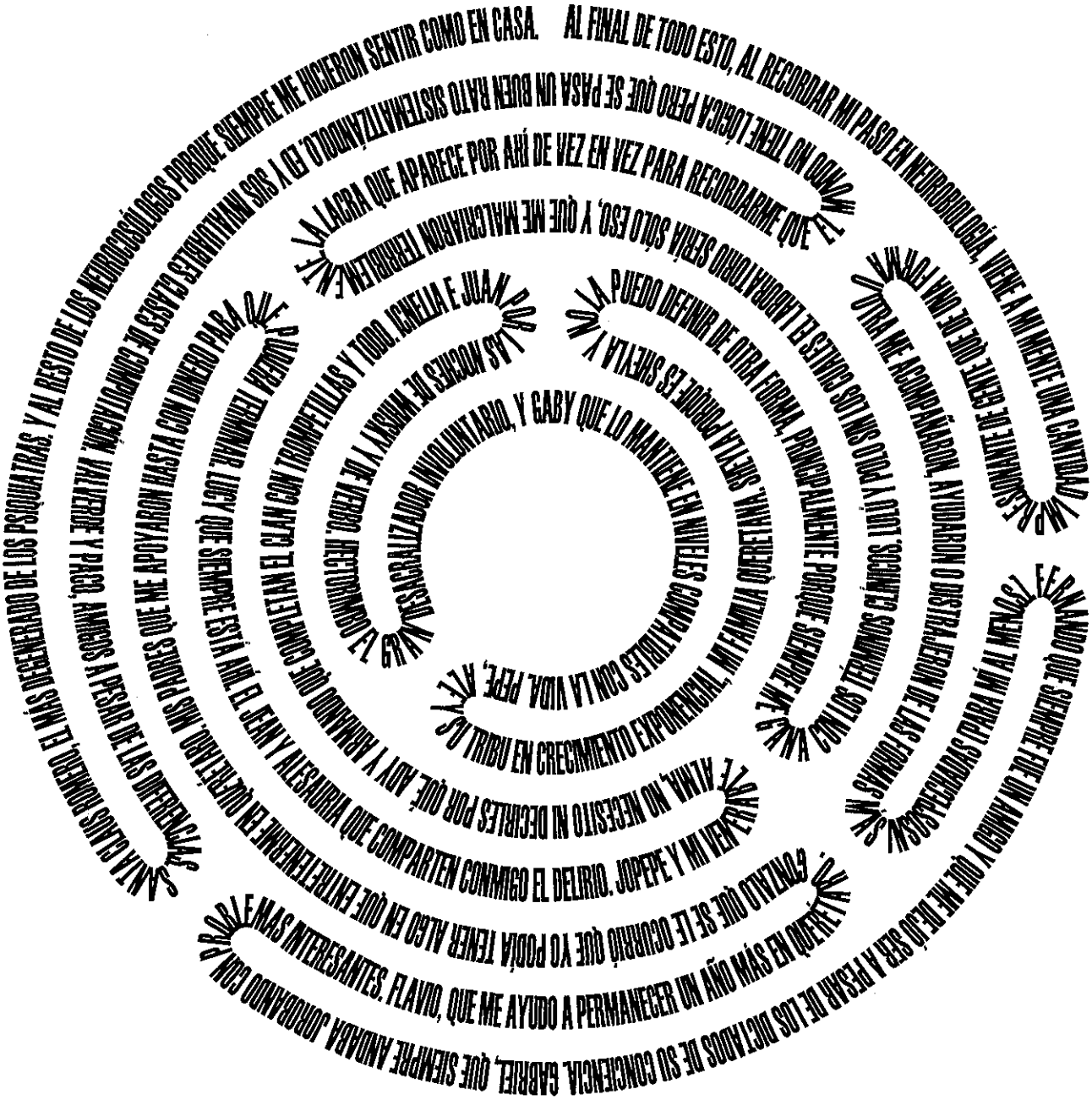
No matter how cynical you are, you can't keep up

Woody Allen

Este trabajo se llevó a cabo en la Unidad de Análisis de Imágenes del Centro de Neurobiología de la UNAM, bajo la dirección de Fernando Barrios Álvarez. Durante el tiempo de su realización conté con el apoyo de una beca crédito del Consejo Nacional de Ciencia y Tecnología y de una beca complemento de la Dirección General de Estudios de Posgrado de la UNAM.

Gracias quiero dar al divino
laberinto de los efectos y de las causas
por la diversidad de las criaturas
que forman este singular universo...

JI



Índice General

Parte I

Resumen

Resumen

En cualquier tarea de análisis, la forma en que presentemos los datos puede facilitarnos o dificultarnos la observación de las relaciones entre los elementos del diseño experimental. En biología es muy común extraer información de un elemento tridimensional (como el tejido cerebral de un animal de experimentación), cortándolo en rebanadas de manera que queden expuestas las estructuras internas con una mínima deformación. Mientras que esta metodología permite ver fácilmente la localización de las estructuras de interés, al mismo tiempo no permite ver con facilidad la manera en que éstas se relacionan en el contexto del objeto de estudio, cuya naturaleza es tridimensional.

Para vencer este obstáculo, en este trabajo se desarrolló un sistema metodológico que utiliza una serie de técnicas de procesamiento digital de imágenes que permiten recolectar y ordenar la información tisular bidimensional para, posteriormente, reconstruir estructuras de interés localizadas dentro del objeto original. El resultado de este trabajo es una metodología sencilla y general que permite reconstruir objetos tridimensionales a partir de elementos tomográficos seriados, aplicada al caso concreto de la representación de una gónada de tortuga en desarrollo y su inervación.

Abstract

In any analysis task, data presentation can either facilitate or impede the visualization of the relations between the experimental design elements. It is very common in Biology to extract information from tridimensional elements (such as the brain tissue from an experimental animal), cutting it in slices to expose its internal structures with minimum deformation. This methodology allows easy localization of structures of interest, but at the same time it does not permits an easy visualization of their relationship in tridimensional space.

To overcome this obstacle, in this work it was developed a methodology, using a series of digital image processing techniques, to collect and arrange bidimensinal tissue information, and render structures of interest located within the original object. The result is a simple and general methodology that allows tridimensional object reconstruction from tomographic serial elements. This methodology was applied to the case of the rendering a turtles gonad in development and its innervation.

Parte II

Introducción

La base operativa de este proyecto es el concepto de imagen digital. Una imagen óptica es un dominio continuo bidimensional con variaciones de color e intensidad luminosa en donde podemos identificar objetos. Si a este continuo de valores de color e intensidad le superponemos una cuadrícula rectangular, y a cada división le asignamos un valor de color que resulte representativo del área correspondiente de la imagen original, obtendremos una imagen digital (ver Figura 1). A esta estructura básica, originada por la división rectangular y asignación de valores de color, se le conoce como pixel (*picture element*) y como voxel (*volume element*) cuando la digitalización es tridimensional [1]. Una vez que una imagen ha sido digitalizada puede modificarse realizando operaciones matemáticas sobre los valores numéricos que representan los colores de cada pixel. Con este tipo de operaciones puede ajustarse el contraste, resaltar bordes, distinguir áreas de interés, eliminar fondo o ruido, etc.

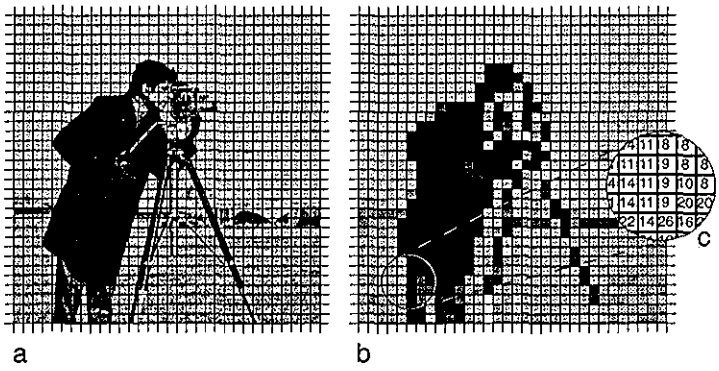


Figura 1: Proceso de digitalización. A la imagen óptica original se le superpone la rejilla de digitalización (a). El resultado es la imagen digitalizada (b), que podemos representar mediante su matriz numérica correspondiente (c).

Entre las aplicaciones más importantes del análisis de imágenes digital está la reconstrucción tridimensional de los objetos digitalizados. Este tipo de reconstrucción consiste en generar una estructura de dos o tres dimensiones que represente a un objeto tridimensional. La información sobre la estructura del objeto a reconstruir puede ser obtenida a partir de tomografía, de imágenes de cortes físicos (como los obtenidos mediante un microtomo), o de imágenes del objeto capturadas desde distintos puntos de vista. Esta reconstrucción nos permite observar con claridad la estructura tridimensional del objeto que estamos analizando.

El proceso para la reconstrucción tridimensional consta de cuatro etapas básicas:

1. Construcción de mosaicos y pilas: unir varias imágenes correspondientes al mismo plano (mosaico) o distintos planos (pila) de una imagen continua. Este proceso es necesario cuando

el objeto de estudio es mayor que el campo de observación.

2 Segmentación: identificación y clasificación de las estructuras contenidas en la imagen

3 Registro: alineamiento relativo de las estructuras ya identificadas. En la realización de esta operación se utilizan rotaciones, traslaciones y en algunos casos escalamiento y transformaciones elásticas (*warping*), para lograr una alineación óptima.

4 Construcción de proyecciones volumétricas: generación de un modelo geométrico tridimensional, o de una proyección plana que lo represente, de las estructuras segmentadas y alineadas. Este modelo se construye a partir de mallas u otras estructuras geométricas.

Es importante mencionar que no necesariamente es obligatorio aplicar estos cuatro procesos a todos los tipos de imágenes capturadas en microscopía.

Técnicas de imagen

Operaciones puntuales

Las operaciones puntuales son aquellas en las que se transforma una sola imagen de entrada (A) en una sola imagen de salida (B), de manera que el valor de cada pixel de la imagen B depende únicamente del valor del pixel correspondiente de la imagen A. Esta transformación podemos representarla mediante la siguiente fórmula:

$$D_B(x, y) = f(D_A(x, y))$$

donde $D_A(x, y)$ y $D_B(x, y)$ representan el valor del pixel (x, y) en las imágenes A y B, respectivamente.

La forma más sencilla de operación puntual es aquella en la que el nivel de gris de la imagen B depende linealmente del nivel de gris de la imagen A. En este caso la transformación adquiere la siguiente forma:

$$D_B(x, y) = aD_A(x, y) + b$$

donde a y b son números reales. Si $a = 1$ y $b = 0$, tenemos una función identidad que transcribe la imagen A en la imagen B. Si $a > 1$, aumentamos el contraste en la imagen B. Si $0 < a < 1$, el contraste se reduce. Si $a = 1$ y $b \neq 0$, la operación sólo recorre los valores de gris de todos los

pixeles Si $a < 0$, se invierten los valores de todos los pixeles (ver Figura 2) Este tipo de funciones se emplean normalmente para ajustar linealmente el contraste de una imagen. Para un ajuste óptimo se utiliza la función:

$$D_B = (D_A - D_m)\frac{D_R}{D_M - D_m} = D_A\frac{D_R}{D_M - D_m} + \frac{-D_m D_R}{D_M - D_m}$$

donde D_R es el valor máximo que admite la escala de gris que se está utilizando, D_M es el valor máximo de la imagen A y D_m es el mínimo.

Figura 2: Efecto de la aplicación de distintos filtros lineales a) original, b) efecto de negativo utilizando $a = -1$ y $b = 255$, c) reducción del contraste utilizando $a = 0.5$ y $b = 0$, d) aumento de contraste a la Figura c con $a = 2$ y $b = 0$, y e) traslación de los valores de gris utilizando $a = 0$ y $b = 128$ sobre la Figura c. Nótese que las Figuras a y d son equivalentes en cuanto al contraste, pero la Figura d posee la mitad de la resolución en la escala de grises debido al redondeo La Figura e se puede obtener directamente de la original utilizando $a = 0.5$ y $b = 128$

También se pueden definir operaciones puntuales no lineales, normalmente se utilizan funciones monótonas para no alterar el orden de los valores de gris Estas funciones se emplean frecuentemente para incrementar el contraste entre algunos niveles de gris a costa del contraste de los demás. Pueden tener formas muy diversas, pero las más utilizadas son las sigmoideas (con forma de S), que nos permiten incrementar el contraste entre los tonos de gris intermedios, y las funciones de ecualización, cuya forma varía de una imagen a otra y sirven para distribuir el contraste entre toda la escala de valores de gris (ver Figura 3)

Las operaciones puntuales no lineales se pueden utilizar para ajustar el contraste de una imagen para calibrarla y hacer que los valores de la escala de gris representen una propiedad física (por

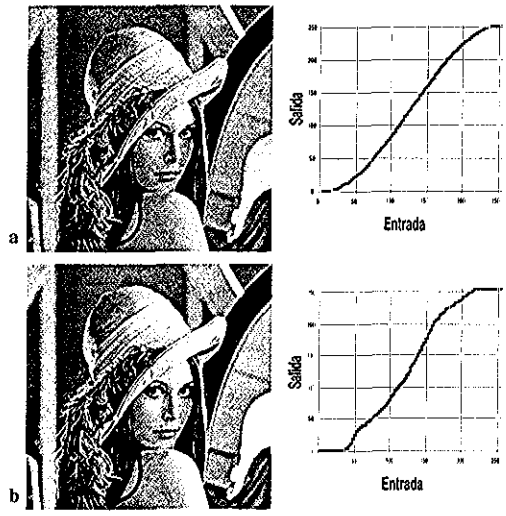


Figura 3: Aplicación de una función sigmoidea (a) y una función de ecualización (b) a la imagen de la Figura 2 a. A la derecha de cada imagen se puede ver la función de transferencia utilizada

ejemplo, en una imagen de un gel de electroforesis es deseable que la densidad óptica represente la densidad de la substancia que se está analizando), para segmentar imágenes, etc [2]

Operaciones de vecindad

Las operaciones de vecindad son aquellas en las que para determinar el valor que le corresponde a un pixel de la imagen B se realizan operaciones que involucran tanto al valor del pixel correspondiente como a los de sus vecinos.

Los filtros de vecindad más sencillos son los lineales, en estos se calcula el valor de cada pixel de la imagen de salida haciendo una suma ponderada de los valores de los pixeles de la región correspondiente de la imagen de entrada Esta relación se puede expresar con la fórmula:

$$D_B(x,y) = \sum_{j=-N}^{N} \sum_{i=-N}^{N} w_{ij} D_A(x+i, y+j)$$

donde N es el número de vecinos que vamos a tomar en consideración y W_{ij} es el peso del pixel que se encuentra en la posición (i,j) relativa al pixel (x,y). Por sencillez, esta suma se puede representar mediante el uso de *kernels*, matrices cuadradas como la de la Figura 4, en los que se representan los pesos de cada uno de los elementos de la suma conforme a su posición relativa al pixel central

Normalmente se divide el valor obtenido de la sumatoria entre la suma de todos los pesos para evitar obtener valores que se salgan de la escala de grises que se está trabajando

Existen tres aplicaciones básicas para los filtros de vecindad: reducción de ruido, detección de

0	1	2	1	0
1	6	10	6	1
2	10	16	10	2
1	6	10	6	1
0	1	2	1	0

Figura 4: Matriz de ponderación para filtro lineal de vecindad.

bordes y aumento del contraste dentro de una imagen

A la reducción de ruido obtenida por este tipo de filtros se le conoce también como promediación espacial, ésta consiste en promediar el valor de varios pixeles de la imagen de entrada para obtener el valor del pixel correspondiente de la imagen de salida. Este tipo de filtrado reduce la diferencia entre pixeles con valores muy distintos a los de sus vecinos, reduciendo el ruido a costa de reducir el detalle. En la Figura 5 a se ve una figura con ruido y en las Figuras 5 b y c se ve esta imagen después de haberles aplicado dos filtros de reducción de ruido. El filtro b tiene un *kernel* de 3×3 pixeles en el que se promedia al pixel central con doble peso y sus cuatro vecinos inmediatos, mientras que el filtro c tiene un *kernel* gaussiano de 7×7 pixeles que pondera a cada elemento de acuerdo con su posición relativa debajo de una curva normal bidimensional centrada en el pixel central.

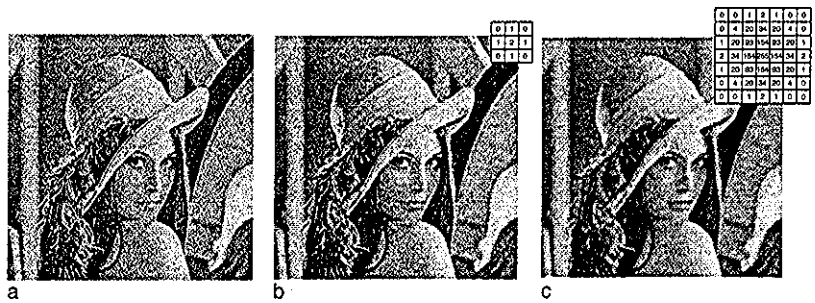


Figura 5: Efecto de la aplicación de distintos filtros de promediación espacial. a) imagen original, b) promediación ponderada simple con cuatro vecinos inmediatos, y c) gaussiano de 7×7 pixeles con $\sigma = 1$. En el ángulo superior derecho de cada figura se puede ver el *kernel* utilizado, en ambos casos se dividió el resultado entre la suma de los valores del *kernel*.

Los filtros de detección de bordes varían desde lo trivial hasta lo verdaderamente ingenioso. Los más sencillos consisten únicamente en restar a cada pixel el valor de uno de sus vecinos (Figura 6). Este tipo de operaciones nos permiten observar bordes en dirección horizontal, vertical y diagonal a 45 grados.

Otra forma de resaltar bordes consiste en ponderar negativamente los pixeles que se encuentran alrededor del pixel central. Si la suma de los valores negativos es igual a la suma de los valores positivos el resultado es que en una región homogénea el valor resultante es cero o cercano a cero,

Figura 6: Detección de bordes A cada pixel de la Figura 2.a se le restó el pixel de la izquierda, esto tiende a oscurecer las regiones homogéneas o en las cuales el pixel de la izquierda es más brillante, mientras que resalta los puntos en los que el pixel de la derecha es más brillante que el de la izquierda. Se ajustó el contraste de esta imagen para poder apreciar mejor la diferencia

mientras que en una región heterogénea este valor es distinto de cero, positivo o negativo Es común en la práctica dividir entre dos este valor y sumarle una constante (habitualmente el valor de la mitad de la escala de grises utilizada) para evitar que los valores negativos se salgan del rango de la escala de grises En la Figura 7 vemos el resultado de aplicar uno de estos filtros, conocidos genéricamente con el nombre de *sharpen*. Uno de los filtros más populares pertenecientes a esta categoría es el laplaciano, que es una aproximación de la segunda derivada lineal del brillo B en las direcciones x y y. El laplaciano es invariante a la rotación y, por lo tanto, a la dirección en la cual corre la discontinuidad. Una de las funciones generadoras de laplacianos es la campana de Gauss, cuya familia de filtros generados recibe el nombre de laplaciano de un gaussiano [1] En la Figura 8 podemos ver el efecto de la aplicación del laplaciano de un gaussiano, así como una proyección isométrica de la superficie del *kernel* laplaciano

Figura 7: Aplicación de un filtro *sharpen* El efecto mostrado en la imagen b es el resultado de restar el valor de sus ocho vecinos inmediatos a cada pixel de la imagen a, multiplicado por ocho Obsérvese que se acentúan las áreas en las que existe contraste

La mayoría de los operadores de detección de bordes se basan en la idea de que las interfaces entre estructuras presentan una variación intensa en el valor de sus pixeles Siendo esta variación

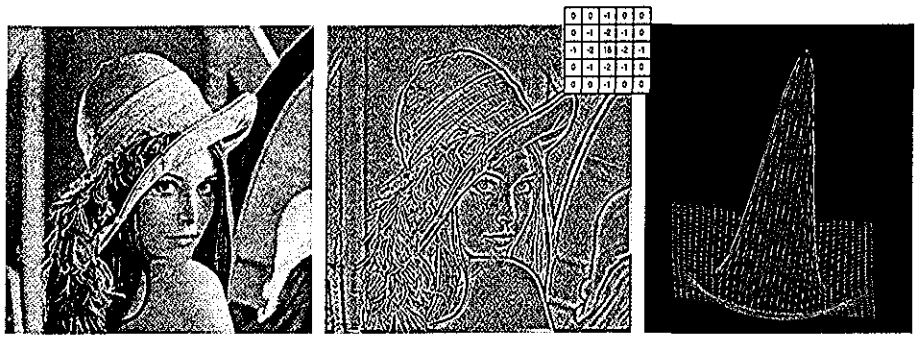


Figura 8: Filtro Laplaciano La imagen b muestra el efecto de aplicar un filtro laplaciano de 5×5 pixeles a la imagen b Obsérvese cómo resaltan los bordes en todas direcciones La imagen c muestra una proyección isométrica de un *kernel* laplaciano típico

un cambio, es natural modelarla mediante el uso de derivadas, de manera que aquellas regiones que presenten una pendiente acentuada tendrán mayor probabilidad de pertenecer a alguna frontera

El operador de Sobel [3] estima la magnitud de la derivada en dos direcciones ortogonales (generalmente las direcciones x y y de la imagen) de la siguiente forma:

$$M = \sqrt{\left(\frac{\partial D}{\partial x}\right)^2 + \left(\frac{\partial D}{\partial y}\right)^2}$$

donde M es la magnitud de la derivada y D representa el valor de los pixeles. La magnitud de las derivadas la estima mediante la aplicación de un par de *kernels* que acentúan la diferencia entre pixeles en una dirección determinada En la Figura 9 se muestran algunos *kernels* que se pueden utilizar para este cálculo.

Además de la magnitud, es posible determinar la dirección de la pendiente mediante la fórmula:

$$\alpha = \arctan\left(\frac{\partial D/\partial y}{\partial D/\partial x}\right)$$

donde α es la dirección principal de cambio En la Figura 10 se puede ver un ejemplo de la aplicación de este filtro.

Segmentación

La segmentación consiste en separar la imagen en varias regiones que tengan significado para una tarea determinada Esta segmentación puede hacerse utilizando el valor de los pixeles, la magnitud de los gradientes de la imagen, o alguna medida de textura Las técnicas de segmentación pueden

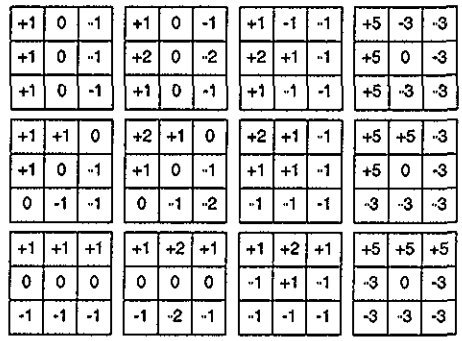


Figura 9: Ejemplos de *kernels* utilizados para el cálculo de la magnitud de las derivadas parciales en el filtro Sobel

estar dirigidas al reconocimiento de áreas, de sus bordes, o pueden estar basadas en algún método estadístico de clasificación.

Uno de los métodos más sencillos de segmentación es el *thresholding* o segmentación por umbral. En el caso más sencillo, nos permite separar dos regiones (objeto y fondo) que se encuentren bien contrastadas, seleccionando un punto de la escala de grises que las separe por completo. En un caso un poco más complejo, se pueden seleccionar varios umbrales para separar más regiones. La selección del umbral puede hacerse de manera manual, o automática. El método automático más común supone que el histograma es la suma de dos más distribuciones normales, y coloca los umbrales en los valles del histograma.

Alineamiento

Ya que se han segmentado e identificado las estructuras de interés, el siguiente paso es alinear todos los cortes. En el caso de imágenes obtenidas por microscopía de muestras biológicas, este paso es

Figura 10: Aplicación del filtro Sobel a la imagen a: b) magnitud, c) dirección. Ambos con el contraste ajustado para observar mejor el efecto del filtro.

necesario porque normalmente cada sección es digitalizada en la posición en que fue montada en el portaobjetos, de manera que el sistema de coordenadas propias para cada sección es diferente.

Para lograr que la orientación de todas las secciones sea la misma es necesario realizar dos operaciones fundamentales: rotación y traslación [4] El ángulo de la rotación y la magnitud de la traslación pueden ser determinados por distintos métodos; a continuación se mencionan algunos de los más comunes [1, 2, 5].

Alineamiento manual

Consiste en ajustar manualmente la rotación y la traslación Algunos programas ajustan el alineamiento de cada sección mostrándola semitransparente sobre una sección de referencia, de manera que el usuario puede ver cómo se ajusta el alineamiento con cada variación de los parámetros de rotación y traslación

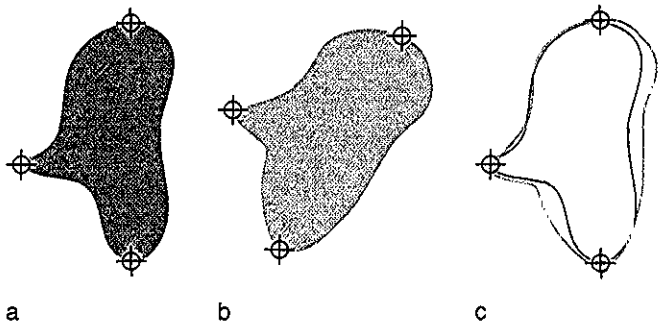


Figura 11: Alineamiento por puntos de referencia Las imágenes a y b corresponden a dos rebanadas de una misma muestra, en ambas se han marcado tres puntos de referencia. En la imagen c se pueden ver los contornos de ambas figuras después de haber hecho coincidir los puntos de referencia

Reconocimiento de puntos de referencia

Otro método muy utilizado es marcar como puntos de referencia (*landmarks*) a algunas estructuras, que sean características y se mantengan en la misma posición a lo largo de la muestra, y alinear las rebanadas haciendo coincidir estas marcas Si no existen estas estructuras, se puede perforar la muestra (o el medio de montaje) con un taladro o aguja [6, 7].

Métodos basados en la intensidad de los pixeles

Una aproximación muy utilizada consiste en transformar secuencialmente la imagen a alinear, y compararla con la imagen base con alguna medida que refleje que tan bien alineadas se encuentran,

después de un número predeterminado de iteraciones, se transforma la imagen con los parámetros que maximicen el valor de la función de similitud (o minimicen, en el caso de una función de disparidad) Esta aproximación es válida cuando suponemos que no existe mucha diferencia entre las dos imágenes, y puede dar un mal alineamiento si esto no se cumple o si unos cuantos pixeles varían mucho en su intensidad de una imagen a otra

La función de disparidad más utilizada es la suma de los cuadrados de las diferencias (SCD):

$$SCD = \frac{1}{N}\sum_{i=1}^{N}\left(A(i) - B'(i)\right)^2 \qquad \forall i \in A \cap B'$$

donde A es la imagen original, B' es la imagen que se desea alinear después de haber sido transformada, y N es el número de pixeles existentes en $A \cap B'$

Si las imágenes A y B están relacionadas linealmente, en algunas aplicaciones (sobre todo de imágenes intermodales) se puede utilizar el coeficiente de correlación (CC) como medida de similitud

$$CC = \frac{\sum_{i=1}^{N}\left(A(i) - \bar{A}\right)\left(B'(i) - \bar{B'}\right)}{\sqrt{\sum_{i=1}^{N}\left(A(i) - \bar{A}\right)^2 \sum_{i=1}^{N}\left(B'(i) - \bar{B'}\right)^2}} \qquad \forall i \in A \cap B'$$

donde $\bar{A}$ y $\bar{B}$ son los valores medios de los pixeles de las imágenes A y B', respectivamente

Otra alternativa para el alineamiento intermodal es calcular la uniformidad de la imagen de proporción (UIP) [8] Para cada transformación de B se calcula una imagen de proporción (P) y se determina la uniformidad de P con la desviación estándar normalizada (el cociente de la desviación estándar entre la media) El algoritmo determina los parámetros de transformación que minimizan esta desviación normalizada

$$P(i) = \frac{B'(i)}{A(i)} \qquad \forall i \in A \cap B'$$

$$\mu_P = \frac{1}{N}\sum_{i=1}^{N} P(i)$$

$$\sigma_P = \frac{1}{N}\sum_{i=1}^{N}\left(P(i) - \mu_P\right)^2$$

$$UIP = \frac{\sigma_P}{\mu_P}$$

Técnicas de representación volumétricas

Una vez que contamos con una pila de imágenes segmentadas y debidamente alineadas surge el problema de representarlas de tal manera que se pueda observar de la forma más sencilla toda la información pertinente respecto al objeto de estudio. Esta representación gráfica de la información tridimensional recibe el nombre de *rendering* [9]

Proyección volumétrica

Si trazamos una serie de líneas paralelas, a través del volumen que nos interesa visualizar, podemos determinar qué voxeles atraviesa cada una de estas líneas (Ver Figura 12). Para formar la proyección del volumen asignamos a cada pixel de esta proyección un valor que depende de los valores de los voxeles que atraviesa cada una de estas líneas paralelas [10]. Este valor se puede asignar mediante cualquier función matemática, entre otras la función de absorción, el promedio, la suma aritmética, y el valor máximo o mínimo. En la Figura 13 se muestra un ejemplo de proyección por máximos

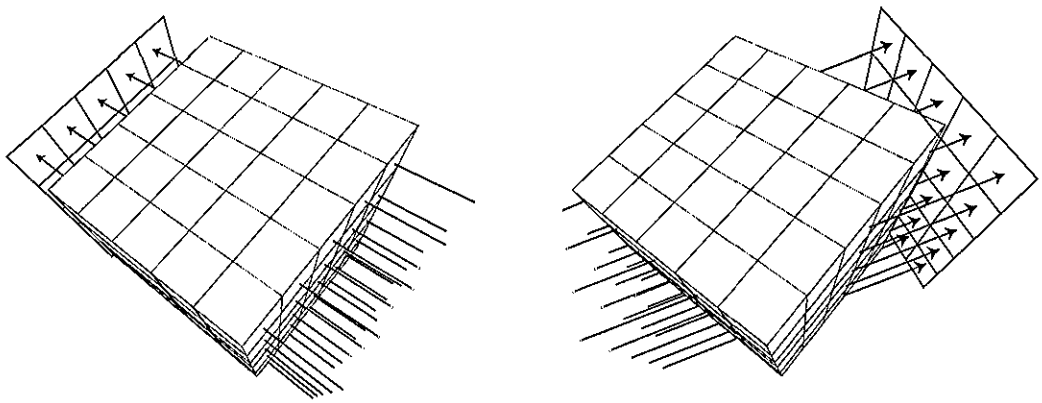


Figura 12: Reconstrucción por proyección. En la figura se muestran dos ejemplos de líneas paralelas que cruzan un volumen y son proyectadas en un plano (pantalla)

Reconstrucción por planos de corte

Este tipo de reconstrucción tridimensional consiste en graficar la imagen plana resultante de hacer un corte a través del volumen de interés (Ver Figura 14 a) Estos planos de corte pueden tener cualquier orientación; generalmente se utilizan cortes perpendiculares a los ejes del volumen [1]

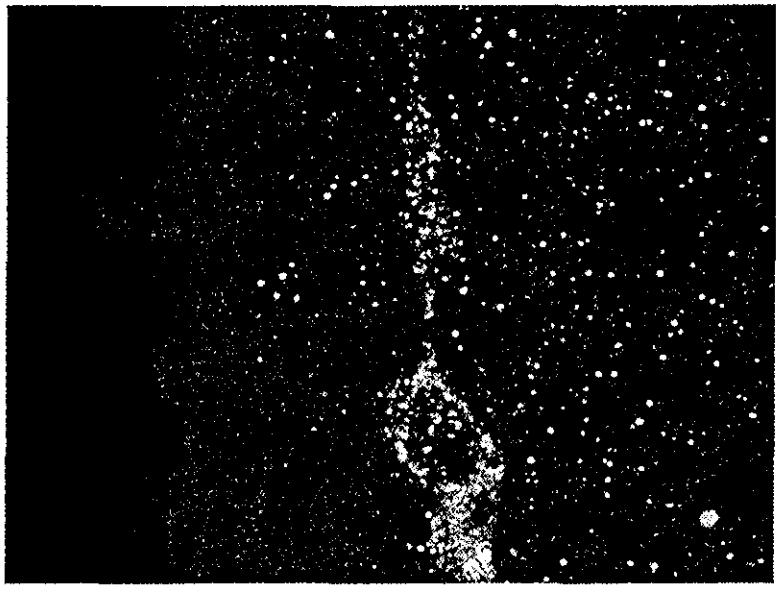


Figura 13: Proyección de valores máximos de una imagen de microscopía confocal.

Reconstrucción en voxeles

Esta reconstrucción tridimensional toma como base un volumen ya segmentado. A estos voxeles se les asigna un valor de color (y a veces también de transparencia) de acuerdo a algún modelo matemático o propiedad. Cada voxel es representado en un espacio tridimensional como un paralelepípedo cuyas dimensiones corresponden a la resolución espacial de la imagen [11]. Para encontrar una representación bidimensional de este modelo geométrico tridimensional se utilizan distintos algoritmos, por ejemplo *raytracing* [12, 13]. Ver Figura 14.b.

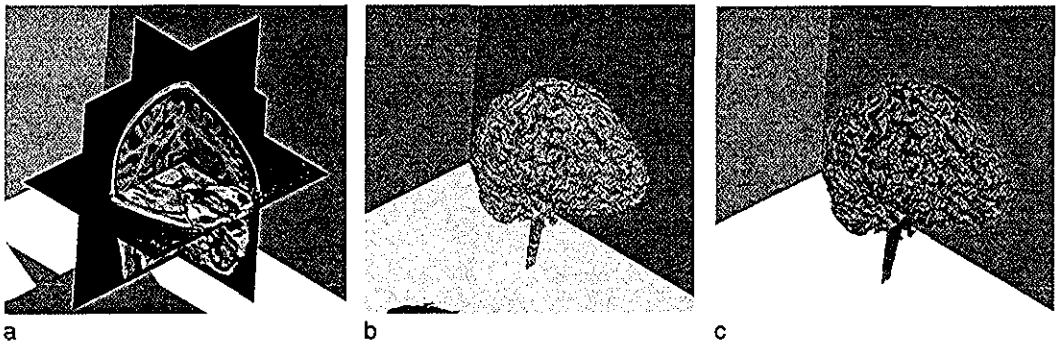


Figura 14: Reconstrucción tridimensional. a) Reconstrucción por planos de corte. b) Reconstrucción por voxeles. c) Reconstrucción por superficies.

Reconstrucción por superficies

En un volumen segmentado, podemos descartar todos aquellos voxeles que no se encuentran en la frontera de la estructura a la cual pertenecen. Los voxeles resultantes de cada estructura se unen formando una malla tridimensional (generalmente formada por triángulos), a la cual se le puede asignar un valor de color y transparencia que la caracterice [14]. Cada elemento de esta malla se puede representar como un elemento sólido que se proyecta en la pantalla mediante alguno de los algoritmos de *rendering* (Ver Figura 14.c)

Caso de estudio

El desarrollo en este proyecto de un sistema de análisis de imagen tiene una aplicación inmediata en la neurobiología; en el caso específico de este trabajo, se aplicaron estas técnicas en un problema del desarrollo del sistema nervioso de la tortuga marina *L. olivacea*

Los vertebrados muestran normalmente determinación sexual genética; sin embargo éste no es el caso en algunas especies de reptiles [15, 16]. El sexo de los cocodrilos, la mayoría de las tortugas y algunos lagartos depende de la interacción entre la temperatura y el embrión en desarrollo [17, 18]. Por ejemplo, en el caso de *L. olivacea* si se incuban los embriones a 32° C se obtienen hembras y machos a 27° C [19]. El mecanismo detrás de la diferenciación sexual dependiente de temperatura aún es desconocido. Se ha propuesto que la temperatura de incubación afecta directamente la capacidad de las gónadas indiferenciadas para producir estrógenos, y que a mayor concentración de estos se activa el mecanismo de diferenciación de la gónada a ovario. Este mecanismo implica que el factor que induce la diferenciación se encuentra dentro de la gónada. Sin embargo, se ha demostrado recientemente que la concentración de estrógenos no está correlacionada con la producción de hembras [20]. También se ha encontrado que no existe esta capacidad de diferenciación dependiente de temperatura en gónadas cultivadas organotrópicamente [21]. Estas observaciones sugieren que existen señales extragonadales que modulan la diferenciación. Los vertebrados han desarrollado diversos sistemas para monitorear cambios en la temperatura ambiental. La mayoría de estos sistemas conducen información de receptores localizados en la superficie del cuerpo al sistema nervioso central. Después de recibir e integrar esta información, el cerebro coordina una serie de respuestas fisiológicas y conductuales que regulan la temperatura corporal. Además, se ha demostrado recientemente que el cerebro en desarrollo de los reptiles responde directamente a cambios en la temperatura de incubación [22], y esta respuesta es independiente de los niveles circulantes de hormonas sexuales. También se han encontrado fibras nerviosas en gónadas indiferenciadas en

L. olivacea durante el periodo termosensible [19]. Esta evidencia sugiere que la diferenciación sexual puede deberse a una diferenciación del sistema nervioso.

Objetivos

- Desarrollar una técnica de análisis de imágenes, que permita visualizar en tres dimensiones estructuras contenidas en cortes seriados, como las obtenidas al cortar una muestra con el microtomo

Objetivos específicos

- Segmentar la inervación
- Segmentar la gónada
- Alinear las distintas imágenes
- Elaborar estructuras tridimensionales para *rendering*

Para lograr cada uno de estos objetivos se utilizará el programa comercial IPLab (*Scanalytics, Inc*). Los algoritmos que sea necesario programar serán codificados en el lenguaje de macros nativo de IPLab o como extensiones para IPLab en lenguaje C.

Hipótesis

Es posible desarrollar un sistema de análisis de imagen digital que, a partir de módulos programados, lleve a cabo la segmentación, el registro y la construcción de estructuras tridimensionales para realizar una reconstrucción tridimensional de imágenes digitalizadas de microscopía.

Parte III

Métodos

Metodología para el análisis de imagen

Este trabajo está basado en la manipulación de imágenes digitales utilizando el programa comercial IPLab Adicionalmente se utilizaron otros programas, algunos de ellos de dominio publico (*NIH Image, W Rasband, NIH*) y otros desarrollados enteramente en la Unidad de Análisis de Imagen del CNB en sistemas UNIX usando OpenGL (*Silicon Graphics, Inc*) y programas propietarios de SGI para hacer las proyecciones de los modelos tridimensionales

Estructura del archivo de imagen

En los procesos digitales de análisis de imágenes, éstas se representan principalmente como matrices de dos o más dimensiones Para facilitar el análisis, se ha unido dentro de estructuras a estas matrices con diversos campos que proporcionan información sobre la matriz misma, la imagen, procesos intermedios e interacción con el usuario Esta representación en el programa IPLab, que nosotros usaremos como base de trabajo, sigue la idea de representar a la matriz de imagen usando una estructura de localización dinámica en memoria de trabajo [23] Esta estructura tiene los siguientes campos:

La operación básica de los procesos en análisis de imagen, es el acceso de los elementos (pixeles) que la forman, para lectura y escritura de sus valores Este proceso de acceso se hace utilizando el puntero `fDataPtr` y los valores `fWidth` y `fHeight` de la estructura de la imagen El código en lenguaje C de los principales módulos se puede ver en el apéndice.

IPLab permite la creación de máscaras independientes del valor de los pixeles para la segmentación Estas máscaras pueden crearse de manera automática, mediante algunas de las funciones del programa, o manualmente. Esta máscara es almacenada en una estructura tipo PixMap, la cual almacena valores de 4 bits para cada pixel de manera que no ocupa tanto espacio como una imagen con datos tipo byte al mismo tiempo que permite hasta 16 valores distintos para cada punto de la máscara, lo que posibilita la segmentación simultánea de varias estructuras. Para facilitar la manipulación de las máscaras, siempre que fue necesario, se tradujo a imágenes tipo byte para no enfrentar los problemas asociados a leer y escribir datos menores a 1 byte

Generación de filtros lineales

IPLab tiene incluidos varios de los filtros lineales más utilizados, así como un editor de filtros. Este editor está limitado a *kernels* de 5×5 pixeles Sin embargo, IPLab puede aplicar filtros con *kernels* mayores que se encuentren definidos en forma de imagen Para poder generar automáticamente

Tabla 1: Estructura de Referencia de Imagen

Tipo de dato	Nombre del campo
typedef structure{	
WindowPtr	fWPtr
short	fWindowKind
DataTypes	fDatatype
long	fWidth
long	fHeight
short	fNumFrames
short	fCurrentFrame
Ptr	fDataPtr
Ptr	fFramePtr
Ptr	fDisplayPtr
Ptr	fCTablePtr
long	fROILeft
long	fROITop
long	fROIRight
long	fROIBottom
RgnHandle	fROIRegion
PolyHandle	fROIPolygon
Ptr	fDataMinPtr
Ptr	fDataMaxPtr
PixMapHandle	fOverlay
short	fDataChanged
short	fReNormalize
short	fCLUTChanged
short	fRecSize
} TImageReference;	

kernels de filtros gaussianos y laplacianos de cualquier tamaño, programamos módulos que pudieran generar *kernels* gaussianos y laplacianos de tamaño arbitrario

Metodología en el caso de estudio

Se digitalizaron los cortes con un microscopio Eclipse-600 de Nikon, utilizando una cámara MTI CCD72 y el programa IPLab 3 2 4 Las imágenes fueron almacenadas en el formato nativo de IPLab

Segmentación de la inervación

La identificación de las fibras colinérgicas se hizo de manera directa, mediante una función de *thresholding*, porque la técnica de acetilcolinesterasa con la que fueron tratadas las muestras las marca de un color oscuro que las hace ffácilmente distinguibles del resto del tejido

La segmentación de la gónada no fue tan sencilla porque ésta no se distingue por su color del fondo de la imagen. Para segmentar la gónada se probaron varias combinaciones de filtros digitales para suavizar el fondo, la gónada, para resaltar el borde de la gónada y de crecimiento de regiones [24]. Después de la aplicación de estos filtros se realizó una segmentación por *thresholding*.

Alineamiento

Para poder hacer el registro de las distintas rebanadas supusimos que los cortes se habían hecho perfectamente transversales al eje mayor de la gónada; y que si la gónada no mantenía su área de sección constante en el intervalo que analizamos, el grosor aumenta o disminuye proporcionalmente en ambas dimensiones.

Dados estos supuestos, la forma más sencilla de calcular la traslación que hay que aplicar para alinear es calcular la posición del centroide para cada una de las gónadas, y trasladarlas hasta el origen.

Para determinar el ángulo de rotación, probamos varias alternativas:

Alineamiento manual

Utilizando una rebanada base de fondo, hicimos rotaciones y traslaciones hasta encontrar un ajuste adecuado. Las rotaciones se hicieron utilizando la función `rotate & scale` incluida en IPLab, introduciendo manualmente los ángulos de rotación. La traslación se hizo arrastrando con el ratón cada imagen hasta hacerla coincidir con la imagen patrón.

Alinear al ángulo más frecuente

Se calculó el ángulo formado, con respecto al eje horizontal, por líneas trazadas a partir de cada uno de los pixeles pertenecientes a la gónada hasta el centroide. Esté ángulo fue calculado con la función `atan2` de lenguaje C, que regresa un valor entre 0 y 2π. Posteriormente se convirtieron los ángulos a grados y se redondearon a enteros y se seleccionó el ángulo más frecuente.

Mínima distancia entre centroides

Se tomo cada una de las gónadas segmentadas y se localizó su centroide. A partir del centroide se traza una línea horizontal que divide a la gónada en dos partes, se localiza el centroide de cada una de ellas y se calcula la distancia entre los centroides de las hemigónadas. Se repite el procedimiento

trazando líneas en ángulos comprendidos entre 0 y π radianes y se selecciona el ángulo que presente una menor distancia entre centroides.

Construcción de mallas tridimensionales

Para la reconstrucción de la gónada trazamos una malla triangular, cuyos nodos fueron las intersecciones del borde la gónada con líneas trazadas cada 20° a partir del centroide de cada rebanada

Para representar la inervación no fue posible conectar los distintos segmento mediante mallas, debido a que los segmentos correspondientes se encontraban separados debido a la deformación que sufre el tejido al cortarlo Así que elegimos representarlos como voxeles Para reducir el número de triángulos a representar juntamos todos aquellos pixeles contiguos que se encontraran en la misma línea como un sólo paralelogramo

El resultado final fue escrito en formato de *Open Inventor* y graficado en una máquina Indigo 2 con 256 MB de RAM utilizando el programa *SceneViewer* de *Silicon Graphics*.

Parte IV

Resultados

La captura de los cuadros por el sistema proporcionó una serie de imágenes digitalizadas, como la que se muestra en la Figura 15. Esta imagen muestra claramente a la inervación en color negro y la gónada y parte del mesonefros en un tono de gris intermedio. Tanto la gónada como el mesonefros presentan bordes irregulares (más el mesonefros, que se encuentra muy fragmentado) y regiones bastante amplias donde el nivel de gris desciende hasta igualarse con el del fondo

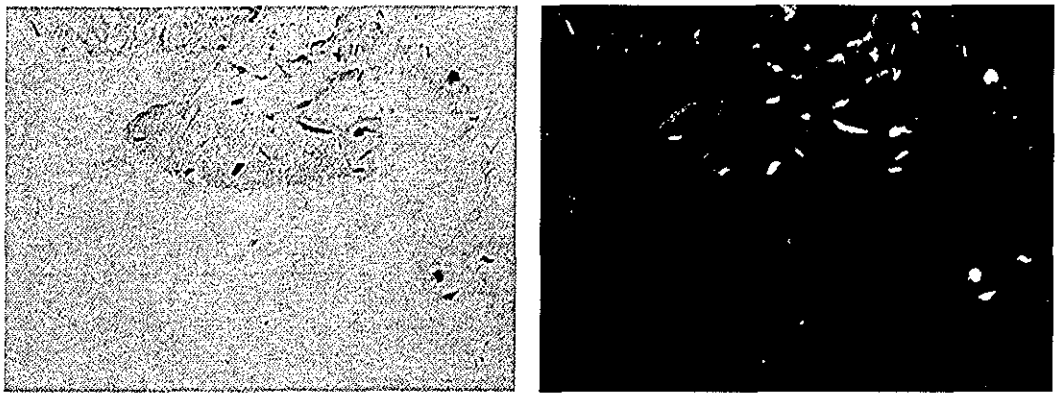
Figura 15: Imagen original digitalizada

Segmentación de la inervación

La tinción de acetilcolinesterasa marca en negro a las fibras colinérgicas, lo que permite localizarlas con mucha facilidad, ya que el resto de la preparación tiene un nivel de gris más bajo. Por esto, para segmentarlas, sólo fue necesario seleccionar los pixeles más oscuros. En los casos en que había impurezas oscuras en la muestra (identificadas fácilmente porque su forma difiere mucho del de los tractos nerviosos), éstas fueron borradas manualmente(ver Figura 16)

Segmentación de la gónada

La segmentación de la gónada no fue tan sencilla, debido al poco contraste que presenta con respecto al tejido circundante. La segmentación por textura, detección de bordes y crecimiento de regiones tampoco presentaron resultados favorables, debido a lo irregular del tejido (ver Figura 17).



a b

Figura 16: Segmentación de la inervación. La imagen b fue obtenida después de aplicar un *threshold* de manera manual a la imagen a

Probamos distintas combinaciones de filtros para lograr a) eliminar estas irregularidades y probar los métodos básicos, o b) incrementar estas irregularidades de manera localizada en la gónada (ver Figura 18). Obtuvimos un resultado positivo para nuestro problema utilizando un suavizamiento con un filtro laplaciano de 5×5 pixeles, seguido de un filtro de detección de bordes Sobel (Figura 18 h)

Alineamiento

El problema del alineamiento lo dividimos en dos partes: rotación y traslación. Dado que la forma más sencilla de componerlos es hacer primero la rotación y luego la traslación, así lo hicimos.

Supusimos que los cortes eran todos transversales al eje principal de la gónada, y que ésta presenta pocos cambios en su espesor a lo largo de la zona estudiada (los "casquetes" de la gónada no pudieron ser colectados). Si nuestros supuestos eran correctos, deberíamos poder calcular la traslación simplemente como la distancia que teníamos que mover cada una de las secciones para hacer coincidir sus centroides.

Para la rotación, el "método de los dos centroides" nos proporcionó un alineamiento más preciso que la rotación manual, y en mucho menos tiempo (ver Figura 19).

Construcción de mallas tridimensionales

La malla fue generada como describimos con anterioridad. Ésta generó un modelo con la resolución necesaria para distinguir las estructuras de interés. Este modelo utiliza memoria RAM de manera razonable, permitiendo un despliegue rápido en la pantalla y manipulación en tiempo real. En la

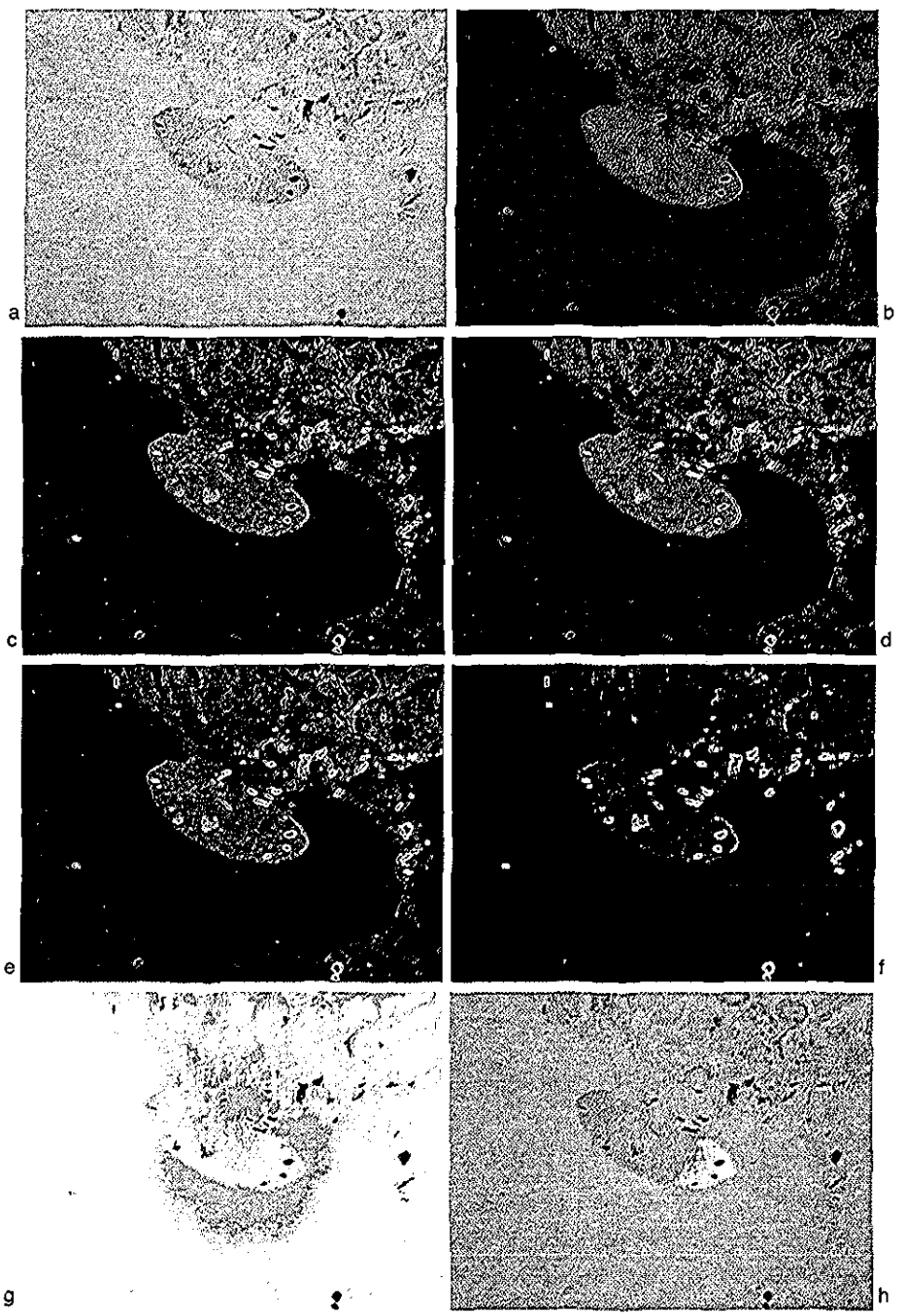


Figura 17: Segmentación de la gónada a) Imagen original; b) laplaciano 5 × 5; c) Roberts; d) Morphgrad; e) Sobel; f) segmentación por textura; g y h) crecimiento de regiones con distintas semillas. Se puede observar que ninguno de estos filtros consigue segmentar la gónada, ni delimitar su contorno

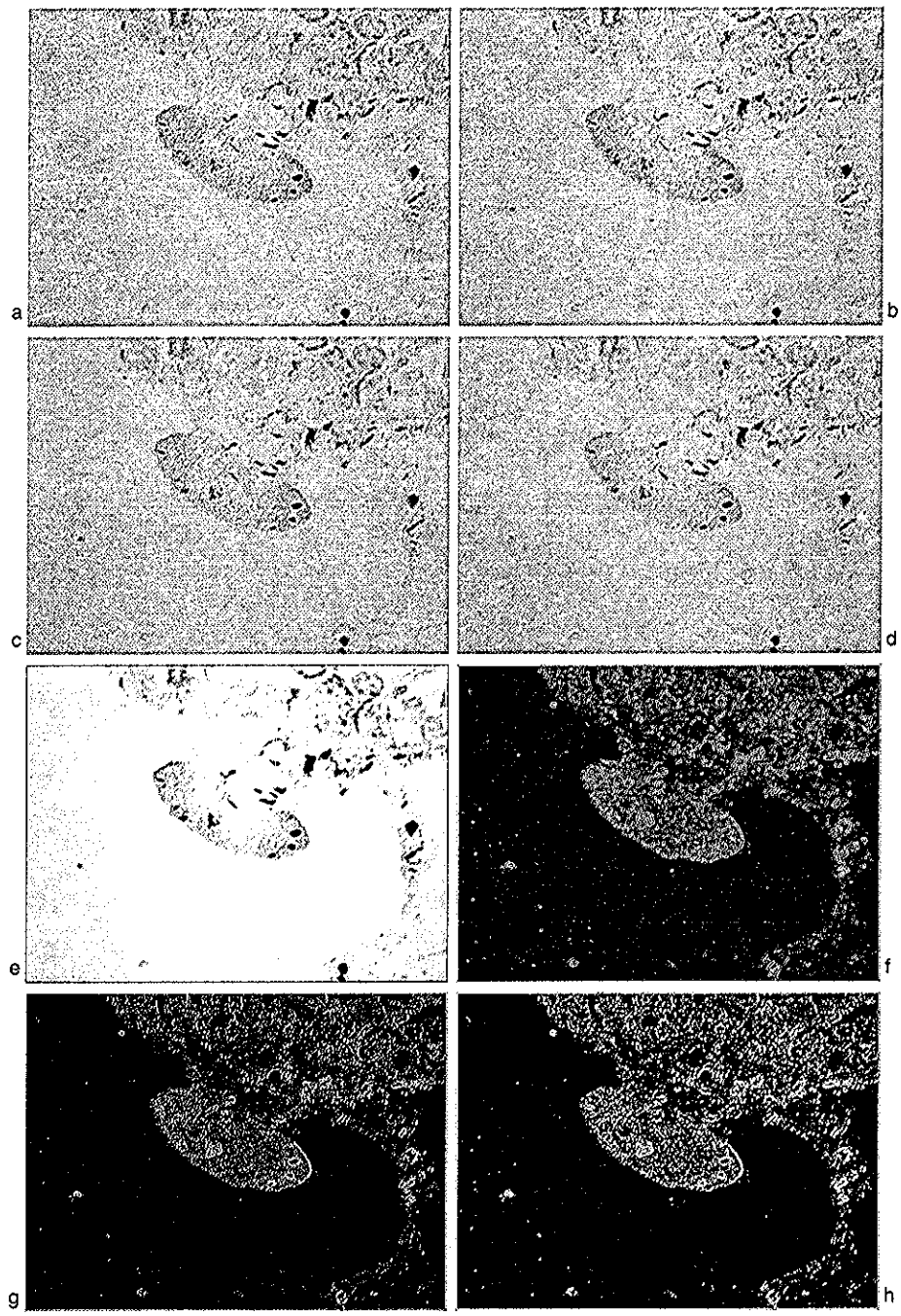


Figura 18: Segmentación de la gónada a) Imagen original; b, c y d) *blur*; e) gaussiano 5×5; f, g y h) aplicación de filtros morphgrad, Roberts y Sobel, respectivamente, a una imagen a la que previamente se le había aplicado un filtro laplaciano 5×5. Obsérvese que ninguno de los primeros cuatro filtros consigue homogeneizar el nivel de gris de la gónada , ni separarlo del fondo, pero que los últimos tres filtros (especialmente el h) consiguen unir gran parte de los puntos de borde previamente detectados por el laplaciano (ver Figura 17.b)

Figura 21 se pueden ver algunas vistas del modelo final.

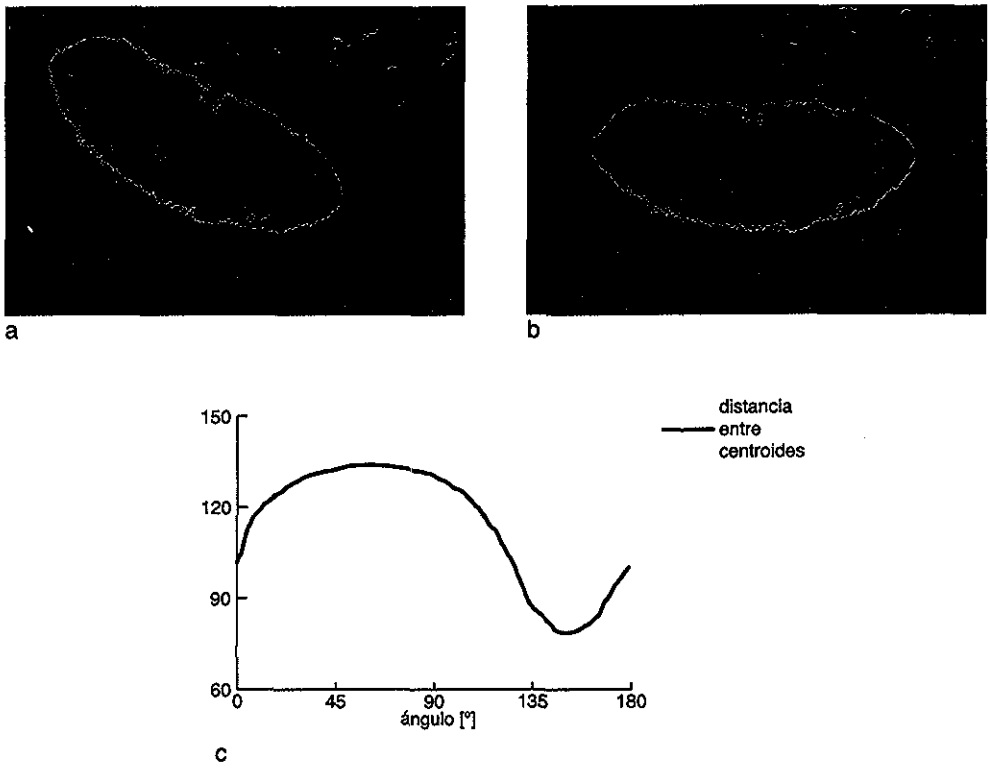


Figura 19: Alineamiento En la imagen a se observa una rebanada segmentada, la gráfica c representa la distancia entre centroides para distintos ángulos, y en la figura b se observa el resultado de haber rotado la imagen a 180^0 menos el ángulo que marca la distancia mínima entre centroides

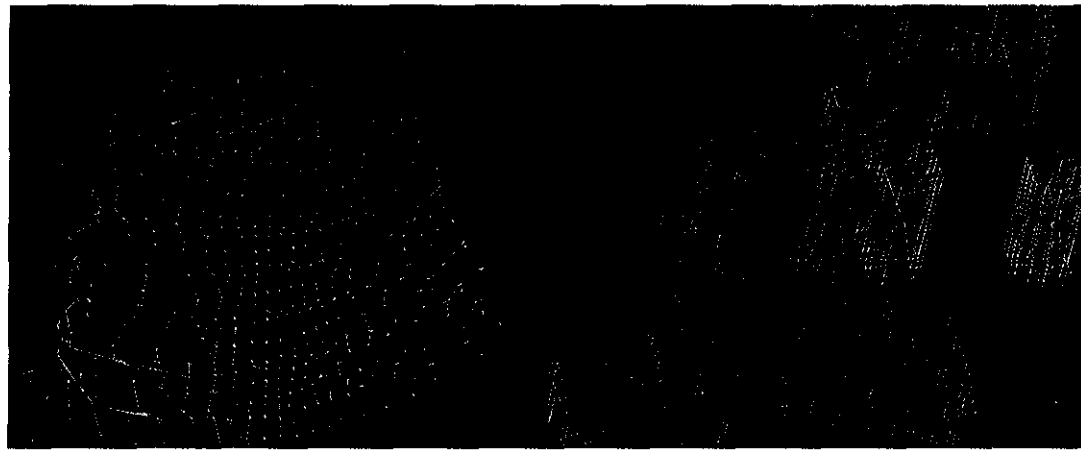


Figura 20: Mallas tridimensionales. En esta figura se puede observar la manera en que están estructuradas las mallas para la gónada (izquierda) y la inervación (derecha)

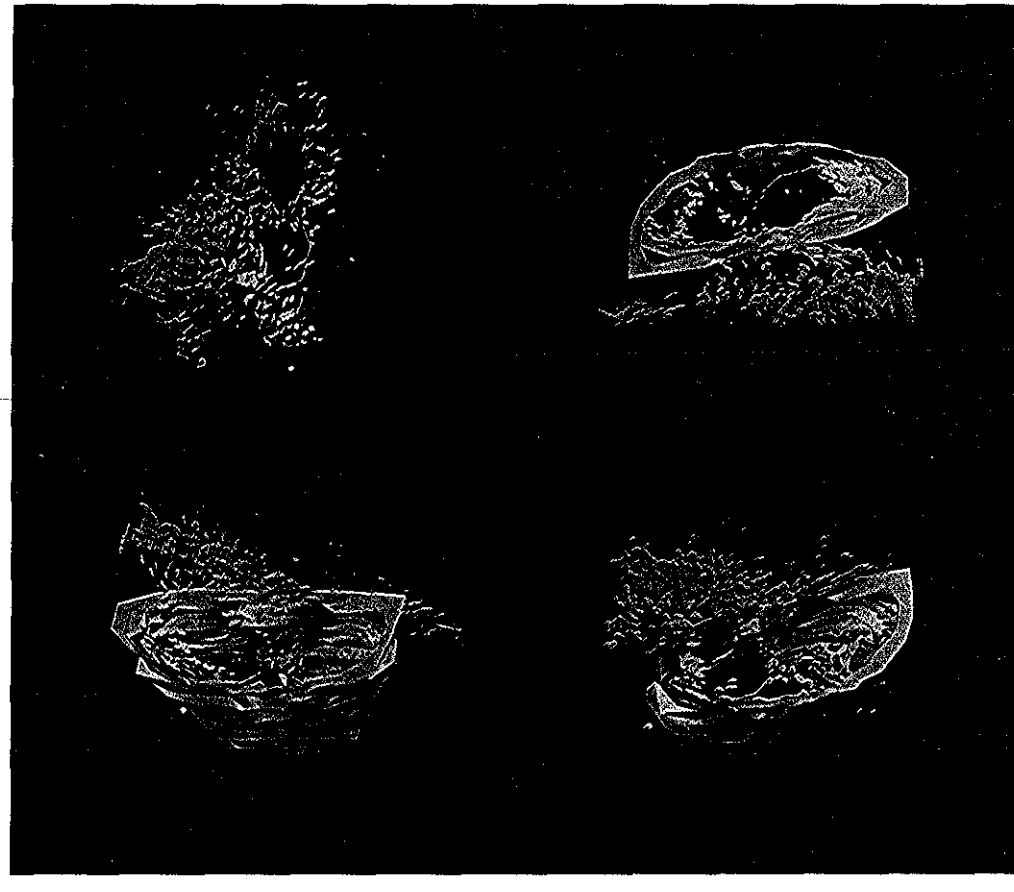


Figura 21: Cuatro vistas del rendering de la gónada. En el extremo superior izquierdo sólo se representó la inervación.

Parte V

Conclusiones

Los resultados de este trabajo deben interpretarse desde dos puntos de vista distintos; uno referente a los resultados de las distintas técnicas aplicadas en el análisis de imágenes, y el otro, referente al caso de estudio de aplicación en la biología.

Análisis de imágenes

Los resultados, en cuanto a las técnicas de análisis de imágenes utilizadas, nos permiten concluir que:

- La inervación colinérgica puede segmentarse utilizando únicamente un *thresholding* para seleccionar los pixeles más oscuros.
- La forma más sencilla de segmentar la gónada consiste en aplicar un filtro laplaciano de 5×5 pixeles, seguido de un filtro Sobel. Si bien esta combinación requiere en ocasiones de un retoque manual, encontramos que reduce al mínimo la intervención del usuario al mismo tiempo que proporciona una resolución anatómica adecuada.
- El mejor alineamiento fue obtenido mediante el método de los "dos centroides", que proporciona un alineamiento más preciso que el método de mínima entropía y en mucho menos tiempo del que requiere el alineamiento manual.
- La creación de mallas para representar únicamente la membrana de la gónada, junto con una representación más detallada de la inervación nos permite visualizar de manera correcta su distribución tridimensional. El poder manipular estos modelos en tiempo real le da una gran ventaja sobre los métodos basados 100% en voxeles y el *raytracing*, debido a que el poder observar rápidamente distintos ángulos de la misma escena nos da una mejor idea de la posición real de sus elementos.
- Los procesos de segmentación, alineamiento y generación de mallas dependen mucho de las características de la imagen y del objeto en estudio y, en el caso del *rendering*, del gusto personal de quien presenta la información. Por lo tanto resulta sumamente difícil (si no es que imposible) crear un método universal de reconstrucción tridimensional. Sin embargo, las técnicas utilizadas en este proyecto pueden modificarse y completarse con otras para realizar reconstrucciones tridimensionales de objetos distintos al de nuestro caso de estudio. Estas técnicas implementadas en la Unidad de Análisis de Imágenes han sido utilizadas en

distintos proyectos del Centro de Neurobiología (como los de los alumnos Juan Mena y Helga Giovannini).

Caso de estudio

Gracias a las técnicas de análisis de imagen desarrolladas durante este trabajo, pudimos visualizar en forma tridimensional la distribución de fibras colinérgicas en la gónada de un embrión de *L. olivacea*, en el estadio 26 de desarrollo, cultivado a temperaturas feminizantes. El modelo tridimensional generado nos permitió determinar que la inervación en este embrión se introduce en la gónada de manera lateral en la parte media, de ahí parten tractos que corren por la parte ventral y dorsal, y de los cuales salen fibras perpendiculares que se internan en la médula gonadal.

Este tipo de modelos tridimensionales nos podría ayudar, al utilizarlos en un mayor número de embriones en distintos estadios de desarrollo e incubados a distintas temperaturas, a visualizar el desarrollo de la inervación nerviosa de la gónada durante la diferenciación sexual dependiente de temperatura, con mayor precisión que las técnicas convencionales.

Parte VI

Apéndice

Funciones de lectura y escritura

```
unsigned char leebyte(IExtensionRecPtr p, IImageReference img,
 int x, int y, int z){
 unsigned char *val;
 CBSelectFrame(p, &img, z, false);
 val = (unsigned char *)(img.fFramePtr + 1 * (x + y
 * img.fWidth));
 return(*val);
 }

void escbyte(IExtensionRecPtr p, IImageReference img,
 int x, int y, int z, unsigned char valor){
 unsigned char *val;
 CBSelectFrame(p, &img, z, false);
 val = (unsigned char *)(img.fFramePtr + 1 * (x + y
 * img.fWidth));
 *val = valor;
 }
```

Traducción de máscaras de IPLab a imágenes tipo byte

```
IImageReference imgRef, imgMask;
short  rengbytes;
PixMap  **pM;
int  i, j;
unsigned char *uChar, *puntero;
unsigned char val, val1, val2;

CBGetFrontImageRef(p, &imgRef);
if(imgRef.fOverlay != NULL){
 CBGetEmptyImageRef(p, &imgMask);
 CBNewImage(p, &imgMask, IPLDImage, byteImage, imgRef fWidth,
 imgRef.fHeight, 1, viewAsData, "\pMascara", nil);

 Inf = (*(DialogParamRecPtr)(p->fParameters)).Inf;
 Sup = (*(DialogParamRecPtr)(p->fParameters)).Sup;

 pM = imgRef.fOverlay;
 rengBytes = (short)((**pM).rowBytes);
 rengBytes = rengBytes << 2;
 rengBytes = rengBytes >> 2;

 puntero = (unsigned char*)((**(imgRef.fOverlay)).baseAddr);
```

```
// Traducir el overlay en imagen de IPLab
for(j = 0; j < imgRef.fHeight; j++)
for(i = 0; i < imgRef.fWidth / 2; i++){
 val = *(puntero + j * rengBytes + i);
 val1 = val & 0x000F;
 val2 = val & 0x00F0;
 val2 = val2 >> 4;
 uChar = (unsigned char*)(imgMask.fDataPtr + j *
imgRef.fWidth + i * 2 + 0);
 *uChar = val2;
 uChar = (unsigned char*)(imgMask.fDataPtr + j *
imgRef.fWidth + i * 2 + 1);
 *uChar = val1;
 }
if((imgRef.fWidth % 2) == 1)
for(j = 0; j < imgRef.fHeight; j++){
 val = *(puntero + j * rengBytes + i);
 val2 = val & 0x00F0;
 val2 = val2 >> 4;
 uChar = (unsigned char*)(imgMask.fDataPtr + j *
imgRef.fWidth + imgRef.fWidth - 1);
 *uChar = val2;
}
}
```

Código completo del generador de *kernels* gaussianos

```
#include <Memory.h>
#include <OSUtils.h>
#include <Dialogs.h>
#include <TextUtils.h>
#include <Resources.h>
#include <math.h>
#include <TextUtils.h>
#include <stdlib.h>
#include "IPLabExtensions.h"

#define kDialogID 20040
#define kOk  1
#define kCancel  2
#define kOkBox  3
#define kSigma  5
#define PI 3.1415926535898
```

```
typedef struct{
 int Sigma;
 } DialogParamRec, *DialogParamRecPtr;

typedef struct{
 DialogParamRecPtr dlgParams;
 } IDialogInfo, *IDialogInfoPtr;

void gaussian (IExtensionRecPtr p);

void ShowMyDialog (IExtensionRecPtr p);
float GetFloatStringItem (DialogPtr dlg, short itemID);
void SetIntStringItem (DialogPtr dlg, short itemID, short num);
pascal void DrawMyDialogItems (DialogPtr dlg, short item);

pascal void main (IExtensionRecPtr p){
 switch (p->fAction){
 case kDoAction: gaussian(p);
    break;
 case kDialogAction: ShowMyDialog(p);
    break;
 case kScInquireAction: p->fErr = kNoErr;
    *(p->fParameters) = kDoesUseDialog;
    break;
 }
 }

void gaussian (IExtensionRecPtr p){
 IImageReference imgRef;
 int  i, j, N, ancho;
 float  f, *fp;
 float  gSigma = 1;

 p->fErr = kExtensionErr;

 CBGetEmptyImageRef(p, &imgRef);

 gSigma = (*(DialogParamRecPtr)(p->fParameters)) Sigma;

 if(gSigma < 0)
 gSigma = -gSigma;

 N = ceil(2 * gSigma + 1);
 ancho = 2 * N + 1;
 CBNewImage(p, &imgRef, IPLDImage, floatImage, ancho, ancho, 1,
 viewAsIext, "\pGaussian filter", nil);

 for(i = -N; i <= N; i++)
 for(j = -N; j <= N; j++){
```

```
    f = exp(-0.5 * ((i * i + j * j) / (gSigma * gSigma))) /
   (gSigma * gSigma * 2 * PI);
    fp = (float *)(imgRef.fDataPtr + ((j + N) * ancho +
   (i + N)) * sizeof(float));
    *fp = f;
    }
   CBUpdateImage(p, &imgRef);
   CBReleaseImageRef(p, &imgRef);

   p->fErr = kNoErr;
   }

  float GetFloatStringItem (DialogPtr dlg, short itemID){
   short dIyp;
   Handle dHnd;
   Rect dBox;
   Str255 str;
   float num;
   char *Cstr;

   Cstr = (char *)malloc(256);
   num = 0;
   GetDialogItem (dlg, itemID, &dIyp, &dHnd, &dBox);
   GetDialogItemIext (dHnd, str);
   Cstr = p2cstr(str);
   num = atof(Cstr);
   free(Cstr);
   return (num);
   }

  void SetIntStringItem (DialogPtr dlg, short itemID, short num){
   short dIyp;
   Handle dHnd;
   Rect dBox;
   Str255 str;

   NumIoString (num, str);
   GetDialogItem (dlg, itemID, &dIyp, &dHnd, &dBox);
   SetDialogItemIext (dHnd, str);
   return;
   }

  pascal void DrawMyDialogItems (DialogPtr dlg, short item){
   short dIyp;
   Handle dHnd;
   Rect dBox;

   GetDialogItem (dlg, item, &dIyp, &dHnd, &dBox);
```

```
 if (item == kOkBox){
 InsetRect (&dBox, -4, -4);
 PenSize (3,3);
 FrameRoundRect (&dBox, 16, 16);
 PenSize (1,1);
 }
 }

void ShowMyDialog (IExtensionRecPtr p){
 short  dIyp;
 Handle  dHnd;
 Rect  dBox;
 short  item;
 DialogPtr dlg;
 DialogRecord dlgRec;
 DialogParamRec values;
 MenuHandle opMenu;
 IDialogInfo dlgInfo;
 long  menuResult;
 UserItemUPP itemUPP;

 // capturar la caja de dialogo
 dlg = GetNewDialog (kDialogID, (Ptr)&dlgRec, (WindowPtr)-1);

 itemUPP = NewUserItemProc (&DrawMyDialogItems);

 GetDialogItem (dlg, kOkBox, &dIyp, &dHnd, &dBox);
 SetDialogItem (dlg, kOkBox, dIyp, (Handle)itemUPP, &dBox);

 // llenar los campos de edicicion con nuestros parametros
 values = *(DialogParamRecPtr)(p->fParameters);
 values.Sigma = 1;
 SetIntStringItem (dlg, kSigma, values.Sigma);

 // llenar nuestro archivo de informacion y pegarselo a la CD
 dlgInfo.dlgParams = &values;
 ((WindowPeek)dlg)->refCon = (long)&dlgInfo;

 // seleccionar el primer campo
 SelectDialogItemIext (dlg, kSigma, 0, 32767);

 // mostrar la ventana
 ShowWindow (dlg);
 SetPort (dlg);

 do{
 ModalDialog (nil, &item);
 } while ((item != kOk) && (item != kCancel));
```

```
if (item == kOk){
values.Sigma = GetFloatStringItem (dlg, kSigma);
*(DialogParamRecPtr)(p->fParameters) = values;r
p->fParameterSize = sizeof(values);
p->fErr = kNoErr;
}
else{
p->fErr = kUserCancelled;
}

// cerrar la CD y liberar memoria
CloseDialog (dlg);
DisposeHandle(dlgRec items);

// eliminar el UPP
DisposeRoutineDescriptor(itemUPP);
}
```

Generador de filtros laplacianos

El módulo para generar filtros laplacianos es completamente análogo al de gaussianos, sólo hay que cambiar la función `laplacian` por la función `gaussian`

```
void laplacian (IExtensionRecPtr p){
 IImageReference imgRef;
 int  i, j, N, ancho;
 float  f, *fp;
 float  gSigma = 1;

 p->fErr = kExtensionErr;

 CBGetEmptyImageRef(p, &imgRef);

 gSigma = (*(DialogParamRecPtr)(p->fParameters)).Sigma;

 if(gSigma < 0)
 gSigma = -gSigma;

 N = ceil(3 * gSigma + 1);
 ancho = 2 * N + 1;
 CBNewImage(p, &imgRef, IPLDImage, floatImage, ancho, ancho,
 1, viewAsIext, "\pGaussian filter", nil);
```

```
 for(i = -N; i <= N; i++)
 for(j = -N; j <= N; j++){
  f = (1 - 1.0* (i * i + j * j) / (2* gSigma * gSigma))
 * exp(- 1.0 * (i * i + j * j) / (2* gSigma * gSigma))
 / (2 * PI * gSigma * gSigma);
  fp = (float *)(imgRef fDataPtr + ((j + N) * ancho +
 (i + N)) * sizeof(float));
  *fp = f;
  }
 CBUpdateImage(p, &imgRef);
 CBReleaseImageRef(p, &imgRef);

 p->fErr = kNoErr;
 }
```

Filtro Sobel

```
void Sobel (IExtensionRecPtr p){
 IImageReference imgRef, imgRes;
 unsigned int i, j, ancho, alto;
 unsigned char *uCharPtr;
 float *floatPtr, floatVal1, floatVal2, otrofloat;

 p->fErr = kExtensionErr;

 if((CBGetFrontImageRef(p, &imgRef) == kNoErr) &&
  (CBGetEmptyImageRef(p, &imgRes) == kNoErr) &&
  (CBNewImage(p, &imgRes, IPIDImage, floatImage, imgRef fWidth,
imgRef fHeight,
  2, viewAsData, "\pSobel", nil) == kNoErr)){

  ancho = imgRef fWidth;
  alto = imgRef fHeight;

  for(i = 1; i < ancho - 1; i++)
   for(j = 1; j < alto -1; j++){
    floatVal1 = floatVal2 = 0;

    uCharPtr = (unsigned char *)(imgRef fDataPtr + (j - 1) * ancho +
i - 1);
    floatVal1 += *uCharPtr;
    floatVal2 += *uCharPtr;
    uCharPtr = (unsigned char *)(imgRef fDataPtr + (j - 1) * ancho +
i);
```

```
    floatVal2 += *uCharPtr;
    uCharPtr = (unsigned char *)(imgRef.fDataPtr + (j - 1) * ancho +
i + 1);
    floatVal1 -= *uCharPtr;
    floatVal2 += *uCharPtr;
    uCharPtr = (unsigned char *)(imgRef.fDataPtr + j * ancho + i - 1);
    floatVal1 += *uCharPtr;

    uCharPtr = (unsigned char *)(imgRef.fDataPtr + j * ancho + i + 1);
    floatVal1 -= *uCharPtr;

    uCharPtr = (unsigned char *)(imgRef fDataPtr + (j + 1) * ancho +
i - 1);
    floatVal1 += *uCharPtr;
    floatVal2 -= *uCharPtr;
    uCharPtr = (unsigned char *)(imgRef.fDataPtr + (j + 1) * ancho +
i);

    floatVal2 -= *uCharPtr;
    uCharPtr = (unsigned char *)(imgRef.fDataPtr + (j + 1) * ancho +
i + 1);
    floatVal1 -= *uCharPtr;
    floatVal2 -= *uCharPtr;

    // Magnitud
    floatPtr = (float *)(imgRes.fDataPtr + sizeof(float) * (j * ancho
+ i));
    otrofloat = floatVal1 * floatVal1 + floatVal2 * floatVal2;
    *floatPtr = (float)(sqrt(otrofloat));
    // Direccion
    floatPtr = (float *)(imgRes fDataPtr + sizeof(float) * (ancho *
alto +
     j * ancho + i));
    *floatPtr = (float)(atan2(floatVal2, floatVal1));

    }
  imgRes fDataChanged = 1;
  imgRes.fReNormalize = kForceReNormalize;
  }
 CBUpdateImage(p, &imgRef);
 CBUpdateImage(p, &imgRes);
 CBReleaseImageRef(p, &imgRef);
 CBReleaseImageRef(p, &imgRes);
 p->fErr = kNoErr;
}
```

Calculo de las distancias entre centroides

```
void twocent (IExtensionRecPtr p){
 unsigned int i, j, k;
 unsigned int cuenta = 0, cuentaA, cuentaB;
 IImageReference imgRef, imgRes;
 Point   punto;
 float   centx, centy, centxA, centyA, centxB, centyB;
 float   sumax = 0, sumay = 0, sumaxA, sumayA, sumaxB, sumayB;
 double   theta;
 float   *fPtr;

 // asignar referencias de imagen
 if((CBGetFrontImageRef(p, &imgRef) == kNoErr)
  && (CBGetEmptyImageRef(p, &imgRes) == kNoErr)){

  // verificar la existencia de una ROI en la ventana de imagen
  if ((EmptyRgn(imgRef.fROIRegion)) ||
   (imgRef.fWindowKind != kImageKind))
   SysBeep(1);
  else {
   // encontrar el centroide
   for(i = imgRef.fROILeft; i < imgRef.fROIRight; i++)
    for(j = imgRef.fROITop; j < imgRef.fROIBottom; j++){
     SetPt(&punto, i, j);
     // verificar si el p unto esta dentro de la ROI
     if(PtInRgn(punto, imgRef.fROIRegion)){
      sumax += i;
      sumay += j;
      cuenta ++;
      }
     }
   centx = sumax / cuenta;
   centy = sumay / cuenta;

   // crear la tabla de resultados
   CBNewImage(p, &imgRes, IEXIFormat, floatImage, 6, 180, 1,
viewAsText,
    "\pCentroids.txt", nil);
   CBSetImageLabel(p, &imgRes, 0, true, "\pangle");
   CBSetImageLabel(p, &imgRes, 1, true, "\pcentxA");
   CBSetImageLabel(p, &imgRes, 2, true, "\pcentyA");
   CBSetImageLabel(p, &imgRes, 3, true, "\pcentxB");
   CBSetImageLabel(p, &imgRes, 4, true, "\pcentyB");
   CBSetImageLabel(p, &imgRes, 5, true, "\pdistance");


   // para cada uno de los angulos de division calcular el centroide
```

```
de cada seccion
   for(k = 0; k < 180; k++){
    cuentaA = 0;
    cuentaB = 0;
    sumaxA = 0;
    sumayA = 0;
    sumaxB = 0;
    sumayB = 0;
    // para cada uno de los puntos dentro de la ROI
    for(i = imgRef.fROILeft; i < imgRef.fROIRight; i++)
     for(j = imgRef.fROITop; j < imgRef.fROIBottom; j++){
      // verifivar que el punto este dentro de la ROI
      SetPt(&punto, i, j);
      if(PtInRgn(punto, imgRef.fROIRegion)){
       theta = 180 * atan2(centy - j, i - centx) / 3.14159265358979;
       // convertir a angulo positivo
       if(theta < 0)
        theta = 360 + theta;
       // asignar un punto a la region A o B
       if(theta >= k && theta <= k + 180){
        sumaxA += i;
        sumayA += j;
        cuentaA ++;
        }
       if(theta <= k || theta >= k + 180){
        sumaxB += i;
        sumayB += j;
        cuentaB ++;
        }
       }
      }
    centxA = sumaxA / cuentaA;
    centyA = sumayA / cuentaA;
    centxB = sumaxB / cuentaB;
    centyB = sumayB / cuentaB;

    // escribir el angulo
    fPtr = (float*)(imgRes.fDataPtr + sizeof(float) * (6 * k + 0));
    *fPtr = k;
    // escribir  los centroides
    fPtr = (float*)(imgRes.fDataPtr + sizeof(float) * (6 * k + 1));
    *fPtr = centxA;
    fPtr = (float*)(imgRes.fDataPtr + sizeof(float) * (6 * k + 2));
    *fPtr = centyA;
    fPtr = (float*)(imgRes.fDataPtr + sizeof(float) * (6 * k + 3));
    *fPtr = centxB;
    fPtr = (float*)(imgRes.fDataPtr + sizeof(float) * (6 * k + 4));
    *fPtr = centyB;
    // escribir la distancia
```

```
    fPtr = (float*)(imgRes.fDataPtr + sizeof(float) * (6 * k + 5));
    *fPtr = sqrt((centxA - centxB) * (centxA - centxB) +
     (centyA - centyB) * (centyA - centyB));
    }
   CBUpdateImage(p, &imgRes);
   }
  }
 CBReleaseImageRef(p, &imgRef);
 CBReleaseImageRef(p, &imgRes);
 }
```

Escritura de la malla de la inervación en formato de *Open Inventor*

```
 if(((pf = fopen("nerv.iv", "w+")) == NULL) || (CBGetFrontImageRef(p,
&imgRef)
  != kNoErr))
  SysBeep(1);
 else{
  // Escribir encabezado
  fprintf(pf, "#Inventor V2 0 ascii\n\n");

  ancho = imgRef.fWidth;
  alto = imgRef.fHeight;
  frames = imgRef.fNumFrames;

  for(k = 0; k < frames; k++)
   for(j = 0; j < alto; j++){
    // calcular coordenada y corregida al centroide
    y = j - cent[k][1];

    for(i = 0; i < ancho; i++){
     uCharPtr = (unsigned char *)(imgRef.fDataPtr +
      k * ancho * alto + j * ancho + i);
     uChar = *uCharPtr;
     if(uChar == 11){
      // calcular coordenada x inicial, corregida al centroide
      xi = i - cent[k][0];

      for(; i < ancho; i++){
       uCharPtr = (unsigned char *)(imgRef.fDataPtr +
        k * ancho * alto + j * ancho + i);
       uChar = *uCharPtr;
       if(uChar != 11) break;
       }
      // calcular coordenada x final, corregida al centroide
```

```
  xf = i - cent[k][0];

  m = (xi + xf) / 2;

  // Escribir cuboide
  fprintf(pf, "Separator {\n");
  fprintf(pf, "\tMaterial { diffuseColor\t1\t%f\t0 }\n",
   (y - kYMin) / (kYMax - kYMin));
  fprintf(pf, "\tTranslation { translation\t%f\t%f\t%f }\n",
   m, y, (float)(k * 15));//
  fprintf(pf, "\tCube {\n");
  fprintf(pf, "\t\twidth\t%f\n", xf - xi);
  fprintf(pf, "\t\theight\t1\n");
  fprintf(pf, "\t\tdepth\t15\n"); // altura del cuboide
  fprintf(pf, "\t\t}\n");
  fprintf(pf, "\t}\n\n");
  }
 }
}
```

Índice de Figuras

Bibliografía

[1] J. C. Russ, *The Image Processing Handbook* Boca Raton, Florida: CRC Press, 2nd ed , 1994.

[2] M Sonka and J. M. Fitzpatrick, eds , *Handbook of Medical Imaging, vol 2: Medical Image processing and Analysis* Bellingham, Washington: SPIE Press, 2000.

[3] I Sobel, "Camera models and machine perception," Tech Rep AIM-21, Stanford Artificial Intelligence Lab, Palo Alto, California, 1970.

[4] M Artin, *Algebra* Upper Saddle River, New Jersey: Prentice Hall, 1st ed., 1991.

[5] K S Fu and J. K. Mui, "A survey in image segmentation," *Pattern Recognition*, vol. 13, pp 3–16, 1981

[6] J. Rangarajan, "A robust point-matching algorithm for autoradigraph alignment.," *Medical Image Analysis*, vol. 1, no 4, pp 379–398, 1997.

[7] C R. Maurer Jr , J M Fitzpatrick, M Y. Wang, R. L. Galloway, R. J Maciunas, and G S. Allen, "Registration of head volume images using implantable fiducial markers," *IEEE Transactions on Medical Imaging*, vol. 16, no 4, pp. 447–462, 1997.

[8] R P. Woods, S R. Cherry, and J. C. Mazziotta, "Rapid automated algorithm for aligning and reslicing pet images," *Journal of Computed Assisted Tomography*, vol. 16, pp. 620–633, 1992

[9] Y Kim and S C. Horii, eds , *Handbook of Medical Imaging, vol 3: Display and PACS* Bellingham, Washington: SPIE Press, 2000

[10] S Schreiner, C B. Paschal, and R. L. Galloway, "Comparison of proyection algorithms used for the construction of maximum intensity projection images," *Journal of Computer Assisted Tomgraphy*, vol. 20, no. 1, pp 56–67, 1996

[11] G. P. Carnielli, A. X Falcão, and J Udupa, "Fast digital perspective shell rendering," in *XII Brazilian Symposium on Computer Graphics and Image Processing*, pp. 105–111, 1999

[12] H Meinzer, Y Meetz, and D. Scheppelmann, "Raytracing of medical 3D tomographies," in *Proceedings of the Annual International Conference of the IEEE. Engineering in Medicine and Biology Society*, vol 13, pp. 41–42, 1991.

[13] C Barillot, "Surface and volume rendering techniques to display 3-D data," *IEEE Engineering in Medicine and Biology Magazine*, vol 12, no 1, pp 111–119, 1993

[14] S. L. Wood, "Visualization and modeling of 3-D structures (medical imaging)," *IEEE Engineering in Medicine and Biology Magazine*, vol 11, no 2, pp. 72–79, 1992

[15] A. M Coriat, E. Valleley, M W Ferguson, and P T Sharpe, "Chromosomal and temperature-dependent sex determination: the search for a conserved mechanism ," *The Journal of Experimental Zoology*, vol. 270, no 1, pp 112–116, 1994.

[16] J. R Spotila, L D Spotila, and N F. Kaufer, "Molecular mechanisms of tsd in reptiles: a search for the magic bullet," *The Journal of Experimental Zoology*, vol. 270, no 1, pp 117–127, 1994

[17] D Crews, J M. Bergeron, J. J Bull, D Flores, A. Tousignant, J. K. Skipper, and T. Wibbels, "Temperature-dependent sex determination in reptiles: proximate mechanisms, ultimate outcomes, and practical applications," *Developmental Genetics*, vol 15, no 3, pp. 297–312, 1994

[18] F J Janzen and G L. Paukstis, "Environmental sex determination in reptiles: ecology, evolution, and experimental design ," *Quarterly Review of Biology*, vol 66, no. 2, pp 149–179, 1991.

[19] H Merchant-Larios, I Villalpando-Fierro, and B Centeno-Urruiza, "Gonadal morphogenesis under controlled temperature in the sea turtle *Lepidochelys olivacea*," *Herpetological Monographs*, vol. 3, pp 43–61, 1989

[20] R White and P Thomas, "Adrenal-kidney and gonadal steroidogenesis during sexual differentiation of a reptile with temperature-dependent sex determination.," *General and Comparative Endocrinology*, vol 88, no. 1, pp 10–19, 1992

[21] I. Villalpando and H Merchant-Larios, "Determination of the sensitive stages for gonadal sex-reversal in *Xenopus laevis* tadpoles," *International Journal of Developmental Biology*, vol 32, no. 2, pp. 281–285, 1990.

[22] P. Coomber, D. Crews, and F. Gonzalez-Lima, "Independent effects of incubation temperature and gonadal sex on the volume and metabolic capacity of brain nuclei in the leopard gecko (*Eublepharis macularius*), a lizard with temperature-dependent sex determination," *Journal of Comparative Neurology*, vol. 380, no. 3, pp. 409–21, 1997.

[23] Signal Analytics Corporation, *IPLab Spectrum User's Guide*, 1997

[24] R. Adams and B. Leanne, "Seeded region growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641–647, 1994.