UNIVERSIDAD AUSTRAL DE CHILE
FACULTAD DE CIENCIAS DE LA INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN INFORMÁTICA

RECONSTRUCCIÓN, VISUALIZACIÓN Y PARAMETRIZACIÓN TRIDIMENSIONAL DE ESTRUCTURAS BIOLÓGICAS A TRAVÉS DE IMÁGENES CONFOCALES DE FLUORESCENCIA

TESIS PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

PATROCINANTE:
STEFFEN HÄRTEL GRÜNDLER
DIPLOMADO FISICA
DR. CIENCIAS NATURALES

CO-PATROCINANTE:
ERICK ARAYA ARAYA
INGENIERO DE EJECUCIÓN ELECTRÓNICO
MAGISTER EN INGENIERÍA ELECTRÓNICA

INFORMANTE:
GLADYS MANSILLA G.
INGENIERO CIVIL MATEMATICO
MAGISTER EN ESTADISTICA

ETIENE V. VERDUGO RUIZ

VALDIVIA – CHILE
2005

Valdivia, 14.03.2005

A: Directora Escuela Ingeniería Civil en Informática

Motivo: Calificación Proyecto de Título

Titulo: Reconstrucción, visualización y parametrización tridimensional de estructuras biológicas a través de imágenes confocales de fluorescencia

Alumno: Etiene V. Verdugo Ruiz

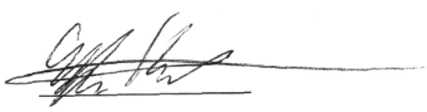
En su proyecto, el Sr. Etiene Verdugo realizó todos los objetivos planteados en el anteproyecto de su tesis con mucha disciplina y dedicación. El desarrolló e implementó técnicas interactivas para la reconstrucción y visualización tridimensional de estructuras biológicas a raíz de objetos gráficos utilizando modelos de superficies, de textura, de volúmenes, cortes en volúmenes, y sus respectivas proyecciones. Parámetros de los modelos gráficos se calibraron utilizando estructuras microscópicas de tamaños conocidos y aplicando técnicas de deconvolución. Se desarrolló un ambiente interactivo para el acceso rápido a diversos parámetros intrínsecos de los objetos visualizados.

Como patrocinante quiero destacar la gran disposición, el empeño y la perseverancia del Sr. Verdugo en realizar su trabajo dentro de un entorno interdisciplinario e internacional dedicado a la ciencia. Lo que separa este trabajo destacable de la excelencia es la falta de maduración final de algunos temas particulares que se podrían haber concretado con más precisión y sometido a una discusión más crítica en el entorno global de la materia.

Más allá de su trabajo en programación el Sr. Verdugo mostró gran interés en los diversos campos de investigación y supo solucionar muchos de los problemas cotidianos en computación que sufrieron sus colegas en el Centro de Estudios Científicos. El gran esfuerzo invertido generó resultados valiosos para diferentes proyectos dentro del CECS.

Nota: 6.9 (seis punto nueve)

Atentamente,



Dr. Steffen Härtel

VALDIVIA, 28 de Marzo de 2005

DE : ERICK ARAYA ARAYA

A : DIRECTORA ESCUELA INGENIERÍA CIVIL EN INFORMÁTICA

MOTIVO:

INFORME TRABAJO DE TITULACIÓN

Nombre Trabajo de Titulación: RECONSTRUCCION, VISUALIZACION Y PARAMETRIZACION TRIDIMENSIONAL DE ESTRUCTURAS BIOLOGICAS A TRAVÉS DE IMÁGENES CONFOCALES DE FLUORESCENCIA

Nombre del Alumno : .ETIENE VERDUGO RUIZ

.....
.....

Nota : .....6,5..... ..SEIS COMA CINCO.....
(en números) (en letras)

FUNDAMENTO DE LA NOTA: El gran mérito de este trabajo es la contribución que un estudiante de Ingeniería Civil Informática puede aportar a un trabajo y a un grupo científico de alta calidad en la solución de problemas relacionados. Si bien los algoritmos son simples, han sido bien desarrollados por el alumno e implementados a plena satisfacción de los investigadores. Las conclusiones, sin embargo, deberían haber reflejado con más detalle la importancia y los aportes personales del alumno.

Atentamente,

ERICK ARAYA A.

VALDIVIA, 22 DE MARZO DEL 2005

DE: GLADYS MANSILLA GOMEZ
A : MIGUELINA VEGA. DIRECTORA ESCUELA ING. CIVIL EN INFORMATICA

MOTIVO INFORME TRABAJO DE TITULACION

Nombre Trabajo de Titulación: "RECONSTRUCCIÓN, VISUALIZACIÓN Y PARAMETRIZACIÓN TRIDIMENSIONAL DE ESTRUCTURAS BIOLÓGICAS A TRAVES DE IMÁGENES CONFOCALES DE FLUORESCENCIA"

Nombre del alumno: ETIENE VALENTINO VERDUGO RUIZ.

Nota: 7.0 (en números) siete (en palabras)

Fundamento de la nota:

- Este trabajo de tesis presenta un trabajo interdisciplinario de gran valor, en que el alumno logro con éxito incorporar una gran cantidad de nuevos conocimientos a su formación y desarrollar una aplicación util

- Se nota en el alumno el interés por realizar un trabajo que contribuya eficazmente a la investigación en la que este tema está inserto.
-
- En este trabajo es posible destacar la claridad con que el alumno utiliza la nueva terminología relativa a sistemas biológicos y de visualización adquirida durante su trabajo.

- En la realización de este trabajo de titulación se alcanzan plenamente los objetivos planteados al inicio.

- La presentación y redacción del informe están bien elaboradas, abarcando tópicos que inciden directamente en esta tesis y expresado en un lenguaje formal apropiado.

- Es destacable el hecho que el alumno haya participado en la confección y presentación de artículos científicos.

GLADYS MANSILLA GÓMEZ
DOCENTE INSTITUTO DE INFORMATICA

*A*gradecimientos

Quiero agradecer sinceramente a todas las personas que han contribuido desde el periodo universitario hasta este punto, donde culmina una etapa de gran esmero y satisfacción tanto espiritual como intelectual, sin ustedes no habría sido posible.

Quiero agradecer a todos mis maestros, y en especial a mi patrocinante Steffen Härtel, por su infinita paciencia, su dedicación, por toda su experiencia y conocimientos que ha sabido transmitir al desarrollo del proyecto y a una formación más profesional.

Quiero agradecer a la Sra. Gladys, la tía Emerita y en especial a la Sra. Lula que de alguna forma han logrado suplir el afecto y cariño de madre que todos necesitamos.

Quiero agradecer a todas las amistades que estuvieron y han estado presentes, en especial a Carola Bahamondez, Walkiria Gallegos, Jorge Jara, Carolina Montenegro y Dahianna Verdugo por su compañía, confianza y ayuda.

Quiero agradecer a mis padres y hermanos por el apoyo incondicional brindado en todos sus matices, que han sabido entregar a este periodo de vida.

Quiero agradecer a Dios por todo lo que ha significado y por hacerme comprender, que por muchas razones, viajamos y coincidimos.

Este proyecto se desarrolló en el Laboratorio de Biofísica y Fisiología Molecular, perteneciente al Centro de Estudios Científicos (CECS), y fue financiada por los proyectos: *FONDECYT* y *HOWARD HUGHES*, pertenecientes al Dr. Francisco Sepulveda.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

RESUMEN

En la última década, el procesamiento de imágenes ha llegado a ser una herramienta imponderable para diversas áreas de investigación. Gracias a los avances tecnológicos de los microscopios, capacidades computacionales y sofisticadas rutinas de procesamiento de imágenes, investigadores en el área de la biología celular empezaron a visualizar y analizar la morfología de diferentes estructuras celulares en tres dimensiones. Un método multidimensional es la microscopía confocal de fluorescencia *(Laser Scanning Microscopy*, LSM). LSM complementa otras técnicas como la microscopía óptica convencional o la microscopía electrónica por su capacidad de monitorear la intensidad (I) de fotones dentro de una muestra en tres dimensiones I(x,y,z). Aplicando técnicas de fluorescencia, las intensidades se pueden medir en diferentes bandas o canales (c) del espectro visible, monitoreando el comportamiento celular en un determinado tiempo (t) en múltiples colores.

El presente trabajo implementó algoritmos para reconstruir, visualizar y parametrizar estructuras biológicas, convirtiendo datos I(x,y,z,c,t) en objetos gráficos en el entorno del lenguaje computacional científico IDL® (*Interactive Data Language*). Los objetos gráficos fueron diseñados con gran flexibilidad para la selección de diferentes escalas de color y diferentes grados de opacidad para brindar la mejor resolución visual en el proceso del modelamiento tridimensional. Se realizaron calibraciones para optimizar la frecuencia del muestreo en xyz con el propósito de establecer condiciones ideales para el análisis de la morfología y del volumen celular en diferentes condiciones experimentales.

Las primeras aplicaciones de las técnicas desarrolladas permitieron visualizaciones de diferentes estructuras celulares como filopodias, perturbaciones de la membrana o *tunneling nanotubes* en el margen de la resolución teórica de la microscopía confocal. Además, la regulación del volumen celular fue monitoreada por primera vez a nivel de células únicas durante la muerte inducida (*apoptosis* y *necrosis*). Diversos resultados de este trabajo fueron presentados en congresos nacionales e internacionales.

SUMMARY

In the last decade, the image processing has becomed to be an imponderable tool for diverse areas of investigation. Thanks to the technological advances of the microscopes, computational capacities, and sophisticated routines of image processing, investigators in the area of cellular biology began to visualize and to analyze the morphology of different cellular structures in three dimensions. A multidimensional method is the confocal fluorescence microscopy (*Laser Scanning Microscopy*, LSM). LSM complements other techniques like the conventional optical microscopy or the electronic microscopy for their capacity to monitor the intensity (i) of photons within a sample in three dimensions I(x,y,z). By applying the fluorescence technique, intensities can be measured in different bands or channels (c) from the visible spectrum, monitoring the cellular behavior in a certain time (t) in multiple colors. This work implemented algorithms to reconstruct, visualize and parameter biological structures, converting data I(x,y,z,c,t) into graphical objects in the environment of the scientific computer language IDL® (*Interactive Data Language*). The graphical objects were designed with great flexibility for the selection of different color scales and opacity degrees to offer the best visual resolution in the process of the three-dimensional modeling. Calibrations were made to optimize the frequency of the sampling in xyz in order to establish ideal conditions for the analysis of the morphology and the cellular volume in different experimental conditions.

The first applications of the developed techniques allowed visualizations of different cellular structures like filopodias, disturbances of the membrane or tunneling nanotubes in the margin of the theoretical resolution of confocal microscopy. In addition, the regulation of the cellular volume was monitored for the first time a level of unique cells during the induced death (*apoptosis* and *necrosis*). Diverse results of this work had been presented in national and international congresses.

Nomenclatura

α	: Alpha
c	: Canal
CECS	: Centro de Estudios Científicos
DFD	: Diagrama de Flujo de Datos
CMY	: Cyan, Magenta, Yellow
HLS	: Hue, Lightness, Saturation
HSV	: Hue, Saturation, Value
I	: Intensidad
IDL	: Interactive Data Language
LSM	: Laser Scanning Microscope
LUT	: Look Up Table
mM	: Milimolar
NASA	: Nacional Aeronautics and Space Administration
nm	: Nanómetro
pl	: Picolitro
PSF	: Point Spread Function
RGB	: Red, Green, Blue
ROI	: Region of Interest
RSI	: Research Systems, inc.
t	: Tiempo
µm	: Micrómetro

Capítulo 1: Introducción

1.1 Información General del Problema

En la última década, el procesamiento de imágenes ha pasado ser una herramienta imponderable para diversas áreas de investigación. Gracias a los avances tecnológicos de los microscopios, capacidad de las computadoras, y sofisticadas rutinas de procesamiento de imágenes, investigadores en el área de la biología celular, visualizan y analizan tridimensionalmente la morfología de diferentes estructuras biológicas, permitiendo la visualización desde diferentes cortes y perspectivas (rotación, traslación y escalamiento). Uno de los métodos más usado para realizar estudios sofisticados de diferentes estructuras biológicas o el comportamiento de células en cultivo bajo condiciones controladas es la microscopía confocal de fluorescencia (*Laser Scanning Microscope* (LSM)). LSM facilita la determinación de características celulares que no se logran obtener con otros métodos (microscopía convencional y microscopía electrónica).

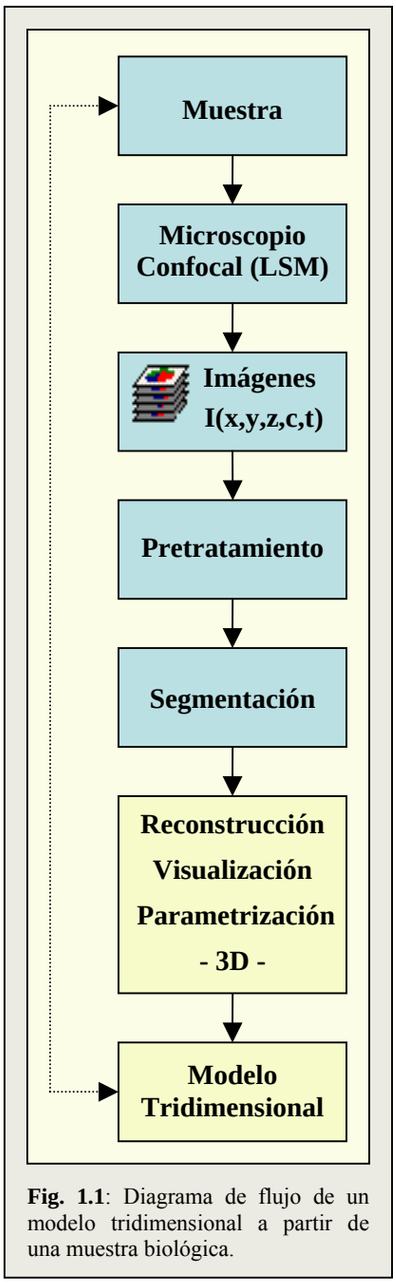


Fig. 1.1: Diagrama de flujo de un modelo tridimensional a partir de una muestra biológica.

Una característica importante en este aspecto es el volumen celular. Recientemente, cambios tempranos en el volumen celular han sido conectados directamente con el control de la sobrevivencia y de la muerte celular [Bar01]. La regulación del volumen se discute como un parámetro esencial para mantener el equilibrio entre la sobrevivencia de células y dos formas de muerte celular: la *apoptosis* o la *necrosis*.

La técnica del LSM permite adquirir datos tridimensionales de las muestras biológicas en múltiples canales en el tiempo (Fig. 1.1 y 1.2). Los datos originales tienen el formato:

I(x, y, z, c, t) , I = intensidad ∈ [0, 255] (8 bits) o ∈ [0, 4095] (12 bits)

, c_i = canal de fluorescencia, i = [1, 2,…, n]

, t = tiempo

, x, y, resolución en el plano xy ∈ [256, 512, 1024, 2048]

, z_i, cantidad de cortes en el eje z, resolución en altura z, i = [1, 2,…, n].

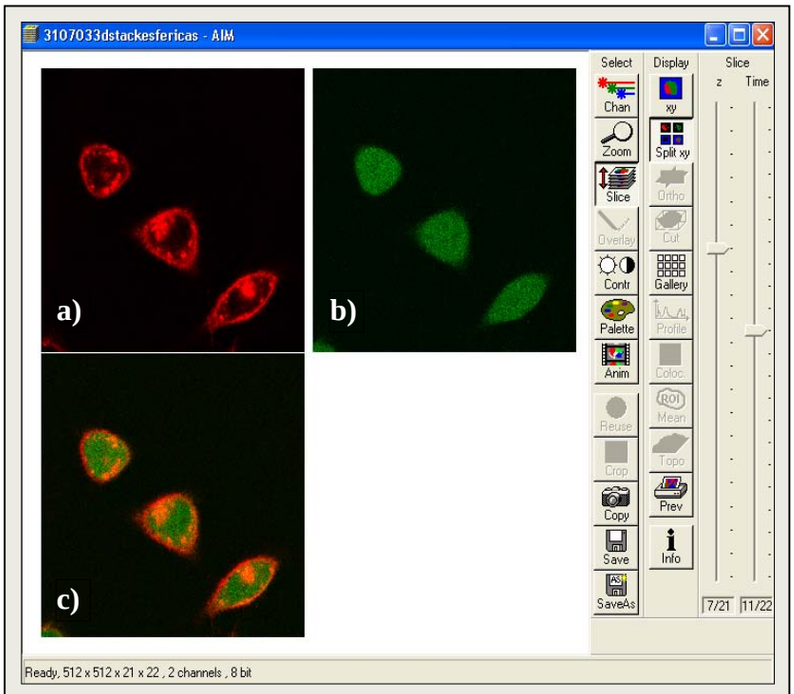


Fig. 1.2: Imagen confocal de fluorescencia representativa de tres células en dos canales de fluorescencia I(x,y,z,c,t) = I(512,512,21,2,22) ∈ [0, 255].
[a] Células cuyas membranas están marcadas con el fluoróforo $DiIC_{18}$ en el canal rojo.
[b] Células marcadas con el fluoróforo calceina en el canal verde.
[c] Superposición de los canales rojo y verde para la visualización de ambas características.

Cada canal (c_i) refleja la intensidad de fluorescencia (número de fotones) en un determinado rango de frecuencia de fotones emitidos. La fluorescencia es la propiedad de algunas moléculas de absorber ondas electromagnéticas (fotones) a una determinada longitud de onda y emitir fotones a una longitud de onda más grande.

La Fig. 1.2a/b muestra imágenes confocales tomadas es distintas longitudes de onda, representando dos canales de fluorescencia. La figura 1.2c muestra la superposición de los 2 canales.

Para realizar reconstrucciones, visualizaciones y parametrizaciones tridimensionales, es necesario aplicar técnicas de **pretratamientos** y **segmentaciones** confiables a las series de imágenes (Fig. 1.1):

- Para la microscopía confocal, el paso más importante del **pretratamiento** es la deconvolución de las imágenes con sus respectivas *point spread functions* (PSF) [Kem99]. Sin deconvolución, imágenes confocales sufren de un alto contenido de ruido (*photon noise*) y distorsiones en las dimensiones x, y, z, debido a la difracción de fotones dentro del sistema óptico del microscopio. Un cálculo dentro de la teoría óptica de difracción entrega la PSF [Kem99]. Esta función forma la base para el proceso iterativo de deconvolución realizado con el programa Huygens Professional[1] (Cap. 2.2) [URL 1].

- Hasta el momento no existe una definición unica para **segmentación**. En este trabajo se define la segmentación como el proceso de separar objetos o regiones de interés (*regions of interest,* ROI) dentro de imágenes [Gon02]. Una ROI es un conjunto de píxeles continuos, que se dispersan en distintas direcciones y presentan uniformidad entre sí. La segmentación consiste en agrupar regiones visuales en términos de proximidad, similitud o continuidad para construir un conjunto de píxeles, regiones o contornos que muestran una similitud en cuanto a intensidad, color, textura, u otra característica de interés [Oya03].

- Para la **reconstrucción**, **visualización** y **parametrización** se implementaron objetos gráficos que incluye IDL en su sistema grafico de objetos. Los objetos fueron diseñados con gran flexibilidad para la selección de diferentes escalas de color y diferentes grados de opacidad para brindar la mejor resolución visual en el proceso del modelamiento tridimensional (Cap. 4).

Las técnicas gráficas implementadas dentro de este trabajo fortalecerán significativamente el potencial de interpretación para los científicos, amplificando las posibles aplicaciones en esta área relevante de investigación.

[1] Scientific Volume Imaging B.V., www.svi.nl

1.2 Antecedentes Existentes

Algoritmos desarrollados en los últimos años para **reconstrucciones**, **visualizaciones** y **parametrizaciones** permitieron realizar experimentos importantes en distintas áreas científicas [Fan02, Jes02, Har03, Har04a/b, Alv03ab, Alv04, Alv05, Har05].

En el año 2002 se presentó la tesis "Plataforma de Identificación, Clasificación y Análisis de Imágenes, obtenidas a partir de Muestras Celulares" [Bar02] para el análisis de estructuras biológicas en 3 dimensiones I(x, y, c). En el 2003, la tesis "Segmentación y análisis automatizado de objetos en movimiento aplicado al estudio de sistemas biológicos" [Oya03] incluyó métodos para el análisis de estructuras en movimiento I(x, y, c, t). El presente trabajo de "Reconstrucción, visualización y parametrización tridimensional de estructuras biológicas a través de imágenes confocales de fluorescencia" es la continuación de los proyectos mencionados para el análisis de estructuras biológicas en 5 dimensiones I(x, y, z, c, t).

1.3 Lenguaje de Programación: Interactive Data Language (IDL)

IDL es un Software desarrollado por RSI (Research Systems, Inc., Boulder, CO. EEUU) a fines de los '70 para ayudar a la NASA a extraer información de datos a través de la visualización. Hoy en día la necesidad del procesamiento, análisis y visualización de datos se ha extendido a diversas áreas de estudio. Científicos y empresas en todo el mundo usan los productos y servicios de RSI para mejorar la evaluación e interpretación de sus datos. IDL es un lenguaje de cuarta generación orientado a objetos. El lenguaje proporciona métodos para el manejo de imágenes y estructuras de datos, que facilitan la implementación de algoritmos con un alto nivel de abstracción, y es adaptable para trabajar en cualquier sistema operativo (*Cross-Platform*).

Para los fines de este proyecto, IDL incluye una colección predefinida de clases de objetos gráficos por ejemplo IDLgrImage, IDLgrPolygon, IDLgrVolume. Estas clases de objetos

están diseñados para la construcción de complejas visualizaciones tridimensionales, y fueron implementados para el desarrollo de este trabajo (Cap. 4.3).

1.4 Objetivos del Proyecto

1.4.1 Objetivos Generales

Desarrollar e implementar técnicas interactivas para la reconstrucción y visualización tridimensional de estructuras biológicas.

1.4.2 Objetivos Específicos

Los objetivos específicos del proyecto de tesis son:

1. Realizar un estudio de las características y funcionalidades de objetos gráficos que incluye IDL en su biblioteca. Los objetos gráficos IDLgrImage, IDLgrPolygon, y IDLgrVolume permiten visualizaciones tridimensionales en distintos niveles de complejidad (dimensión, color, opacidad).
2. Implementar objetos gráficos para reconstruir, visualizar y parametrizar distintas características de estructuras biológicas. Las visualizaciones utilizarán modelos de volúmenes, cortes en volúmenes y sus respectivas proyecciones.
3. Calibración de la parametrización de los modelos gráficos a través de estructuras microscópicas de tamaños conocidos. Aplicación de técnicas de deconvolución, para corregir distorsiones debido a la difracción de fotones dentro del sistema óptico de la microscopía confocal.
4. Desarrollar un ambiente interactivo que permita el acceso rápido a parámetros intrínsecos de los objetos visualizados.

Capítulo 2 : Análisis de Imágenes

2.1 Imagen Digital

Una imagen digital puede ser definida como una función bidimensional de una señal con una intensidad I especifica reflejada o emitida en una región en el espacio que se proyecta en un plano xy en forma de I(x,y). Un píxel es la unidad mínima de representación dentro de una imagen, tiene una posición xy en una intensidad. La intensidad es representada por un valor entero, no negativo y finito, denominado escala o canal de gris. Las imágenes en escala de gris usadas en este trabajo tiene 8 bits de profundidad, quedando definido por la expresión I(x,y) $\in$ [0, 255] Fig. 2.1a/b. También existen imágenes de más bits de profundidad [12, 16, 24 bits]:

$$I(x_i, y_i) = \begin{pmatrix} x_0, y_0 & x_0, y_1 & \cdots & x_0, y_{n-1} \\ x_1, y_0 & x_1, y_1 & \cdots & x_1, y_{n-i} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m-1}, y_0 & x_{m-1}, y_i & \cdots & x_{m-1}, y_{n-1} \end{pmatrix}$$

Nota: IDL usa notación y_0, x_0; y_1, x_1, etc.

En una imagen I(x,y) es posible contar el número de píxeles que corresponden a cada intensidad en la escala de gris, y a la representación gráfica de esta característica se le llama **histograma de intensidades** (Fig. 2.1a/c). El histograma muestra la frecuencia (número de píxeles) vs. valores de la intensidad (entre 0 y 255).

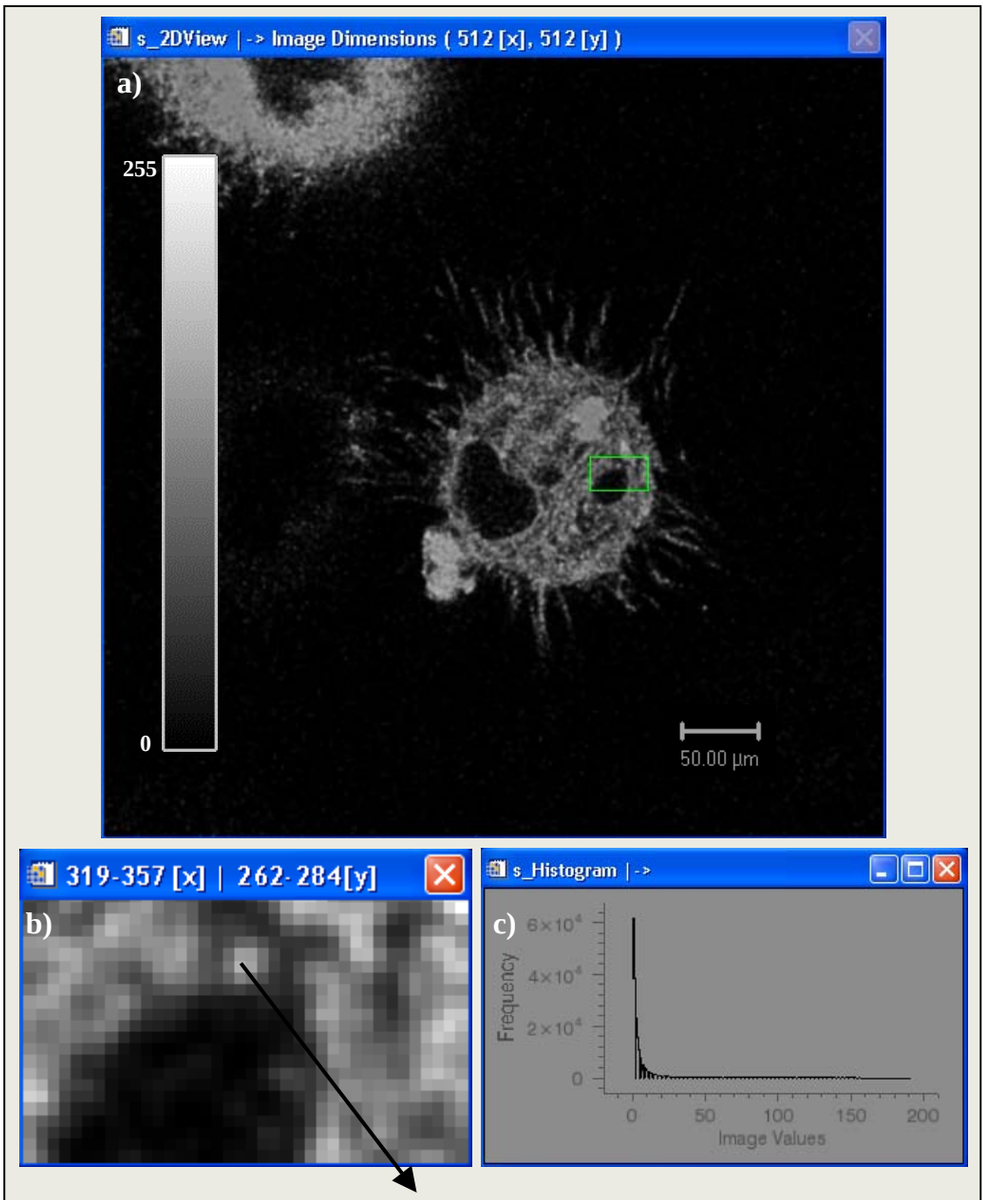


Fig. 2.1: Representación de una imagen digital y un histograma de intensidades.
[a] Imagen tomada en el eje z con un microscopio confocal de fluorescencia y dimensión 512x512 píxeles, el eje z tiene 76 cortes en total. La imagen corresponde al número 17. Las intensidades $I(x,y) \in [0, 255]$ están representadas en una escala de gris (barra izquierda), la barra indica una longitud de 50 µm (abajo a la derecha).
[b] La figura corresponde al cuadrante marcado en la figura [a] y muestra un zoom de 4x, el píxel señalizado tiene posición (338, 268) con intensidad 127 en la escala de gris.
[c] Histograma de intensidades de la figura [a] indica la frecuencia (número de píxeles) en una determinada intensidad de la imagen.

2.2 Pretratamiento: Restauración de Imágenes

La restauración de imágenes corrige errores y distorsiones que han sufrido los datos durante el proceso de adquisición. Existen diferentes algoritmos iterativos de restauración de imágenes que involucran parámetros de su sistema de adquisición. Uno de estos algoritmos es la deconvolución de imágenes, que se describe a continuación.

2.2.1 Introducción a la Deconvolución

La deconvolución es el proceso inverso de la convolución (Fig. 2.2). En microscopía confocal el proceso de convolución se produce durante la formación de la imagen y se caracteriza por la *point spread function* (PSF) del microscopio. La PSF determina como la intensidad de un punto en la muestra se dispersa. La imagen obtenida se compone por la convolución de cada punto en la muestra con la PSF y por la intensidad de cada punto [Kem99].

En general la deconvolución es un proceso que restaura la información dentro de una imagen que se ha perdido durante el proceso de adquisición. Se destacan principalmente dos factores que llevan a una perdida de información en una imagen: (i) la distorsión, producida por propiedades ópticas de los componentes del microscopio y las propiedades electromagnéticas de la luz (Fig. 2.2) y (ii) el ruido dentro de la imagen generado por fotones térmicos y ruido electrónico de los fotomultiplicadores. Ambos factores se dejan modelar a través de una PSF teórico, ruido intrínseco (irradiación de cuerpos negros que emiten fotones como el espejo, detector o la muestra biológica) y ruido extrínseco (Distribución de *Poisson* o la señal electrónica) [Kem99].

Para obtener una reaproximación al objeto real a partir de la imagen observada, es necesario aplicar el proceso de la **deconvolución**, operación que permite invertir el proceso, eliminando el ruido y las distorsiones introducidas por el sistema (Fig. 2.2 y 2.3).

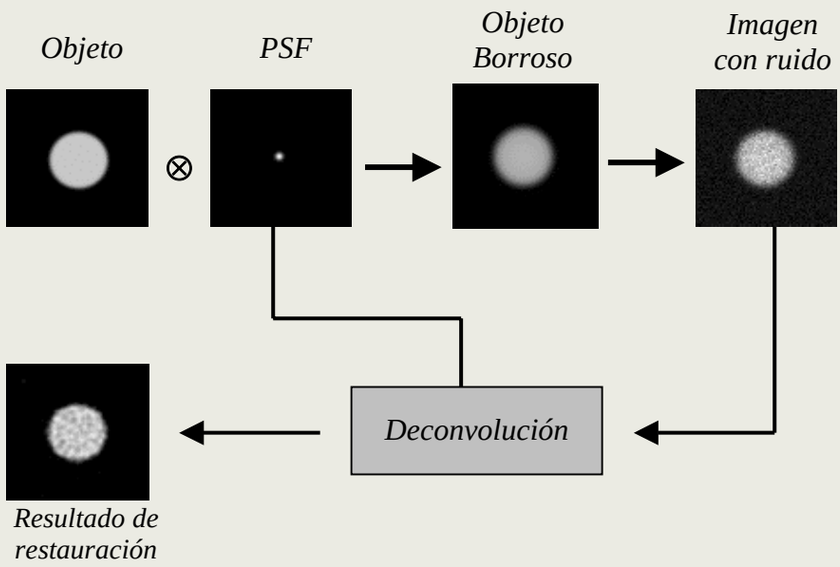


Fig.2.2: Diagrama de flujo general que refleja la adquisición de una imagen en un microscopio y restauración de la imagen. El símbolo ⊗ denota el operador matemático de convolución [Kem99].

La Fig. 2.3a muestra una imagen en su estado inicial de adquisición que presenta distorsiones y ruido. La Fig. 2.3b muestra el resultado de haber aplicado deconvolución, distinguiendo estructuras más definidas. Otro ejemplo de deconvolución se presenta al final del trabajo con una reconstrucción tridimensional (Anexo d).

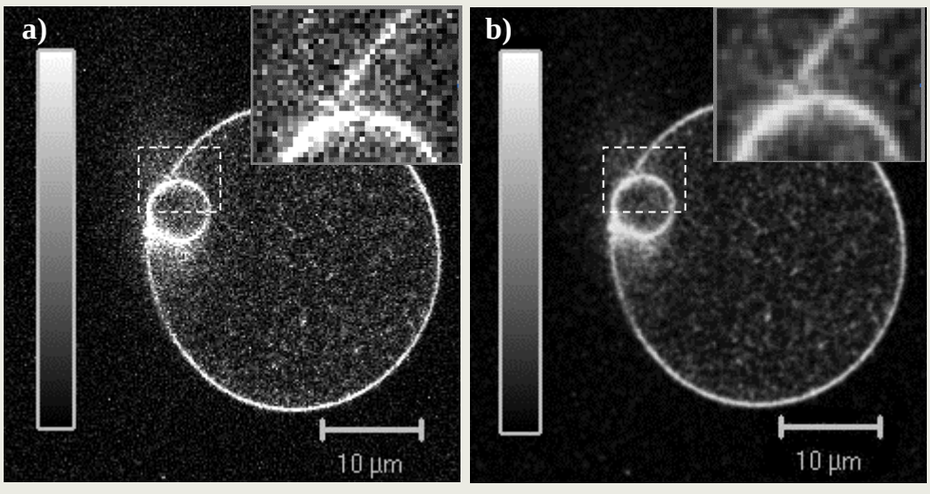


Fig.2.3: Resultado de una deconvolución a una imagen obtenida con un microscopio confocal de fluorescencia. En ambas imágenes se observa una membrana celular de una célula muerta, marcada con el fluoróforo FM1-43. Las barras marcan 10 µm. Las intensidades I(x,y) ∈ [0, 255], están representadas en una escala de gris (barra izquierda). El cuadrante en esquina superior derecha muestra un zoom de 4x de la zona marcada.
[a] Imagen original.
[b] Imagen deconvolucionada.

2.3 Segmentación

En el análisis de imágenes, la segmentación es un paso imprescindible para definir ROIs [URL2].

2.3.1 Segmentación por Umbralización

Existen diversas técnicas de segmentación de ROIs dentro de imágenes [URL3]. La segmentación utilizada en este trabajo utiliza técnicas de umbralización empleando un histograma de intensidades (Fig. 2.1 y Fig. 2.4). El umbral dentro de un histograma separa píxeles que corresponden a una ROI de las que no (*background*). La umbralización a partir de histogramas corresponde a elegir un rango de intensidades de píxeles que separan las ROI del fondo.

Una vez identificadas las ROI, se utilizan filtros morfológicos (Fig. 2.4c/d). Los filtros morfológicos son algoritmos que realizan operaciones estadísticas o matemáticas, para diferenciar y seleccionar estructuras con diferentes propiedades.

2.3.2 Aplicación de una Segmentación y Filtros Morfológicos

La Fig. 2.4 muestra una aplicación de segmentación y filtros morfológicos de una célula necrótica (muerte celular inducida por altas concentraciones de H_2O_2, peróxido). El umbral dentro del histograma utilizado para la segmentación corresponde al intervalo de intensidades $I(x,y) \in [45, 255]$. Los filtros morfológicos separan las estructuras pequeñas y las grandes. Otro ejemplo de filtros morfológicos se presenta al final del trabajo con una reconstrucción tridimensional (Anexo d).

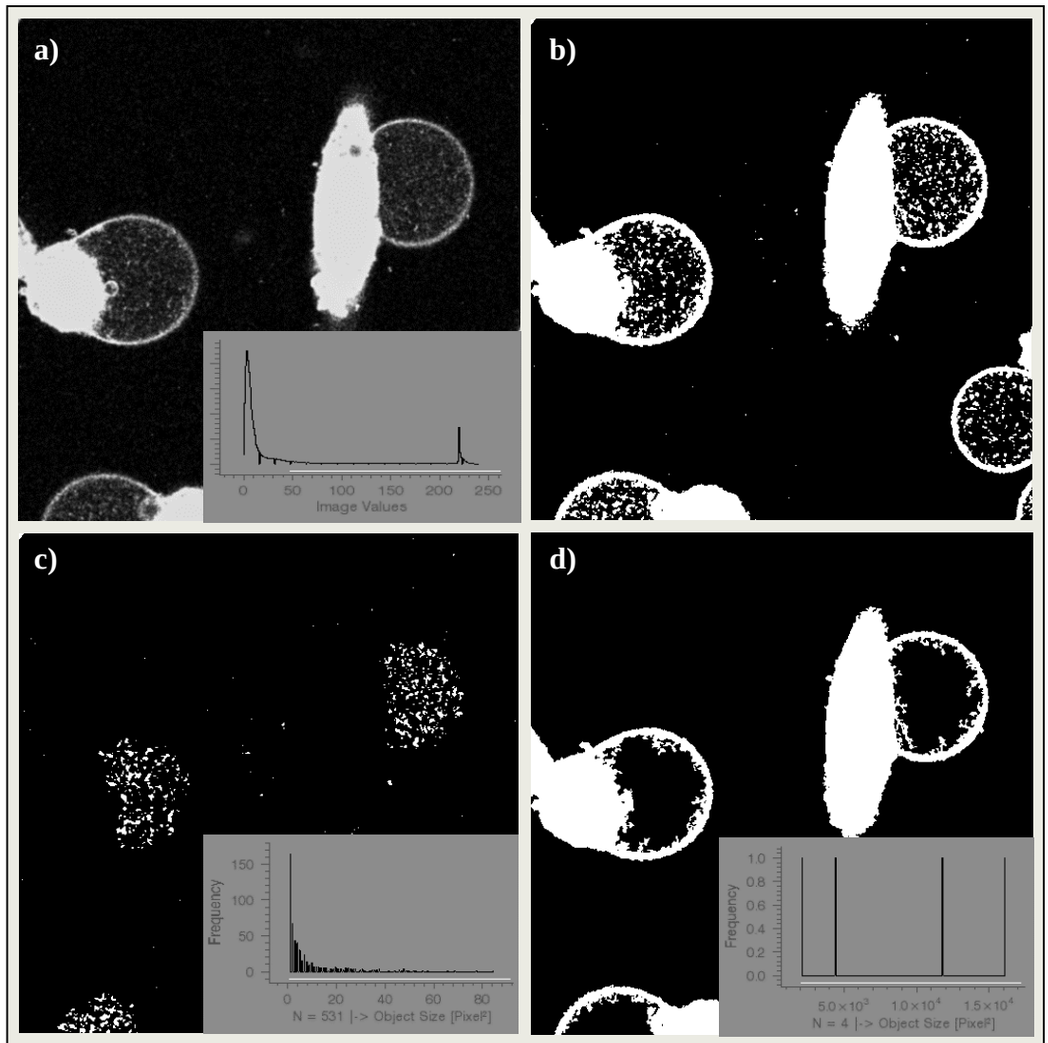


Fig. 2.4: Aplicación de una segmentación por umbralización y filtros morfológicos. La imagen fue obtenida con un microscopio confocal de fluorescencia, de 8 bits en la escala de gris I(x,y) ∈ [0, 255], con dimensiones xy de 512x512 píxeles. Las dimensiones reales en xy son 0.19x0.19 µm por pixel.
[a] Imagen original. La figura insertada es el histograma de intensidades de la imagen.
[b] Imagen segmentada a partir del umbral dentro del histograma marcado en [a].
[c] 1er filtro morfológico aplicado después de la segmentación, marcando las estructuras pequeñas. La figura insertada es el histograma de intensidades de la imagen, indicando el número de objetos identificados (531), y corresponde a la frecuencia vs. tamaño en píxel de cada objeto.
[d] 2do filtro morfológico aplicado después de la segmentación marcando las estructuras grandes. La figura insertada es el histograma de intensidades de la imagen, indicando el número de objetos identificados (4), y corresponde a la frecuencia vs. tamaño en píxel de cada objeto.

Capítulo 3 : Color en IDL

El color es una propiedad importante en el proceso de desplegar y visualizar imágenes. En el presente capítulo se analizan y describen los sistemas de color, los modelos de color, el color en la imagen, tablas de color frecuentemente usadas en IDL y canales de color dentro de la imagen y dentro de la microscopía confocal.

3.1 Sistemas de Color

El color puede ser codificado usando distintos esquemas [Gon02]. Algunos esquemas utilizan 3 valores para representar un color en una posición dentro de un espacio tridimensional. Ejemplos de estos sistemas incluidos en IDL son las mesas de color RGB (*red, green y blue*), HSV (*hue, saturation, y value*), HLS (*hue, lightness, y saturation*), o CMY (*cyan, magenta, y yellow*).

En IDL, el espacio de color RGB es representado como un sistema de coordenadas tridimensionales. Los ejes correspondientes al rojo, verde y azul (Fig. 3.1). Cada uno de los ejes tiene rangos entre 0 al 255. Un color individual es codificado como una coordenada dentro de este espacio [R, G, B].

La Fig. 3.1 muestra la combinación de todos los colores desplegables que corresponden a una posición dentro de un cubo de color tridimensional. El origen (0, 0, 0), corresponde al negro (dentro de la teoría del color). (255, 255, 255) corresponde al blanco, representando una mezcla aditiva del total de las intensidades de cada uno de los 3 colores. Los puntos a lo largo de la diagonal (cada una de las intensidades de los 3 colores primarios es igual) representa la escala de gris. El color amarillo es representado por la coordenada (255, 255, 0), o la mezcla del 100% rojo más 100% verde, etc. (ver Fig. 3.1).

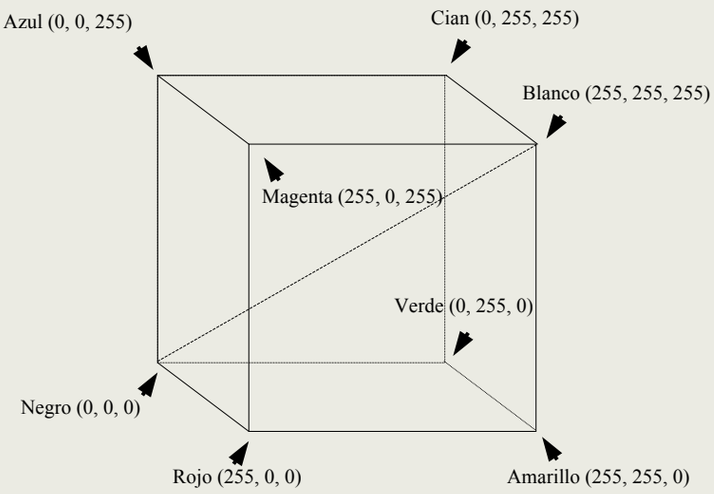


Fig. 3.1: Cubo de color RGB. Nota: Los matices del gris se encuentran en la diagonal.

3.2 Modelos de Color en los Objetos Gráficos de IDL

Los objetos gráficos en IDL usan 2 modelos de color:

- *Modelo Indexado*: Un color se especifica usando un índice dentro de una **tabla de búsqueda** (*Look-up table* (LUT)) de color o paleta de color con 256 colores distintos (Fig. 3.2). Cada entrada de la LUT corresponde a un color individual que consiste de un valor rojo, verde, y azul.
- *Modelo RGB*: Un color se especifica usando un triple RGB [*red, green, blue*]. El número de bits usado para representar cada uno de los componentes rojo, verde y azul son de 8 bits para cada uno.

El término visual, en IDL, se asocia a una interpretación particular del color para un dispositivo de despliegue determinado. La visual tiene dos nombres que indican como el color será representado. Estos nombres son el *PseudoColor* (que usa un esquema de color indexado) y *TrueColor* (que usa un esquema de color RGB).

Además una visual tiene una profundidad asociada con el que se describe cuantos bits son usados para representar un determinado color. La profundidad común incluye 8 bit (para la visual de *PseudoColor*) y 16 o 24 bits (para la visual *TrueColor*).

Una imagen de n–bit es capaz de desplegar 2n colores en total, así un visual *PseudoColor* de 8-bit puede desplegar 2^8 o 256 colores, y un visual *TrueColor* de 24-bit puede desplegar 2^{24} o 16.777.216 colores.

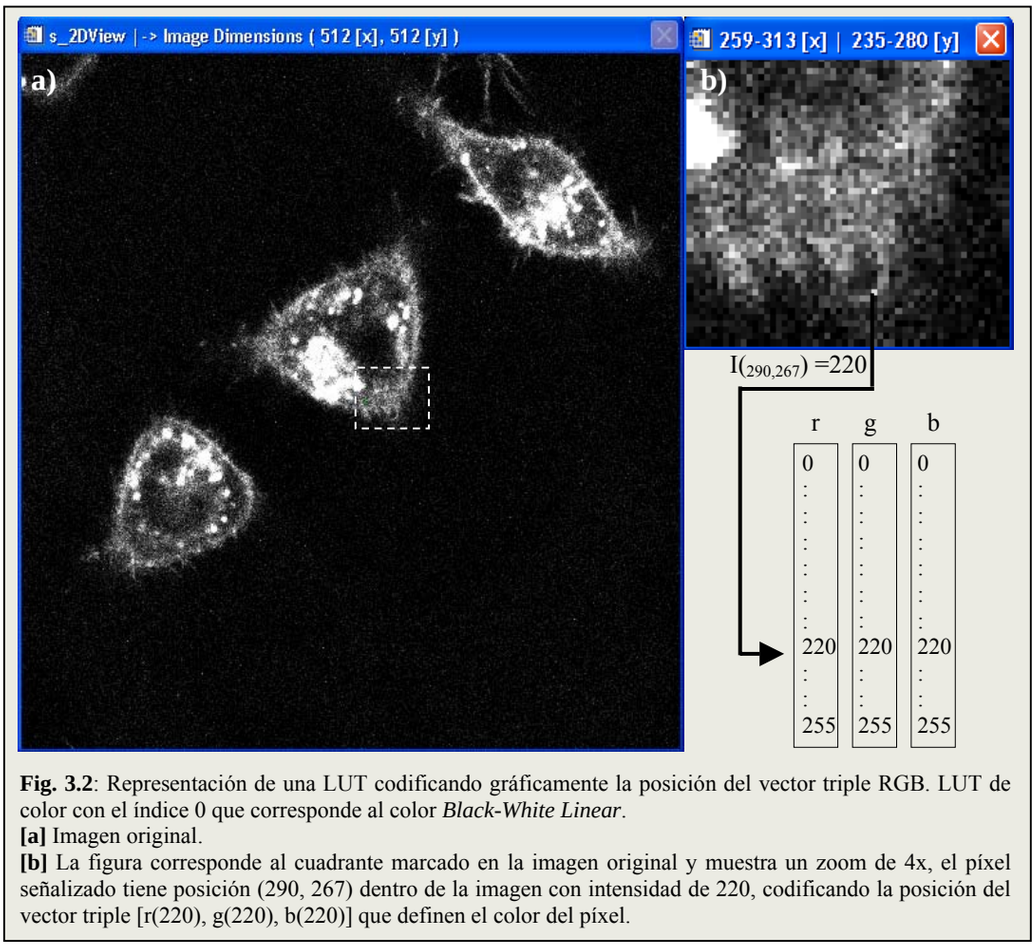


Fig. 3.2: Representación de una LUT codificando gráficamente la posición del vector triple RGB. LUT de color con el índice 0 que corresponde al color *Black-White Linear*.
[a] Imagen original.
[b] La figura corresponde al cuadrante marcado en la imagen original y muestra un zoom de 4x, el píxel señalizado tiene posición (290, 267) dentro de la imagen con intensidad de 220, codificando la posición del vector triple [r(220), g(220), b(220)] que definen el color del píxel.

La intensidad cuyos valores definen una LUT es codificado en las posiciones del vector triple RGB (Fig. 3.2), de esta forma se asigna color a imágenes que están en 1 canal de color (escala de gris).

3.3 El Color en la Imagen

El objeto IDLgrImage representa imágenes como objetos gráficos. Este almacena la información de la imagen utilizando un tipo de dato byte y puede tomar cualquiera de las siguientes formas:

- Un arreglo con dimensiones [x, y]. Cada píxel es interpretado como un índice dentro de una LUT o como un valor de escala gris (Cap. 3.4).
- Un arreglo con dimensiones [2, x, y]. Cada píxel consiste de un valor de escala gris [x, y] y un valor de canal *alpha (α)*, [α, x, y] asociado (α es un valor equivalente a un canal R, G, B cuyo valor $\alpha \in [0, 255]$ codifica la opacidad o transparencia de cada píxel) (Cap. 3.5).
- Un arreglo con dimensiones [3, x, y]. Cada píxel consiste de un triple RGB. [R, x, y], [G, x, y], [B, x, y].
- Un arreglo con dimensiones [4, x, y]. Cada píxel consiste de un triple RGB y un valor de canal α asociado. [R, x, y],..., [α, x, y].

3.4 LUT

IDL proporciona una lista de 41 LUT predefinidos. En el transcurso de este trabajo fueron implementadas 12 LUT adicionales. Con la rutina LOADCT que incluye IDL se puede tener acceso a cualquier LUT. Cada una se especifica por medio de un valor con índice que comprende del 0 al 52 y sirven para resaltar ciertas características dentro de una imagen. La tabla 1.1 contiene los índices con el nombre de las LUT.

0	B-W Linear	27	Eos B
1	Blue/White	28	Hardcandy
2	Grn-Red-Blu-Wht	29	Nature
3	Red Temperatura	30	Ocean

4	Blue/Green/Red/Ye	31	Peppermint
5	Std Gamma-Ii	32	Plasma
6	Prism	33	Blue-Red
7	Red-Purple	34	Rainbow
8	Green/White Linea	35	Blue Waves
9	Grn/Wht Exponente	36	Volcano
10	Green-Pink	37	Waves
11	Blue-Red	38	Rainbow18
12	16 Level	39	Rainbow + white
13	Rainbow	40	Rainbow + black
14	Steps	**41**	**Black-Red**
15	Stern Special	**42**	**Black-Green**
16	Haze	**43**	**Black-Blue**
17	Blue - Pastel – R	**44**	**Black-Yellow**
18	Pastels	**45**	**Black-Cian**
19	Hue Sat Lightness 1	**46**	**Black-Magenta**
20	Hue Sat Lightness 2	**47**	**White-Grey-Red**
21	Hue Sat Value 1	**48**	**White-Grey-Green**
22	Hue Sat Value 2	**49**	**White-Grey-Blue**
23	Purple-Red + Stri	**50**	**White-Red**
24	Beach	**51**	**White-Green**
25	Mac Style	**52**	**White-Blue**
26	Eos A		

Tabla 1.1: Tablas de color en IDL. Los índices del 0 al 40 son tablas de color predefinidas por IDL, del 41 al 52 son tablas de color implementadas en el desarrollo del trabajo.

En la Fig. 3.3 se muestra un ejemplo de la rutina LOADCT aplicando 2 LUT a una imagen en escala de gris.

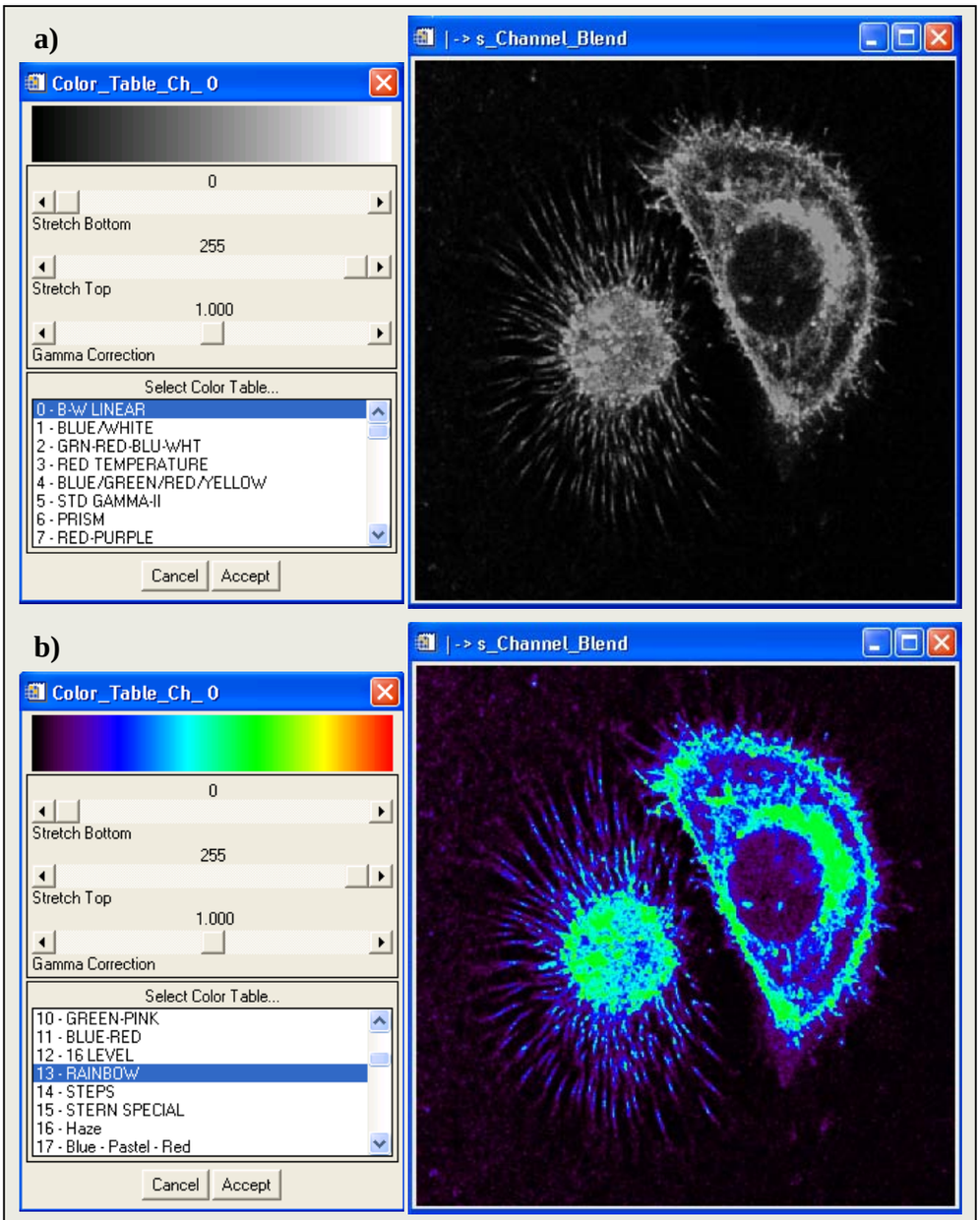


Fig. 3.3: Resultado de aplicar diferentes LUT de color a una imagen de 8 bit con I(x,y) ∈ [0, 255]. Imagen tomada en el eje z con un microscopio confocal de fluorescencia y dimensión 512x512 píxeles .El eje z tiene 56 cortes en total y corresponde al corte número 11.
[a] La imagen tiene aplicada una LUT de color con el índice 0 que corresponde al color *Black-White Linear*. Ver figura anterior (diagonal del cubo de color)
[b] La imagen tiene aplicada una LUT de color con el índice 13 que corresponde al color *Rainbow*.

Existe una ventana interactiva que permite hacer ajustes a las LUT a través de los parámetros *Stretch Bottom*, *Stretch Top* y *Gamma Correction*. Estos sliders se describen de la siguiente manera:

Stretch Bottom: Asigna el color que inicialmente tiene una LUT en la intensidad más baja, a un rango de intensidades más altas dentro de la imagen, en Fig. 3.4 *Stretch Bottom* = 15.

Stretch Top: Asigna el color que inicialmente tiene una LUT en la intensidad más alta, a un rango de intensidades más bajas dentro de la imagen. En Fig. 3.4 Strech Top = 135.

Gamma Correction: Este Slider se utiliza para ajustar y redistribuir los colores entre las intensidades bajas y altas dentro de la imagen. Un valor de 1.0 indica una rampa lineal (sin *gamma correction*). Valores más altos que 1.0 aplican una rampa exponencial y valores más bajos que 1.0 aplican una rampa logarítmica (dependerá del *Stretch Top* y *Stretch Bottom*).

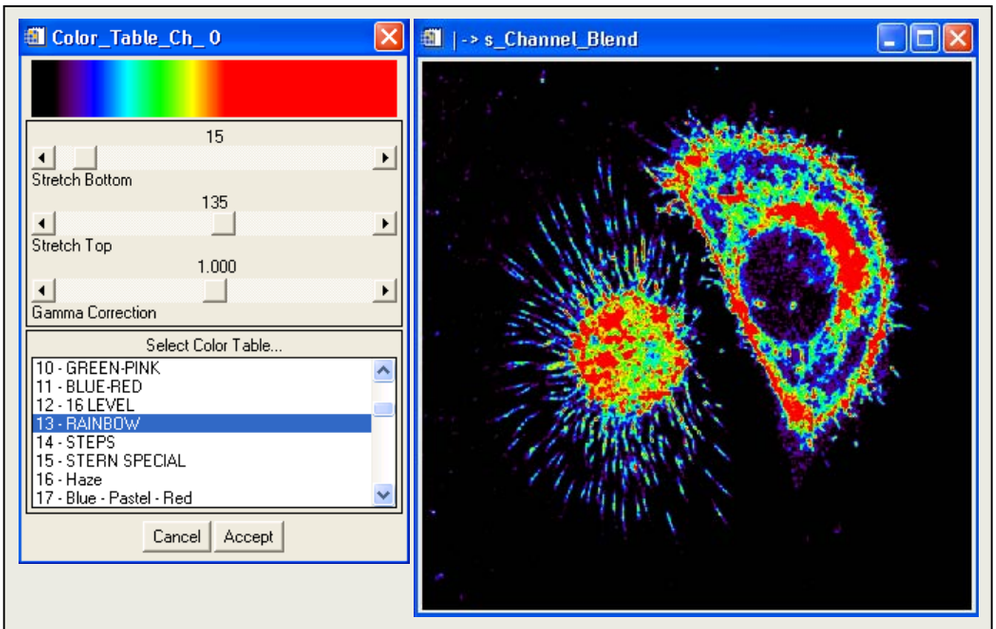


Fig. 3.4: Resultado de una imagen modificada por los parámetros *Stretch Bottom* y *Stretch Top*. La figura de la izquierda es la LUT de color con el índice 13 que corresponde al color *Rainbow*. El parámetro *Stretch Bottom* que inicialmente tiene el color negro en 0, si es fijado en 15 indica que todos los píxeles con intensidades entre 0 y 15 adoptan el color negro en la imagen. El parámetro *Stretch Top* que inicialmente tiene el color rojo en 255, si es fijado en 135 indica que todos los píxeles con intensidades entre 135 y 255 adoptan el color rojo en la imagen.

3.5 Canales de Colores dentro de Imágenes y dentro de la Microscopía Confocal

Los canales de color almacenan la información de los colores de la imagen, la cantidad de canales de una imagen dependerá de su modo de color. Por ejemplo, una imagen RGB tiene 3 canales (Cap. 3.3), uno para la información del rojo, el verde y el azul (ver Fig. 3.2). Las imágenes en escala de gris, mapa de bits y color indexado tienen un solo canal.

En la microscopía confocal una muestra celular se puede marcar con fluoróforos (colorantes fluorescentes) los cuales se unen de forma especifica a estructuras concretas en la célula. El término canal en microscopía se utiliza para distinguir de una misma muestra biológica las distintas longitudes de ondas que excitan o emiten los diferentes moléculas fluorescentes. La imagen de la muestra biológica es capturada en un canal de color con intensidades lineales $I(x,y) \in [0, 255]$ que es proporcional a la cantidad de fotones detectados en una cierta banda de frecuencias (Fig. 3.5a/b). Asignar colores a cada canal de la imagen permite distinguir entre un canal y otro.

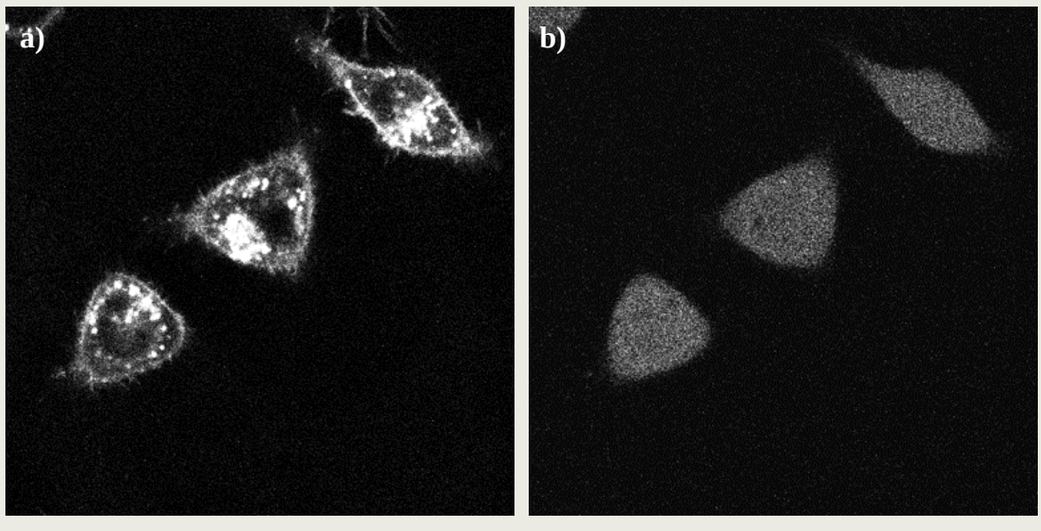


Fig. 3.5: Imagen confocal de fluorescencia de tres células en dos canales de fluorescencia $I(x,y,z,c,t) = I(512,512,21,2,22) \in [0, 255]$.
[a] La membrana de la célula está marcadas con el fluoróforo $DiIC_{18}$.
[b] El interior de la células está marcadas con el fluoróforo calceina.

Una forma de asignar color a un canal de una imagen es a través de un LUT (Cap. 3.2). Una vez asignada una LUT a cada canal de la imagen (Fig. 3.6), es posible realizar una superposición de los colores de cada canal para visualizar características de ambos (Fig. 3.7).

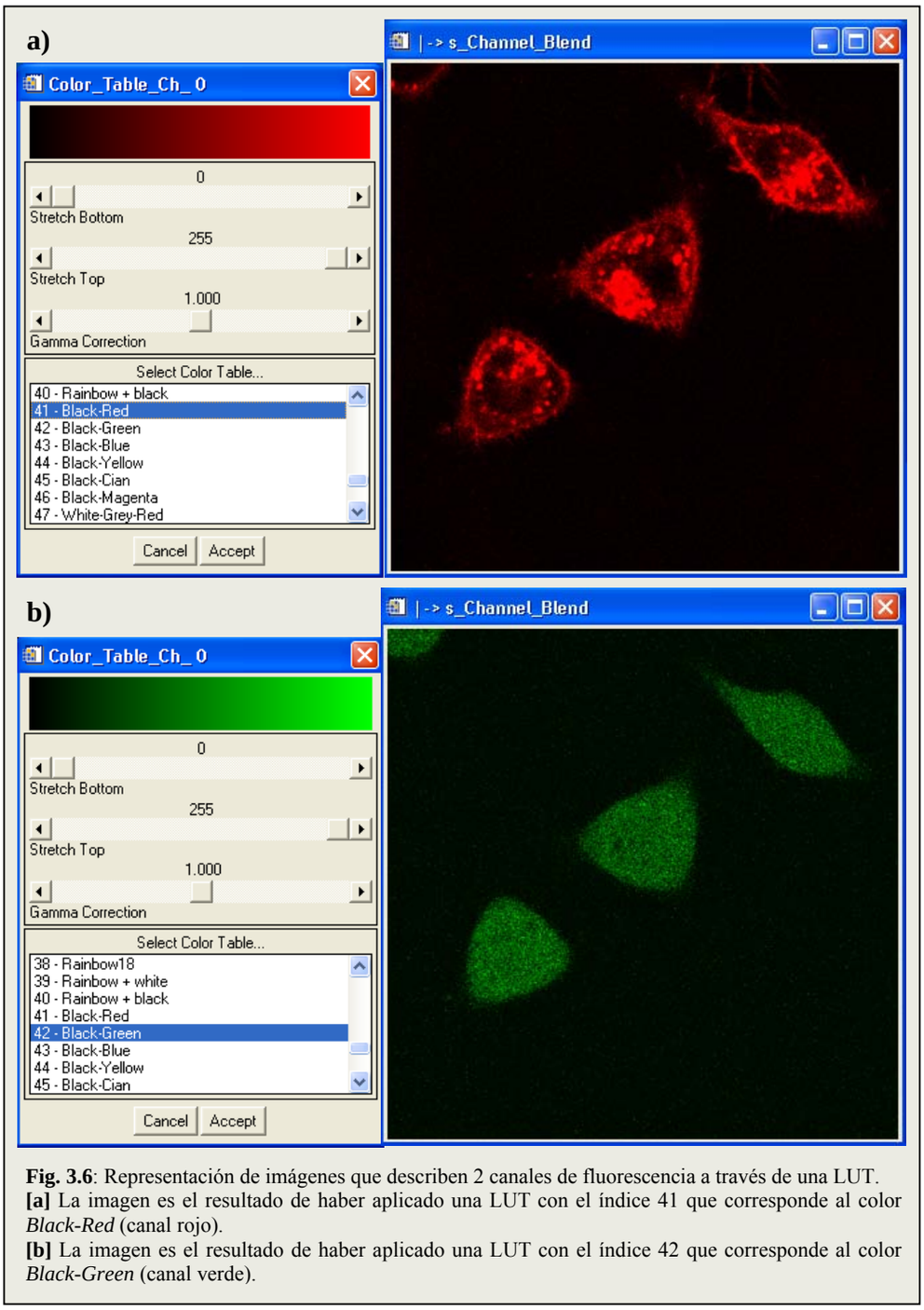
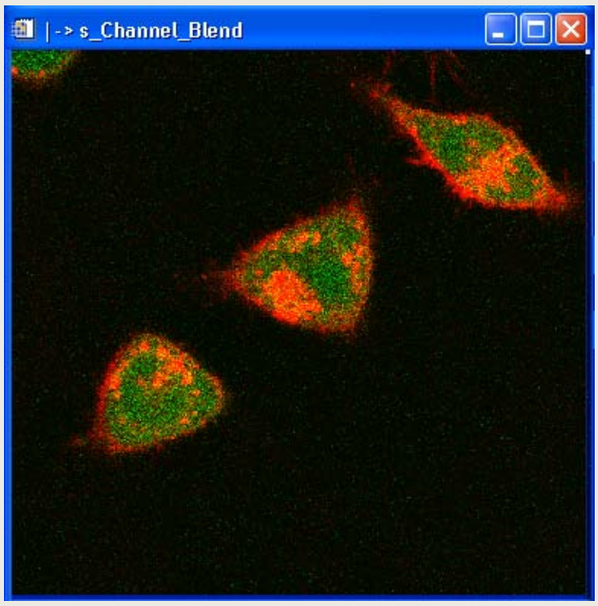


Fig. 3.6: Representación de imágenes que describen 2 canales de fluorescencia a través de una LUT.
[a] La imagen es el resultado de haber aplicado una LUT con el índice 41 que corresponde al color *Black-Red* (canal rojo).
[b] La imagen es el resultado de haber aplicado una LUT con el índice 42 que corresponde al color *Black-Green* (canal verde).



Canal 1

r_1	g_1	b_1
0	0	0
:	:	:
220	0	0
:	:	:
255	0	0

Canal 2

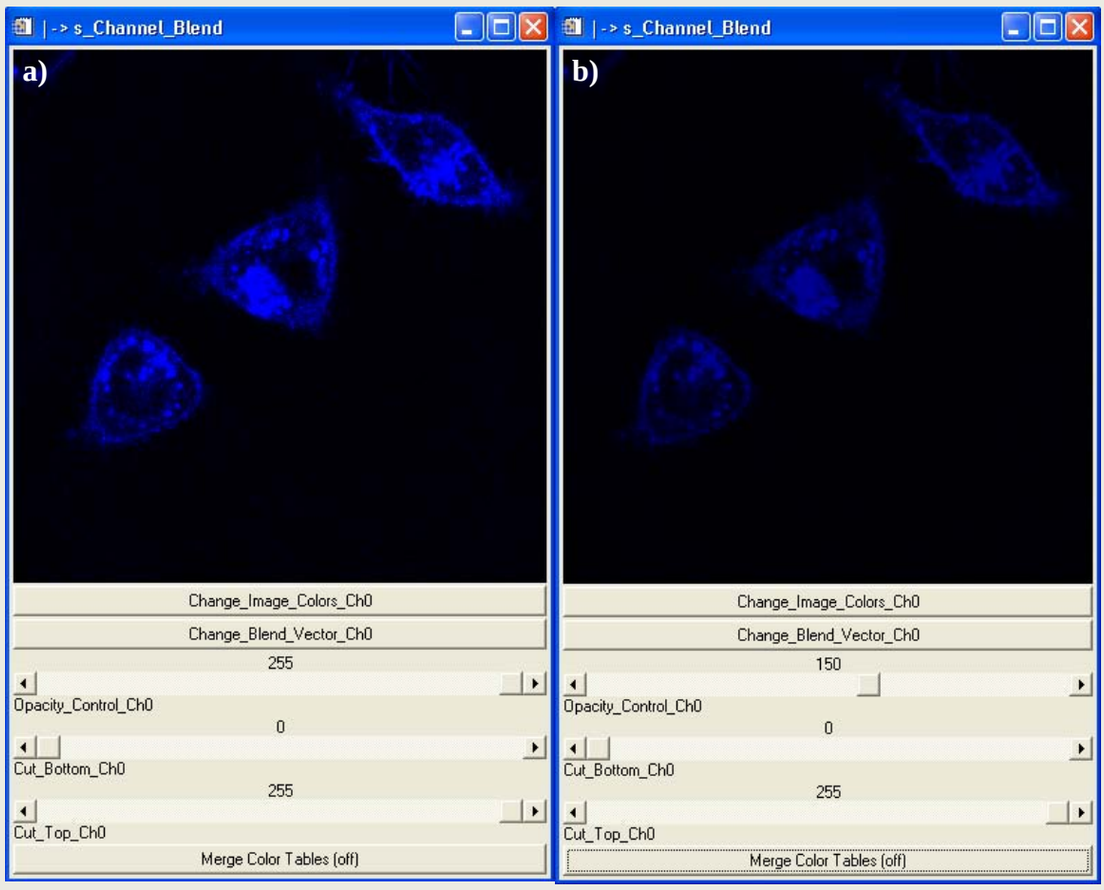
r_2	g_2	b_2
0	0	0
:	:	:
0	220	0
:	:	:
0	255	0

Imagen final = $[r_1 + r_2, g_1 + g_2, b_1 + b_2] < 255$

Fig. 3.7: Superposición entre un canal rojo y un canal verde representando gráficamente el vector triple RGB. Cada canal esta descrito por una LUT de color (Fig. 3.6). La superposición se produce por la suma de las intensidades de los vectores RGB de ambos canales y debe cumplirse que $[r_1 + r_2, g_1 + g_2, b_1 + b_2] < 255$.

3.6 Opacidad o Canal α

Otra característica importante es la **opacidad** o nivel de transparencia de un píxel, denominado canal alfa (α) (Cap. 3.3). Este canal actúa de forma independiente al color establecido. En la Fig. 3.8 se puede observar el efecto del canal α sobre una imagen. Si el valor de α es igual 255 la imagen adopta una opacidad máxima (sin transparencia). Si el valor de α es igual 0 la imagen tiene opacidad mínima (transparencia total).



$r = [r_0, r_1, r_2, \ldots, r_{255}] = [0, 0, 0, \ldots, 0]$
$g = [g_0, g_1, g_2, \ldots, g_{255}] = [0, 0, 0, \ldots, 0]$
$b = [b_0, b_1, b_2, \ldots, b_{255}] = [0, 1, 2, \ldots, 255]$
$o = [o_0, o_1, o_2, \ldots, o_{255}] = [255, 255, \ldots, 255]$

$r = [r_0, r_1, r_2, \ldots, r_{255}] = [0, 0, 0, \ldots, 0]$
$g = [g_0, g_1, g_2, \ldots, g_{255}] = [0, 0, 0, \ldots, 0]$
$b = [b_0, b_1, b_2, \ldots, b_{255}] = [0, 1, 2, \ldots, 255]$
$o = [o_0, o_1, o_2, \ldots, o_{255}] = [150, 150, \ldots, 150]$

Fig. 3.8: Representación de una imagen con distinto control de opacidad. Las imágenes tienen asignada una LUT de color con el índice 43 que corresponde al color *Black-Blue* (canal azul).
[a] La imagen con el Slider Opacity_Control establecido en 255 indica que no hay opacidad (transparencia total).
[b] La imagen con el Slider Opacity_Control establecido en 150 indica que tiene poca opacidad.

Capítulo 4 : Objetos Gráficos en IDL

El sistema de objetos gráficos de IDL contiene una colección de clases de objetos predefinidos: IDLgrAxis, IDLgrContour, IDLgrImage, IDLgrLight, IDLgrPlot, IDLgrPolygon, IDLgrPolyline, IDLgrSurface, IDLgrText, o IDLgrVolume. Cada objeto está diseñado para encapsular una representación visual en particular. Acciones como la modificación de atributos o selección de datos pueden ser ejecutados sobre instancias de estas clases de objetos por la correspondiente llamada a los métodos predefinidos. En el presente capítulo se describe como IDL organiza los objetos gráficos de su sistema y se presentan los objetos gráficos implementados.

4.1 Jerarquía de Objetos Gráficos en IDL

IDL organiza grupos de objetos gráficos por medio de una jerarquía o árbol (Fig. 4.1). Una escena grafica puede tener cualquier número de ramas de objetos, cada uno de los cuales puede contener a su vez sub-ramas de objetos.

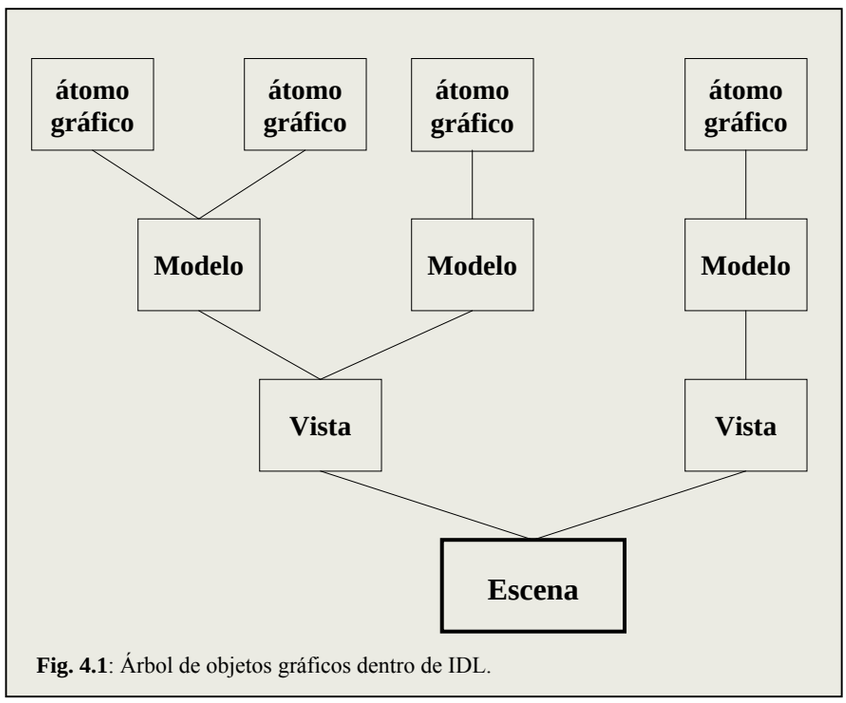


Fig. 4.1: Árbol de objetos gráficos dentro de IDL.

4.2 Clases y Objetos Gráficos en IDL

Un árbol gráfico en IDL está compuesto de las siguientes clases de objetos:

Átomos Gráficos: Un objeto gráfico atómico, o átomos gráficos, son los objetos de nivel más bajos para crear una escena. Es una instancia de una de las siguientes clases: IDLgrAxis, IDLgrContour, IDLgrImage, IDLgrLight, IDLgrPlot, IDLgrPolygon, IDLgrPolyline, IDLgrSurface, IDLgrText, o IDLgrVolume. Átomos gráficos incluidos en un modelo (utilizando el método "*Add"* del objeto modelo) comparten la misma matriz de transformación tridimensional pueden ser rotados, escalados, o trasladados simultáneamente (Fig. 4.2c_1/c_2).

Modelos: Los objetos de la clase IDLgrModel son contenedores de los átomos gráficos (ver arriba). Los modelos incluyen una matriz de transformación tridimensional, que describe como los modelos y todos sus componentes son posicionados en el espacio. Alterando la matriz de transformación cambia la posición y orientación de cualquier objeto contenido en el modelo (Fig. 4.2b_1/c_3/c_4).

Vistas: Objetos de la clase IDLgrView son contenedores de objetos modelos. Una vista o instancia de la clase IDLgrView puede servir como objeto raíz de árboles gráficos. Instancias de la clase IDLgrView tienen los métodos que permiten incluir o remover objetos IDLgrModel en una vista (Fig. 4.2b/c).

Escenas: Objetos de la clase IDLgrScene sirven como contenedor de objetos vistas y grupos de vistas. Una escena es el objeto raíz de un árbol de objetos gráficos. Instancias de la clase IDLgrScene tienen los métodos que permiten incluir o remover objetos vistas o grupos de vistas en una escena (Fig. 4.2a).

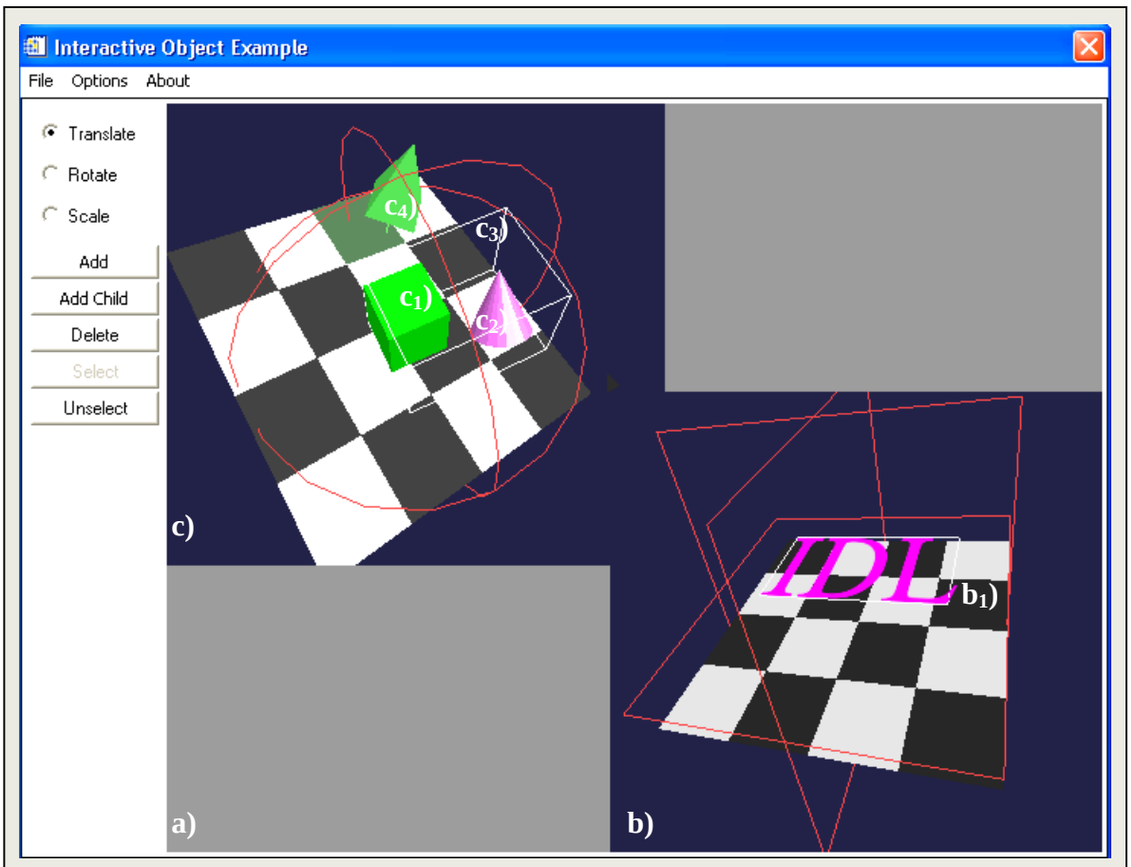


Fig. 4.2: Imagen de una ventana interactiva que permite utilizar una escena tridimensionalmente con los objetos gráficos de IDL La vista es la base representada en cuadros blancos y negros, actúa en forma separada del modelo. Un modelo se distingue de una vista, ya que este se puede trasladar, rotar y escalar. La vista solo cambia de dirección.
[a] Escena, instancia de la clase IDLgrEscene que sirve como contenedor de vistas y grupos de vistas. La escena contiene 2 vistas b) y c).
[b]Vista$_1$, instancia de la clase IDLgrView, incluye un modelo b$_1$), y este a su vez incluye 1 átomo gráfico.
[c]Vista$_2$, instancia de la clase IDLgrView, incluye 2 modelos c$_3$), y c$_4$). El modelo c$_3$) contiene 2 átomos gráficos c$_1$), y c$_2$).El modelo c$_4$) contiene 1 átomo gráfico.

4.3 Interacción con Objetos Gráficos

Los objetos gráficos se diseñaron para la construcción de complejas aplicaciones tridimensionales interactivas. A continuación se presentan los objetos gráficos con los parámetros implementados en el trabajo.

Para la presentación de los 2 primeros objetos gráficos implementados en Cap. 4.3.1/4.3.2 se utilizó la imagen en Fig. 4.3.

En el 3er objeto gráfico implementado (Cap. 4.3.3) se utilizaron todos los cortes en el eje z para la reconstrucción tridimensional.

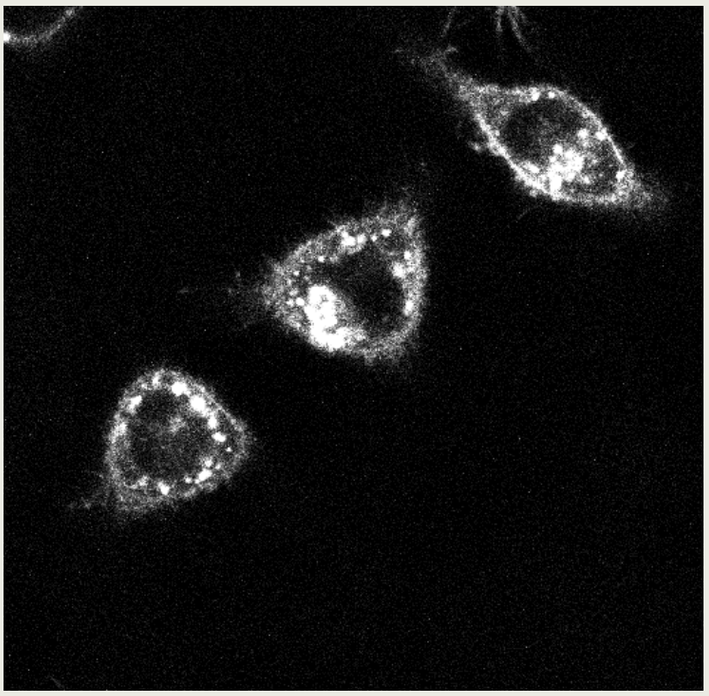


Fig. 4.3: Imagen utilizada para la presentación de los objetos gráficos implementados.

La mayoría de los parámetros predefinidos de cada objeto gráfico a presentar fueron implementados. Los parámetros que mostraban cambios significativos y cumplieron con los objetivos del trabajo se implementaron interactivamente de tal forma que los cambios en el objeto reconstruido fueran visualizados en un entorno interactivo, como se verá en una aplicación más adelante (Cap. 6.4).

4.3.1 IDLgrImage

El objeto IDLgrImage representa una organización a partir de un arreglo bidimensional de valores de datos a un arreglo bidimensional de colores de píxel, dando como resultado una versión plana bidimensional ajustada de la imagen, dibujado en Z = 0. IDLgrImage contiene parámetros predefinidos (Anexo A.1). La Fig. 4.4 muestra una representación del objeto IDLgrImage.

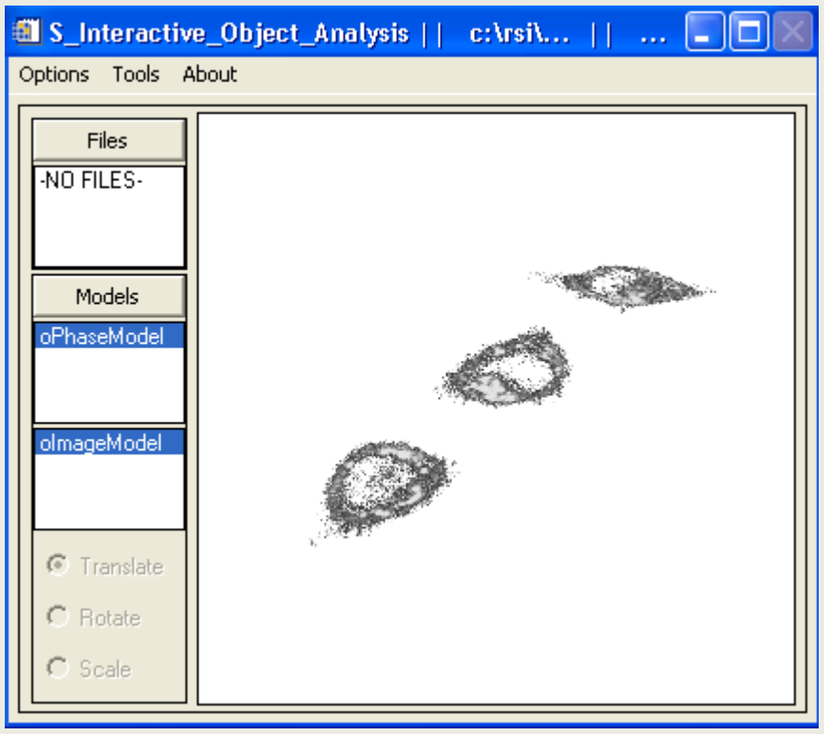


Fig. 4.4: Representación del objeto IDLgrImage en un espacio tridimensional. El modelo activo que representa al objeto es oImageModel.

4.3.2 IDLgrPolygon

Un objeto polígono IDLgrPolygon representa uno o más polígonos que comparten un conjunto dado de vértices y atributos. En IDL los polígonos deben ser convexos, es decir, si se prolongan los lados de un polígono, toda la figura debe quedar al mismo lado, de lo contrario será cóncavo. Los polígonos cóncavos deben ser convertidos a convexos usando un proceso de *Tesselation* a través del objeto IDLgrTessellator. IDLgrPolygon contiene

parámetros predefinidos (Anexo A.2), de los cuales fueron parametrizados interactivamente los siguientes:

Color: Esta palabra clave es adaptada para cambiar el color de un polígono, siendo representada como RGB (*Red, Green, Blue*) y la combinación de todos los colores (Fig. 4.5), el valor por defecto es negro [0, 0, 0].

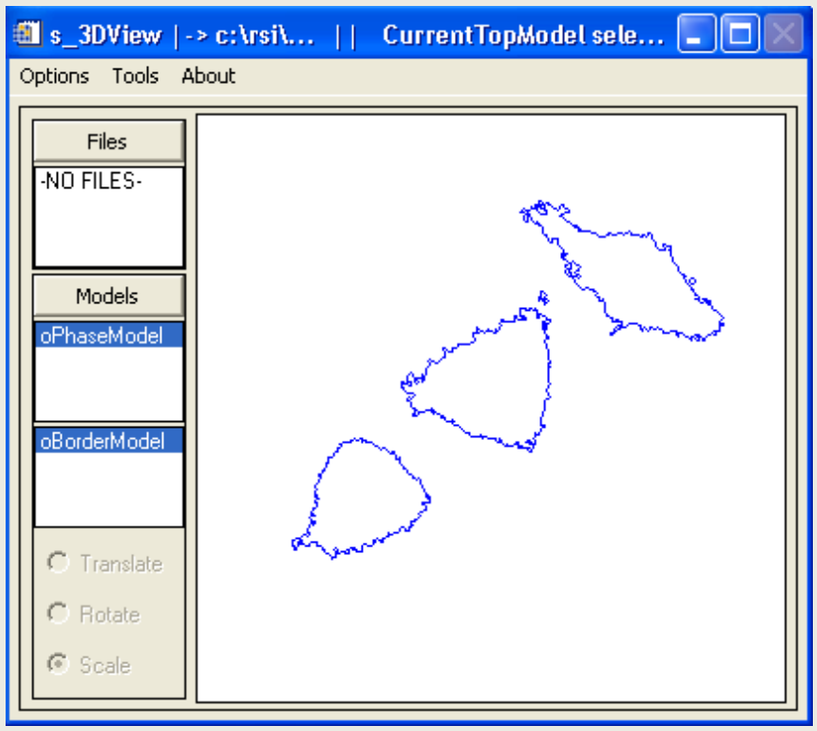


Fig. 4.5: Representación del parámetro **Color** del objeto IDLgrPolygone. El objeto esta en un espacio tridimensional, el modelo activo que representa al objeto es oBorderModel. La figura muestra el parámetro **Color** fijado en [0, 0, 255] representando el color azul con toda su intensidad.

Style: Esta palabra clave es adaptada para determinar si un polígono, es dibujado con puntos, líneas o relleno (Fig. 4.6a/b/c). Los valores que toma Style son 0,1 o 2.

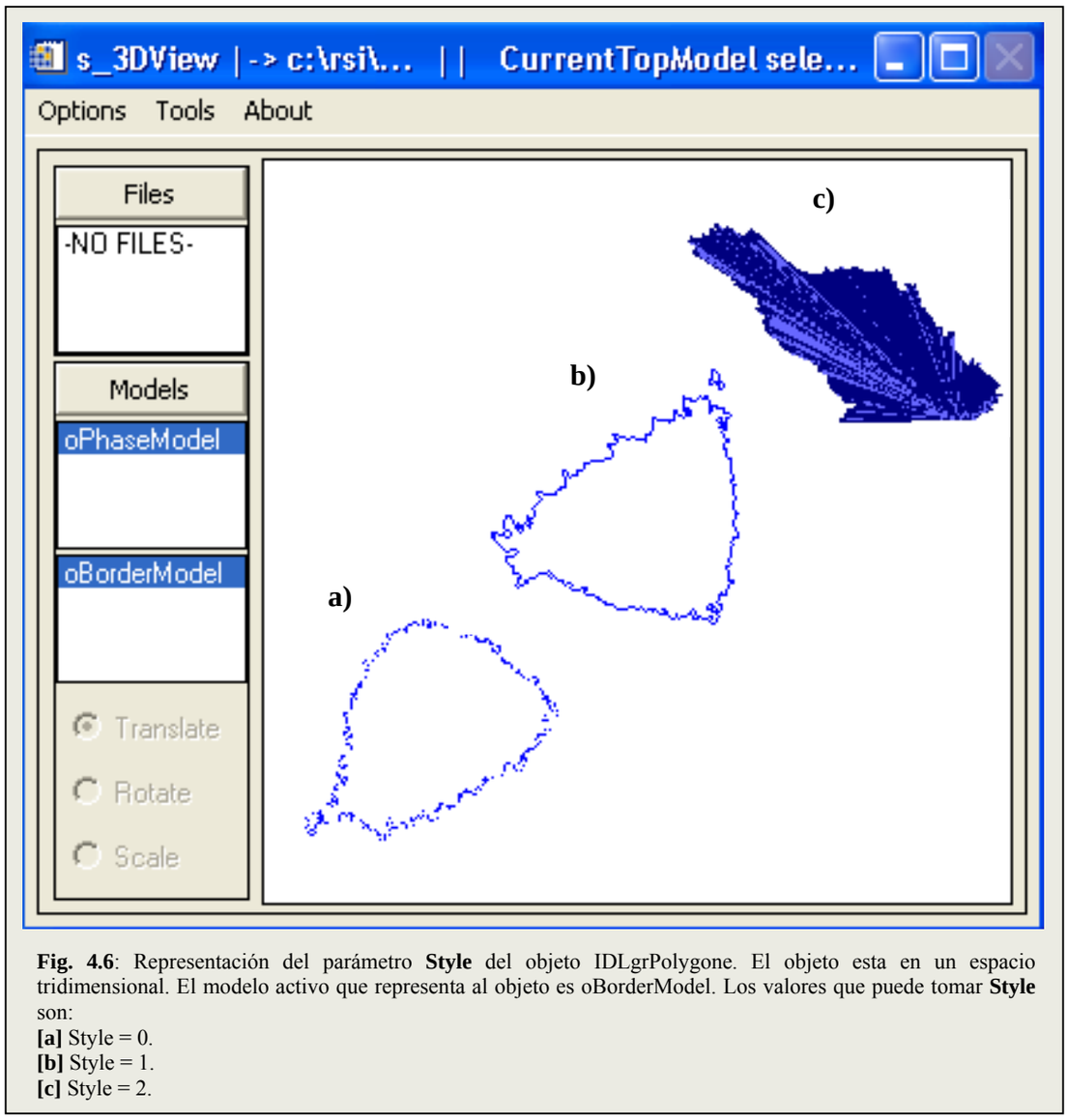


Fig. 4.6: Representación del parámetro **Style** del objeto IDLgrPolygone. El objeto esta en un espacio tridimensional. El modelo activo que representa al objeto es oBorderModel. Los valores que puede tomar **Style** son:
[a] Style = 0.
[b] Style = 1.
[c] Style = 2.

LineStyle: Esta palabra clave es adaptada para indicar el estilo de línea con el cual será dibujado el polígono (Fig. 4.7); los valores enteros que toma LineStyle para determinar su representación van de 1 al 6.

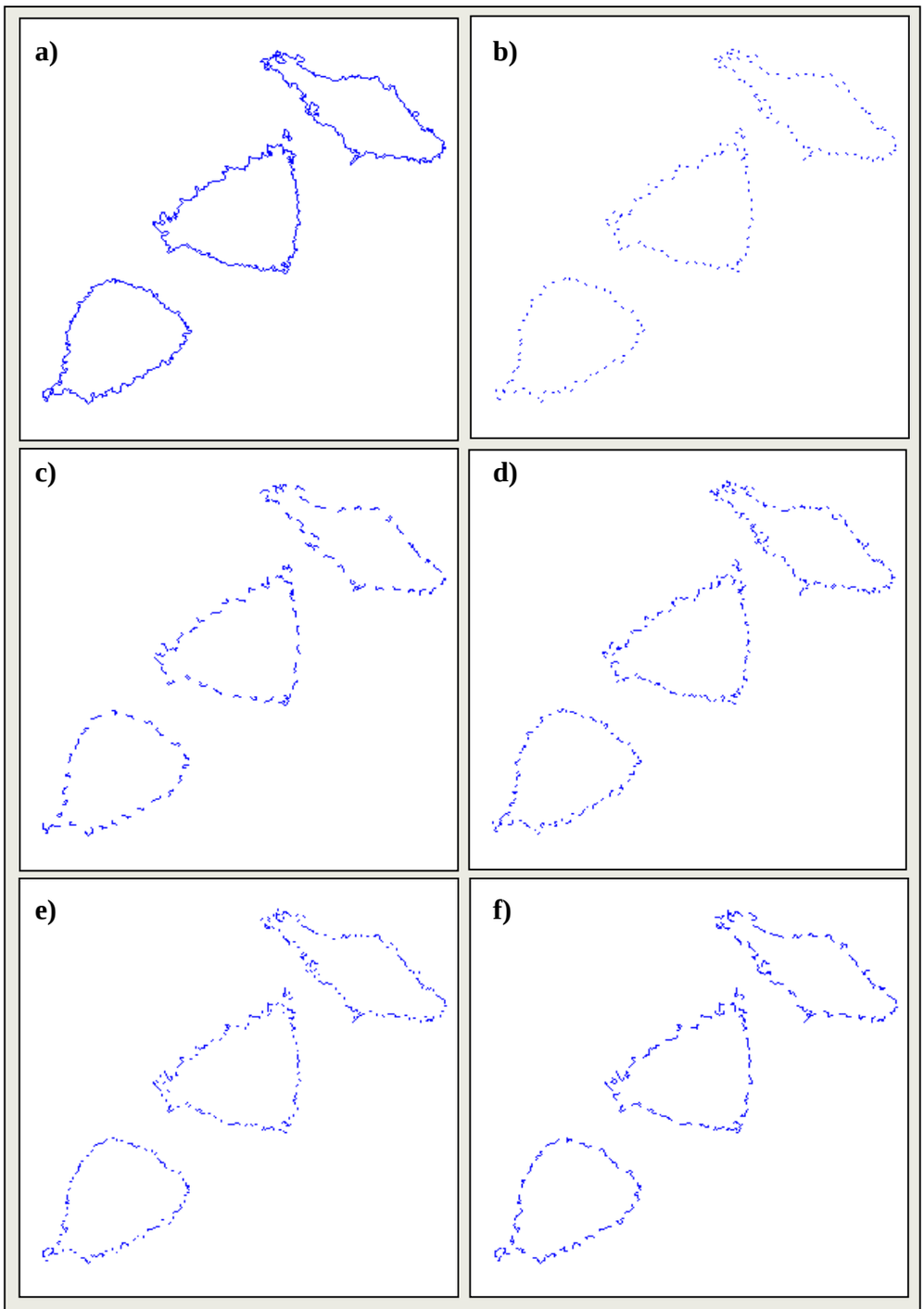


Fig. 4.7: Representación del parámetro **LineStyle** del objeto IDLgrPolygone. Los valores que puede tomar **LineStyle** para representar estilos de líneas son:
[a] LineStyle = 1, sólida.
[b] LineStyle = 2, punteada.
[c] LineStyle = 3, guión.
[d] LineStyle = 4, guión y punto.
[e] LineStyle = 5, guión y punto punto punto.
[f] LineStyle = 6, guión largo.

Thick: Determina el grosor del polígono. Thick puede tomar valores entre 1.0 y 10.0. Si Style = 0 el polígono se representa por puntos. Si Style = 1 el polígono será dibujado en línea (Fig. 4.8).

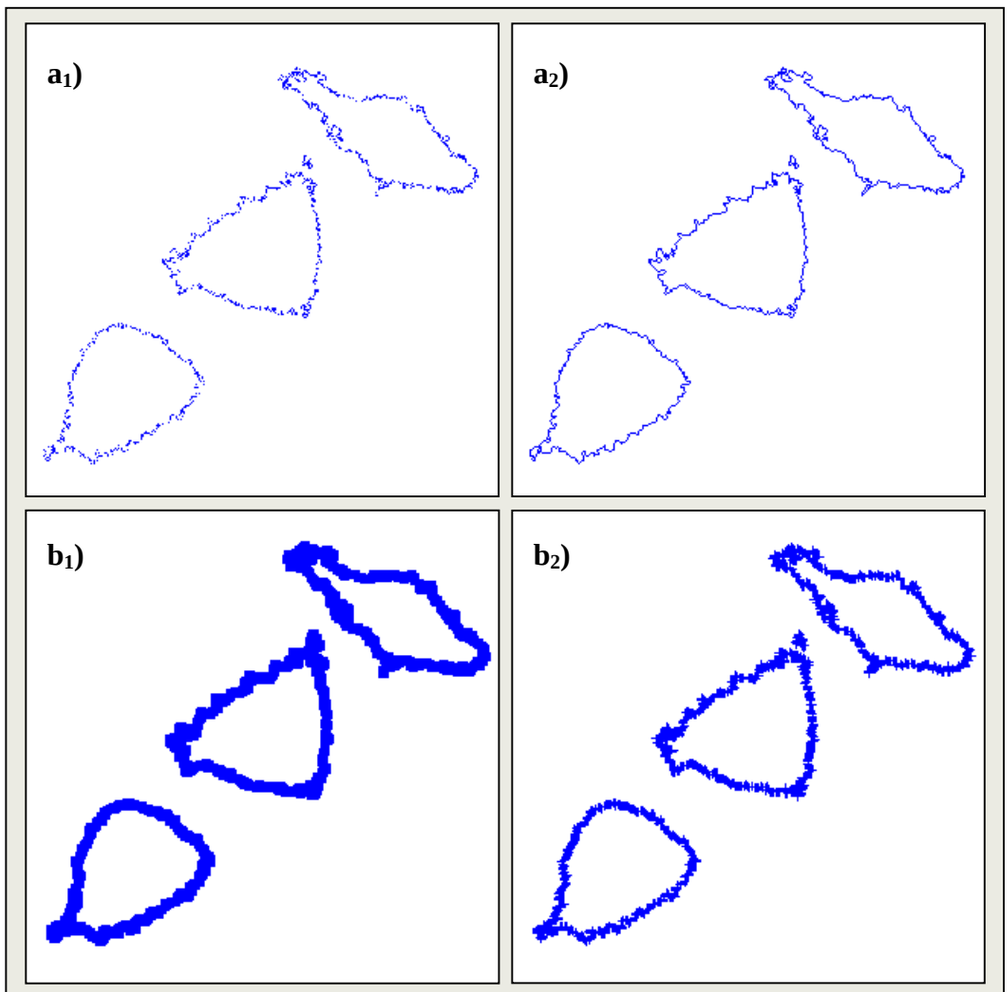


Fig. 4.8: Representación del parámetro **Thick** del objeto IDLgrPolygon. Thick dependerá del parámetro **Style,** si es dibujado en punto = 0 o línea = 1.
[a_1] Thick = 2 Style = 0, **[a_2]** Thick = 2 Style =1
[b_1] Thick = 6 Style = 0, **[b_2]** Thick = 6 Style =1

4.3.3 IDLgrVolume

El objeto IDLgrVolume representa una organización de un arreglo tridimensional de datos a un arreglo tridimensional de voxel de colores, y se proyecta en dos dimensiones. IDLgrVolume contiene parámetros predefinidos (Anexo A.3), de los cuales fueron parametrizados interactivamente los siguientes:

RGB_Table: Esta palabra clave es adaptada para cambiar el color al objeto. Es un arreglo de tipo byte de 256 x 3 elementos y puede representar un determinado color en un canal (R, G o B) o representar una LUT predefinida (combinación de los 3 canales) (Fig. 4.9).

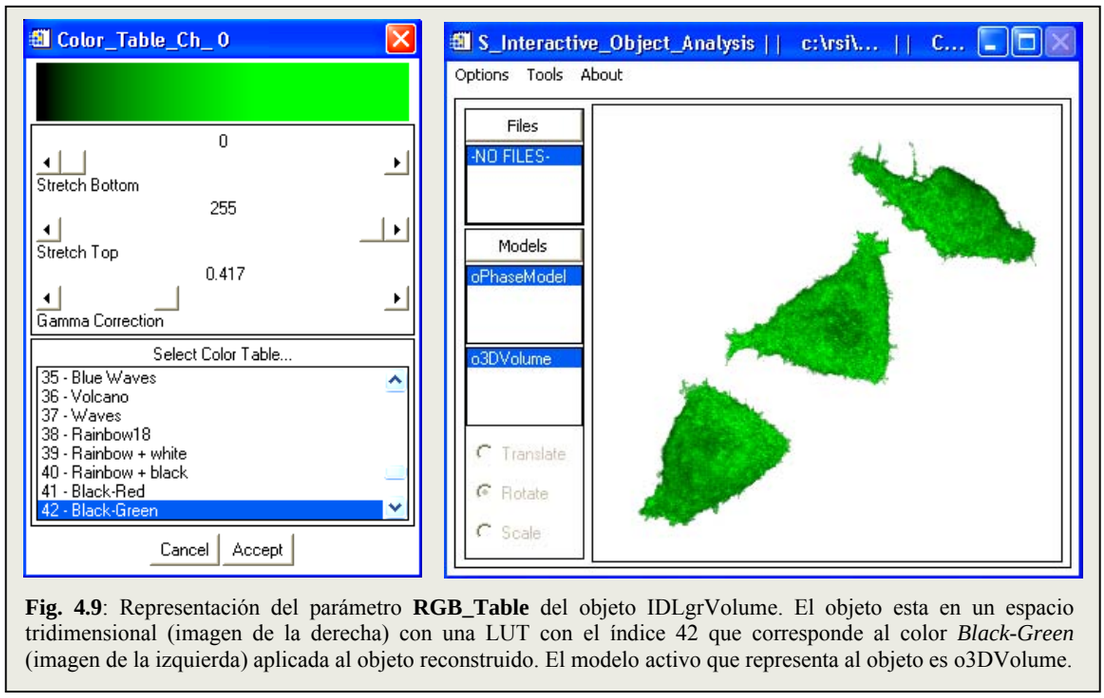


Fig. 4.9: Representación del parámetro **RGB_Table** del objeto IDLgrVolume. El objeto esta en un espacio tridimensional (imagen de la derecha) con una LUT con el índice 42 que corresponde al color *Black-Green* (imagen de la izquierda) aplicada al objeto reconstruido. El modelo activo que representa al objeto es o3DVolume.

Opacity_Table: Esta palabra clave es adaptada para cambiar la opacidad o nivel de transparencia (Cap. 3.6) al objeto volumen reconstruido (Fig. 4.10). Es un arreglo de tipo byte con dimensiones de 256. De esta forma es posible dar a cada voxel del objeto una determinada opacidad. Este parámetro ofrece una gran flexibilidad de RGB + α para cada voxel.

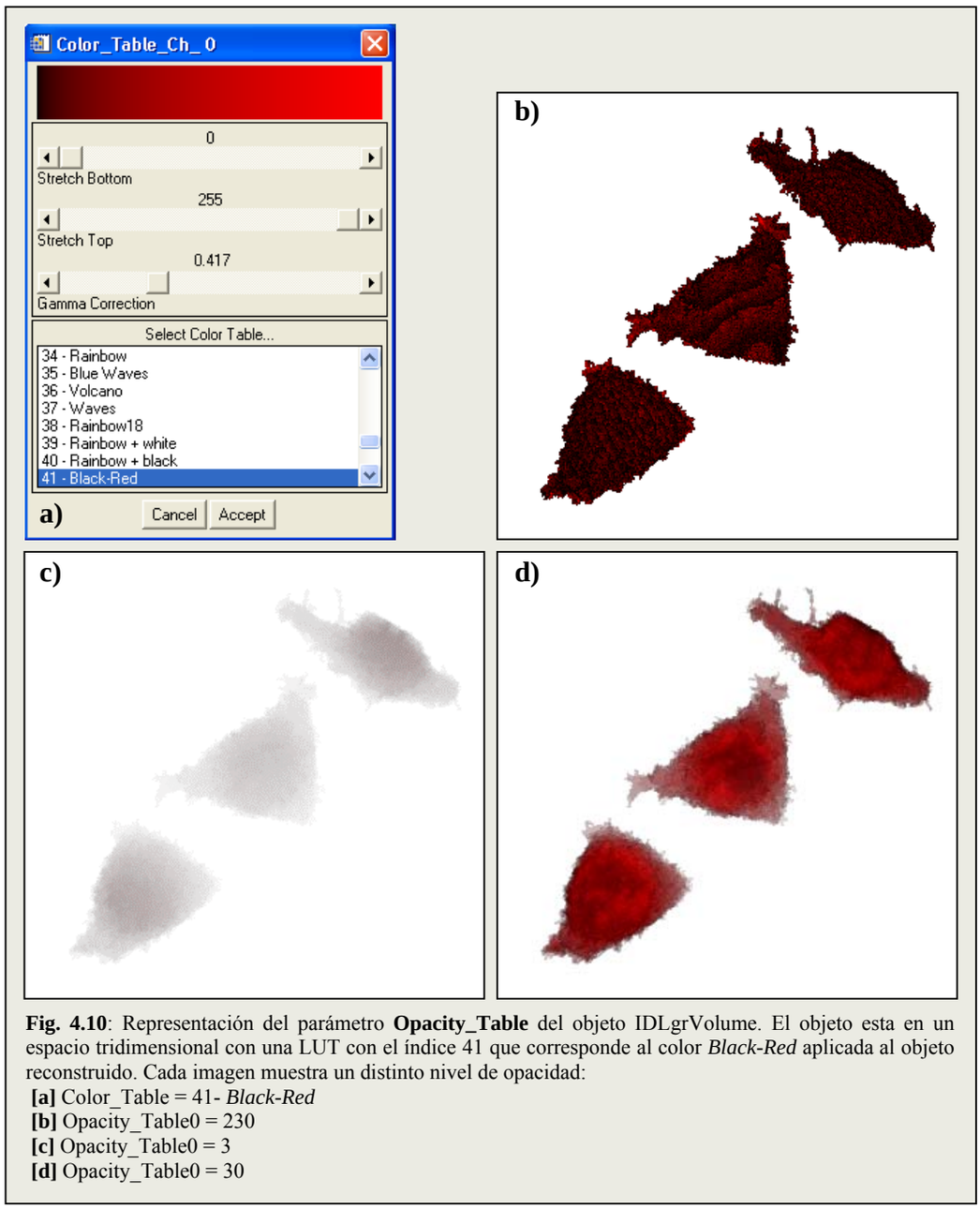


Fig. 4.10: Representación del parámetro **Opacity_Table** del objeto IDLgrVolume. El objeto esta en un espacio tridimensional con una LUT con el índice 41 que corresponde al color *Black-Red* aplicada al objeto reconstruido. Cada imagen muestra un distinto nivel de opacidad:
[a] Color_Table = 41- *Black-Red*
[b] Opacity_Table0 = 230
[c] Opacity_Table0 = 3
[d] Opacity_Table0 = 30

Cutting_Planes: Esta palabra clave es adaptada para generar cortes en los planos x, y, z, al objeto reconstruido (Fig. 4.11). Es un arreglo de tipo punto flotante con dimensiones (4, *n*) especificando los coeficientes de los *n* planos del corte. Los coeficientes son de la forma [{n_x, n_y, n_z, D},…] donde (n_x) X + (n_y) Y + (n_z) Z + D > 0, y (X, Y, Z) son las coordenadas del voxel.

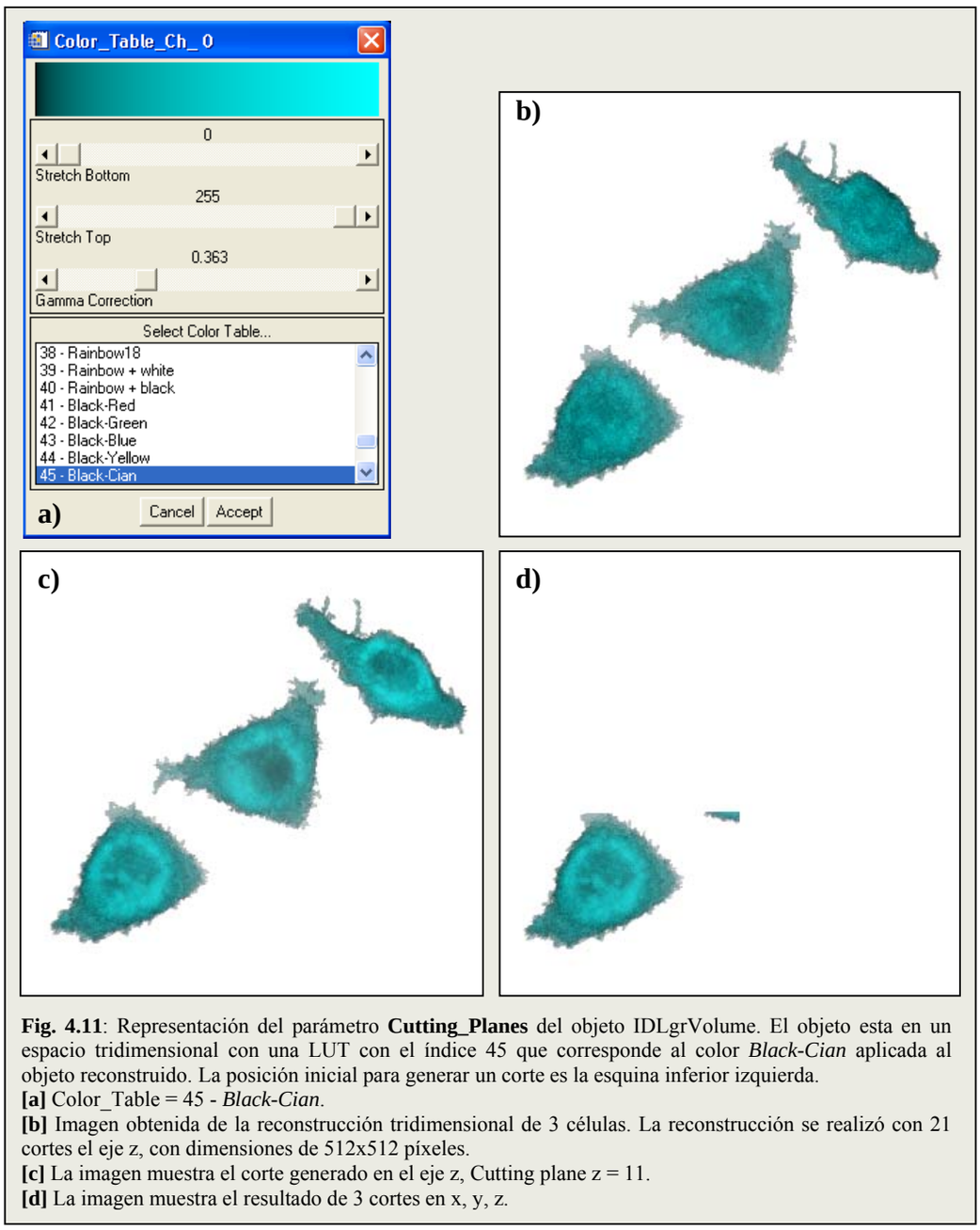


Fig. 4.11: Representación del parámetro **Cutting_Planes** del objeto IDLgrVolume. El objeto esta en un espacio tridimensional con una LUT con el índice 45 que corresponde al color *Black-Cian* aplicada al objeto reconstruido. La posición inicial para generar un corte es la esquina inferior izquierda.
[a] Color_Table = 45 - *Black-Cian*.
[b] Imagen obtenida de la reconstrucción tridimensional de 3 células. La reconstrucción se realizó con 21 cortes el eje z, con dimensiones de 512x512 píxeles.
[c] La imagen muestra el corte generado en el eje z, Cutting plane z = 11.
[d] La imagen muestra el resultado de 3 cortes en x, y, z.

Capítulo 5: Descripción de los Algoritmos Implementados

En el presente capítulo se da una descripción de los algoritmos implementados para la reconstrucción, visualización y parametrización de objetos biológicos en 2 y 3 dimensiones. La ubicación de los algoritmos se describe mediante un diagrama de flujo de datos (DFD) general del sistema (Fig. 5.1). Posteriormente se describen los algoritmos que generan los modelos en 2 y 3 dimensiones (Cap. 5.2) y se especifica el cálculo del volumen celular en forma explicita (Cap. 5.3). Al final se muestra el ambiente interactivo que permite el rápido acceso a los parámetros de los objetos gráficos implementados (Cap. 5.4).

5.1 Diagrama de Flujo para la Reconstrucción, Visualización y Parametrización

A través del DFD (Fig. 5.1), se describe el proceso de reconstrucción, visualización y parametrización. Una vez ingresadas las imágenes I(x,y,z,c,t) y aplicado los filtros de segmentación, se generaron diferentes modelos bidimensionales y tridimensionales con los objetos gráficos descritos en Cap. 4.2. Con los objetos IDLgrImage, IDLgrPolygon, IDLgrVolume se crearon los modelos de imagen, bordes y volumen, respectivamente, para ser visualizados en la ventana interactiva. Diversos parámetros son generados con estos modelos, de los cuales el cálculo del volumen celular realizado en este trabajo se describe más adelante (Cap. 5.3). Es importante señalar que los algoritmos implementados fueron realizados con la supervisión del tutor del proyecto, lo que permitió una mejor construcción y comprensión del sistema en general.

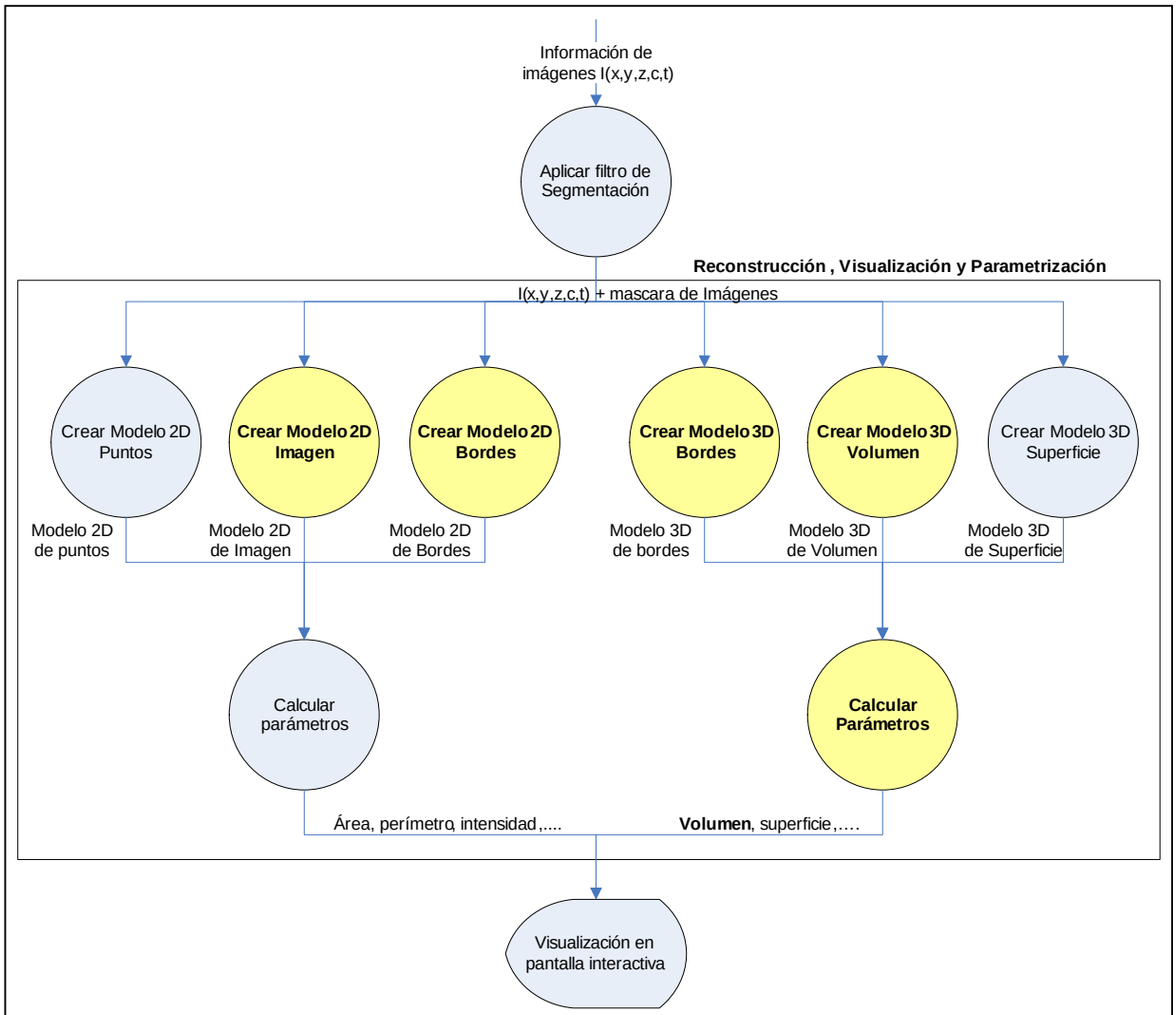


Figura 5.1: Diagrama de contexto (DFD) para segmentación, reconstrucción, visualización y parametrización.

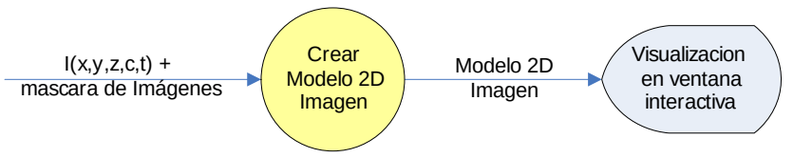
5.2 Algoritmos de los Modelos Generados

Los algoritmos a presentar se implementan en 2 clases principales, cuyo propósito es generar los modelos. Estas clases son: C_ROIGroupObject para los modelos bidimensionales, y C_ROI3DGroupObject para los modelos tridimensionales. Ambas clases son agregadas en una clase contenedora, llamada oObjectModel, y que es una instancia de IDLgrModel. Este contiene a los objetos gráficos atómicos, permitiendo aplicar transformaciones (rotación, escalamiento, y/o traslación) en forma interactiva (Cap.

4.1). A continuación se describen los modelos y el flujo de información de entrada y salida que generan.

5.2.1 Modelo de Imagen

El modelo de imagen tiene como objetivo representar una imagen es un espacio tridimensional. Este modelo fue generado a partir de una instancia de IDLgrImage (Cap. 4.2.1). Se considera el siguiente diagrama de contexto general del modelo.



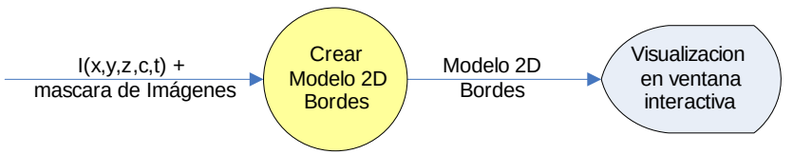
Los parámetros de entrada para crear el modelo de imagen se describe con el siguiente pseudocódigo.

1. inicio.
2. buscar posición actual de la imagen en **Z-Slice**.
3. buscar tabla de color de la imagen y asignar.
4. buscar máscara de la imagen.
5. buscar dimensiones de la imagen, que corresponden a las de la máscara.
6. asignar a la imagen los valores de RGB de la tabla de color.
7. declarar vectores **xp**, **yp** = [-0.5, 0.5].
8. buscar sistema de coordenadas **x**, **y**, **z**.
9. declarar parámetro **div** = (**dimen x** < **dimen y) / dimen x** < **dimen y**).
10. si (**dimen x** < **dimen y**) entonces **xp** = **xp*****div** – (0.5-0.5***div**) sino **yp** = **yp*****div** – (0.5-0.5***div**).
11. Crear objeto grafico atómico de tipo IDLgrImage de nombre **oTextureImage**, y con los parámetros: imagen; coordenadas **x**, **y**, **z**; **interleave** = 0; **loc** [0,0]; **hide** = 1.
12. Llamar método Add del objeto contenedor **oObjectModel** y agregar objeto grafico IDLgrPolygon con los parámetros: vértices **xp**, **yp**, **zp**; **texture_coord** = [[0,0],[1,0],[1,1],[0,1]]; **texture_map** = **oTextureImage**, **color** = [255, 255, 255].
13. fin.

El código fuente de este pseudocódigo se presenta en el anexo B.1.

5.2.2 Modelo de Bordes

Este modelo tiene como objetivo representar los bordes de las ROIs de cada imagen en un espacio tridimensional. Este modelo fue generado a partir de una instancia de IDLgrSurface (Cap. 4.2.2). Se considera el siguiente diagrama de contexto general del modelo.



Los parámetros de entrada para crear el modelo de bordes se describe con el siguiente pseudocódigo:

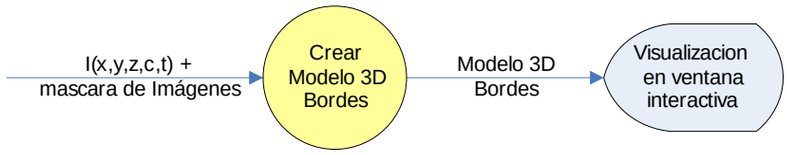
1. inicio.
2. inicializar color del borde, **color** = [0, 255, 0] para **r**, **g**, y **b**.
3. generar parámetro de **color** = [**r**, **g**, **b**] del borde como variable para **r**, **g** y **b** y limitar valores entre 0 y 255 para cada uno.
4. generar parámetro **Style** del borde como variable y limitar valores entre 0 y 2.
5. generar parámetro **LineStyle** del borde como variable y limitar valores entre 0 y 6.
6. generar parámetro **Thick** del borde como variable y limitar valores entre 1. y 10.
7. buscar posición actual de la imagen en **ZSlicePosition**.
8. para cada ROI dentro de la imagen llamar al método **addObjectBorderPolygon** y pasar los parámetros establecidos (**addObjectBorderPolygon** es la función que se encarga de instanciar al objeto grafico IDLgrPolygon con los parámetros).
9. fin.

El código fuente de este pseudocódigo se presenta en el anexo B.2.

5.2.3 Modelo de Bordes Tridimensional

Este modelo tiene como objetivo representar los bordes de las ROIs de un Z-Stack de imágenes en un espacio tridimensional. Este modelo fue generado a partir de una instancia

de IDLgrSurface (Cap. 4.2.2). Se considera el siguiente diagrama de contexto general del modelo.



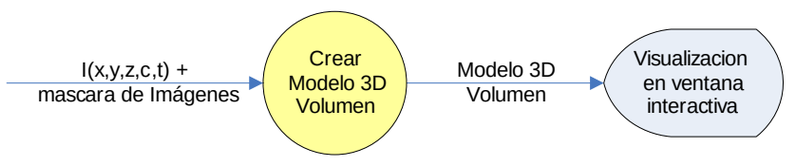
Los parámetros de entrada para crear el modelo de bordes tridimensional se describe con el siguiente pseudocódigo:

1. inicio.
2. inicializar color del borde, **color = [0, 255, 0]** para r, g, y b.
3. generar parámetro de **color = [r, g, b]** del borde como variable para **r**, **g** y **b** y limitar valores entre 0 y 255 para cada uno.
4. generar parámetro **Style** del borde como variable y limitar valores entre 0 y 2.
5. generar parámetro **LineStyle** del borde como variable y limitar valores entre 0 y 6.
6. generar parámetro **Thick** del borde como variable y limitar valores entre 1. y 10.
7. para cada ROI dentro de la imagen llamar al método **addObjectBorderPolygon** y pasar los parámetros establecidos (**addObjectBorderPolygon** es la función que se encarga de instanciar al objeto grafico IDLgrPolygon con los parámetros).
8. fin.

El código fuente de este pseudocódigo se presenta en el anexo B.3.

5.2.4 Modelo de Volumen

Este modelo tiene como objetivo representar el conjunto de las ROIs de un Z-Stack de imágenes en un espacio tridimensional. Este modelo fue generado a partir de una instancia de IDLgrVolume (Cap. 4.2.3). Se considera el siguiente diagrama de contexto general del modelo.



La información de entrada para crear el modelo de volumen se describe con el siguiente pseudocódigo.

1. inicio.
2. buscar dimensiones en píxel de imagen para **x** e **y**, y la cantidad de imágenes que contiene el Z-Stack para **z** y asignarlo a variable **xyzFramePixSize** como vector.

//conseguir 1er objeto y parámetros

3. generar **volData1** = byatrr (**xyzFramePixSize** [0], **xyzFramePixSize** [1], **xyzFrame-PixSize** [2]).
4. para cada objeto volumen donde está almacenada cada posición, asignar valor de intensidad correspondiente.
5. generar parámetro **rgb_table0** = **bytarr**(256,3)
6. generar parámetro **opacVect_0**. Considerar:
   - **i)** Si el valor es constante para todo el vector entonces **opacVect_0** = make_array (256,/byte, **value** = **opacVal**). opacVal es una variable que indica el valor constante del vector opacidad y toma valores entre 0 y 255.
   - **ii)** Si el valor es variable entonces **opacVect_0** = bytArr (256) + (***self.pVolState**).**opacValues** [0,*].
7. verificar valores de entrada para generar un corte en **x1**, **y1**, **z1**, cuyos valores estén dentro de las dimensiones del objeto.
8. generar parámetro **factorX1**, **factorY1**, **factorZ1** entre -1 y 1. Este factor dependerá del valor de entrada:
   - **i)** si es positivo el factor será -1.
   - **ii)** si es negativo, el factor será 1; esto permite generar cortes al objeto en ambos sentidos.

//conseguir 2do objeto y parámetros

9. buscar valor que contiene "**2nd Volume Object**" en lista de parámetros. Este valor puede ser '**channel**', '**time'** o '**cluster'**.
   - **i)** si valor es '**channel'** entonces buscar **strNum** = variable con el numero que indica el channel a agregar, buscar posición del cluster, buscar posición **time**, declarar **oROI3Dgroup** = **s_ISegM_GetROI3Dgroup** (**stack_tlb** = **stack_tlb**, **timePosition** = **timePosition**, **channelPosition** = **strNum**, **clusterPosition** = **clusterPosition**, **fileName** = **fileName**); **s_ISegM_GetROI3Dgroup** es una funcion que contienes los argumentos para obtener el grupo 3D.

ii) Si valor es '**time**', repetir 9i) con argumento **timePosition** = **strNum**.

iii) Si valor es '**cluster**', repetir 9i) con argumento **clusterPosition** = **strNum**.

10. generar **volData2** = bytArr (**xyzFramePixSize** [0], **xyzFramePixSize** [1], **xyzFrame-PixSize** [2]).

11. repetir paso del 4 al 9 para **volData2**.

//Instancia del objeto IDLgrVolume

12. buscar sistema de coordenadas **x**, **y**, **z**.

13. buscar valor que contiene "**Merge Volumes**" en lista de parámetros. Este valor puede ser '**off**' o '**on**'.

i) Si valor es '**off**' entonces llamar método **Add** del objeto contenedor **oObjectModel** y agregar objeto grafico IDLgrVolumen con los parámetros: **data0** = **volData1**,**opacity_table0** = **opacVect_0**, **rgb_table0** = **rgb_table0**, /**interpolate**, **uValue** = '**o3DVolumeModel1**', /**zero**, **Coord_conv** = **xCoord_conv**, **yCoord_conv** = **yCoord_conv**, **zCoord_conv** = **zCoord_conv**, **cutting_plane** = [[**factorX1**,0,0, **x1**], [0,**factorY1**,0, **y1**], [0,0,**factorZ1**, **z1**]], **ambient** = [255, 255, 255],**lighting_model**=1, repetir 19 para **volData2**.

ii) Si valor es 'on' entonces buscar valores de los parámetros **x1**, **y1**, **z1** que generan los cortes y asignarlos al **volData1**.

14. generar 3 matrices de acuerdo a LUT establecida (por defecto *Black-White Linear*) con la dimensión del objeto **volData1** y asignar las intensidades de este objeto los valores de la LUT. Generar los parámetros **volData_0** para **r**, **volData_1** para **g**, **volData_2** para **b**, de la forma:

volData_0 = (bytArr (256) + transpose((***self.pvolState**).**rgbValue**s[0,0,*]))[**volData1**]

volData_1 = (bytArr (256) + transpose((***self.pvolState**).**rgbValues**[0,1,*]))[**volData1**]

volData_2 = (bytArr (256) + transpose((***self.pvolState**).**rgbValues**[0,2,*]))[**volData1**]

15. generar parámetro **volData**_3 con igual dimensión de **volData1** para canal α (opacidad), considerar

i) Si valor es constante para todo el vector entonces **volData_3** = make_array(**xyzFramePixSize**[0], **xyzFramePixSize**[1], **xyzFramePix Size**[2], /byte, **value** = **opacVal**). opacVal es una variable que indica el valor constante del vector opacidad y toma valores entre 0 y 255.

ii) Si valor es variable entonces **volData**_3 = (bytArr (256) + transpose ((***self.pvolState**).**opacValues** [0,*])) [**volData1**].

16. buscar valores de los parámetros **x2**, **y2**, **z2** que generan los cortes y asignarlos al **volData2**.

17. repetir 22 con **volData2** e incluir **volData_0**, **volData_1**, **volData_2** de la forma:

volData_0 = byte ((fix (temporary (**volData_0**) + (bytArr (256) + transpose ((***self.pvolState**).**rgbValues** [1,0,*])) [**volData2**])) < 255)

volData_1 = byte ((fix (temporary (volData_1) + (bytArr (256) + transpose ((***self.pvolState**).**rgbValues** [1,1,*])) [**volData2**])) < 255)

volData_2 = byte ((fix (temporary (**volData**_2) + (bytArr (256) + transpose ((***self.pvolState**).**rgbValues** [1,2,*])) [**volData2**])) < 255)

18. repetir 23 con parámetro **volData_3** con igual dimensión de **volData2** se consideran

i) Si valor es constante para todo el vector entonces **volData2_3** = make_array (**xyzFramePixSize** [0], **xyzFramePixSize** [1], **xyzFramePix Size** [2], /byte, **value** = **opacVal**). opacVal es una variable que indica el valor constante del vector opacidad y toma valores entre 0 y 255.

ii) Si valor es variable entonces **volData**_3 = temporary (**volData_3**) > **volData2_3**.

19. llamar método Add del objeto contenedor oObjectModel e instanciar objeto gráfico IDLgrVolumen con los parámetros: **volume_select** = 2, **data0** = **volData_0**, **data1** = **volData_1**, **data2** = **volData_2**, **data3** = **volData_3**, /**interpolate**, **uValue** = '**o3DVolumeModel1**', /**zero**, **xCoord_conv** = **xCoord_conv**, **yCoord_conv** = **yCoord_conv**, **zCoord_conv** = **zCoord_conv**, **ambient** = [255, 255, 255], **lighting_model** = 1.

20. fin.

El código fuente de este pseudocódigo se presenta en el anexo B.4.

5.3 Cálculo del Volumen Celular

Para describir el proceso del cálculo de volumen celular, se utilizan como ejemplo 2 imágenes consecutivas tomadas en diferentes planos en el eje z de una célula necrótica (Fig. 5.2a/c). Aplicando filtros para la segmentación (Cap. 2.3) se obtienen las respectivas máscaras (Fig. 5.2b/d).

Para cada píxel con dimensiones $\Delta x \Delta y$ se genera un voxel $\Delta x \Delta y \Delta z$. Un voxel o elemento de volumen es la unidad mínima de un objeto tridimensional y se determina con Δx, Δy, y Δz. El volumen de un voxel se define como $v = \Delta x \Delta y \Delta z$. La altura del voxel ($\Delta z$) se define por la distancia entre cortes (Z-Slices) durante la adquisición de la imagen tridimensional. Para unir cortes consecutivos en la reconstrucción (Fig. 5.2c), el centro de cada voxel se localiza en la posición del corte, asegurando de este modo la conectividad entre cortes.

El volumen de un objeto entero (V_{total}) es calculado con la suma de los voxeles del objeto reconstruido (v_i).

$$V_{total} = \sum_{i=1}^{n} v_i$$

n es el número total de voxeles en el objeto.

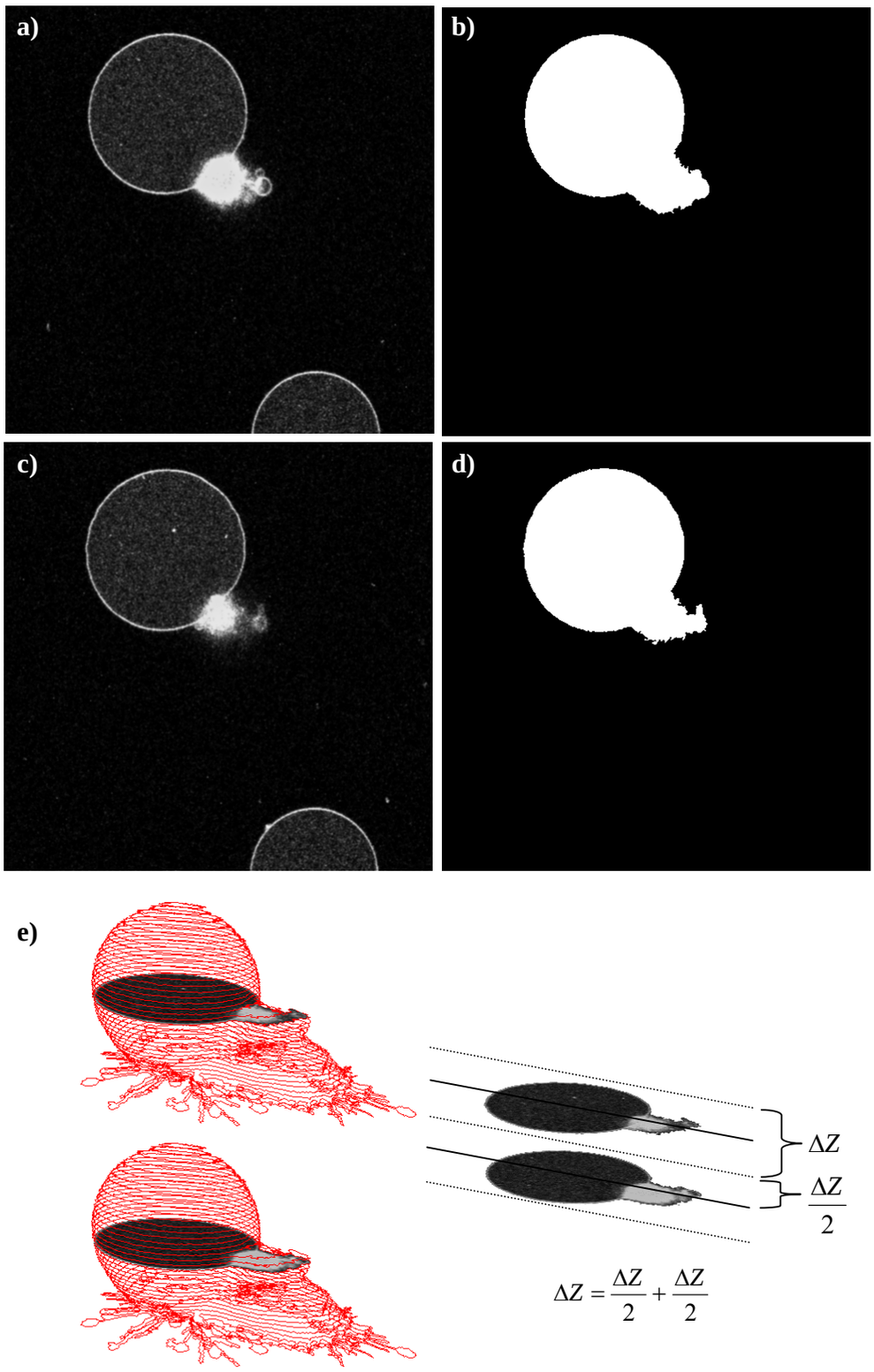


Figura 5.2: Imágenes consecutivas con segmentación y filtros morfológicos de una célula necrótica.
[a] y **[b]** Imágenes originales tomadas en el eje z con un microscopio confocal de fluorescencia.
[c] y **[d]** Resultado de aplicación de segmentación y filtros morfológicos.
[e] Modelo de bordes tridimensional para representar la conectividad entre cortes.

A través de un modelo de superficie (Ver Fig. 5.3) se puede visualizar una reconstrucción tridimensional de todos los voxeles que se generan con los cortes a través de la célula.

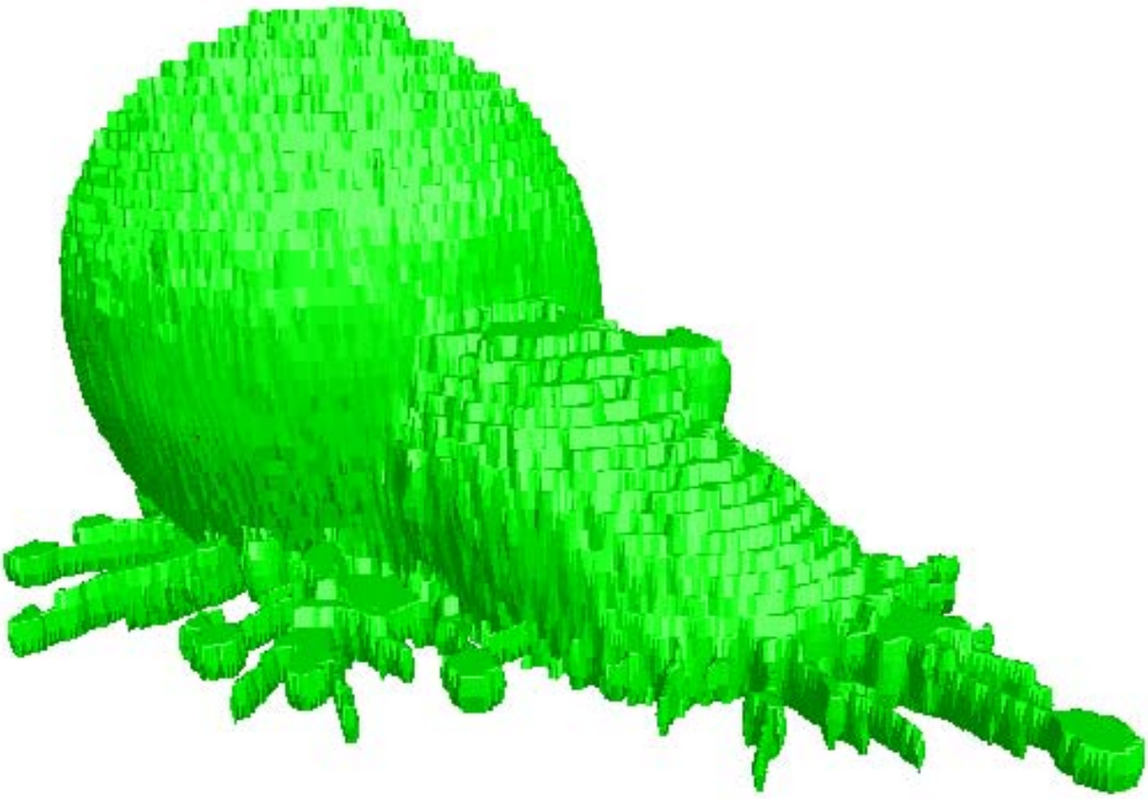


Figura 5.3: Reconstrucción tridimensional de un volumen celular con un modelo de superficie.

La información para calcular el volumen se describe con el siguiente pseudocódigo.

1. inicio.
2. asignar a variable **nObjects** el número de objetos que contiene la clase sROI3DGroupObject.

//hacer bucle para determinar el tamaño de cada objeto [voxel]

3. asignar a variable **paramVect** el número de objetos (**nObjects**) como vector.
4. asignar en cada posición del vector **paramVect**, el número de voxeles de los ROIs para cada objeto.

//Volumen 3D [voxel]

5. asignar en cada posición del vector **paramVect** el número total de voxeles de cada ROI.

//Volumen 3D [x^3]

6. asignar a variable **pParamStruct** el puntero a la lista de parámetros que contiene la función **getpParamStruct** () en la clase C_sROI3DGroupObject.
7. asignar a **xSizePerPixel** el valor del tamaño total en **x** de la imagen (valor entero en µm) dividido por el tamaño total en píxel (ancho de la imagen).
8. asignar a **ySizePerPixel** el valor del tamaño total en **y** de la imagen (valor entero en µm) dividido por el tamaño total en píxel (alto de la imagen).
9. asignar a **zSizePerPixel** el valor del tamaño total del alto de Z-Stack (valor entero en µm) dividido por el tamaño en píxel (número de cortes) para **z**.
10. multiplicar (**xSizePerPixel*ySizePerPixel*zSizePerPixel**) por el vector **paramVect**.
11. fin.

5.4 Ambiente Interactivo

Para la descripción del ambiente interactivo se presenta inicialmente una descripción general de las ventanas iniciales de la aplicación (Fig.5.4, Fig. 5.5 y Fig. 5.6). Luego se presentan las ventanas de controles que formarán parte de este trabajo, indicando las funcionalidades implementadas que permiten acceder a los atributos de los objetos visualizados (Fig. 5.7).

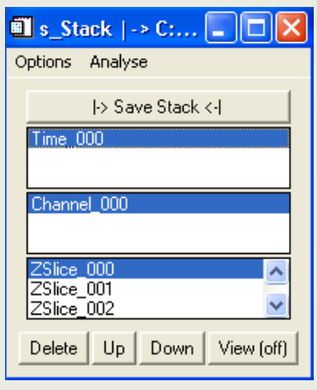


Fig. 5.4: Ventana inicial con la información del Z-Stack.

Una vez cargadas las imágenes confocales de fluorescencia desde el menú '*Options*' en la ventana inicial (Fig. 5.4) de la aplicación, se activa de este menú la opción '*Set Stack Information*' que contiene los valores de inicialización de los algoritmos (Fig. 5.5). Se ingresan manualmente los valores reales de ***x-Size***, ***y-Size*** y ***z-Interval***. Los otros valores son leídos automáticamente.

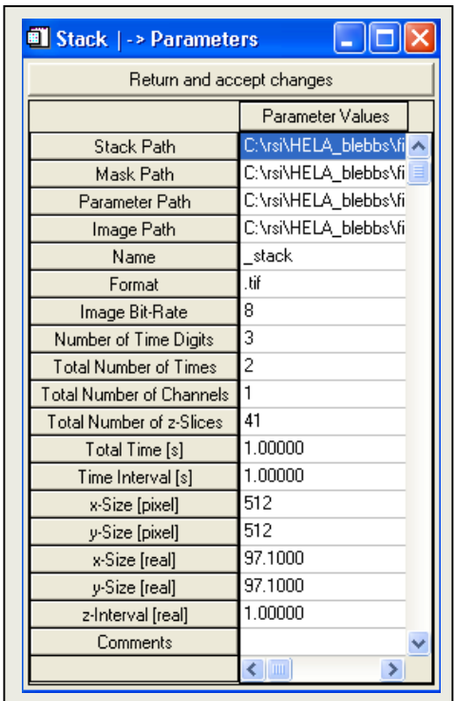


Fig. 5.5: Ventana con los valores de inicialización de los parámetros.

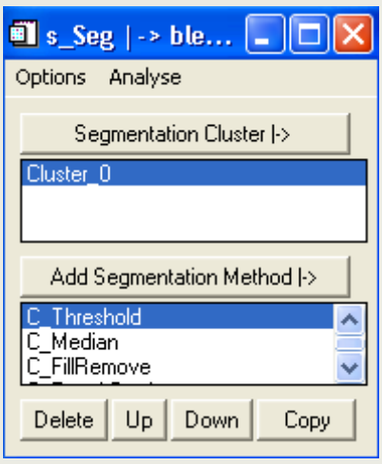


Fig. 5.6: Ventana con los métodos de segmentación.

Del menú '*Analyse*' de la ventana inicial se activa la ventana que contiene los métodos de segmentación (Fig. 5.6). En el menú '*Analyse*' de la Fig. 5.6 se activa la ventana de los ROIs (Ver Fig. 5.7a), que a continuación se describe para indicar las intervenciones realizadas y el flujo que se genera a partir de ella.

La Fig.5.7 en general muestra la interacción y el flujo que se genera a partir de la ventana de los ROIs y el rápido acceso a los parámetros de los objetos visualizados en la ventana interactiva. La Fig. 5.7a es la ventana principal de los ROIs, que contiene los modelos oROI2DGroup y oROI3DGroup. En esta ventana se agregan los diversos parámetros de cada modelo generado. En el proyecto se implementó el parámetro '*3D Object Volume*', y los sub parámetros '*3D Volume (Voxel)*' y '*3D volume* (x^3)'. En el menú '*Options*' se agregó la opción '*Set Group Object Information*' que contiene la ventana con la lista de parámetros de los modelos tridimensionales (Fig. 5.7b) y el botón '*Visualization*' (Fig. 5.7c) que permite abrir las ventanas '*Color Table*' (Fig. 5.7e) y '*Opacity Table*' (Fig. 5.7d) para manejar las LUT y la opacidad. En el menu '*Analyse*' de la ventana de los ROIs se encuentran los modelos bidimensionales y tridimensionales implemantados que pueden ser agregados a la ventana interactiva.

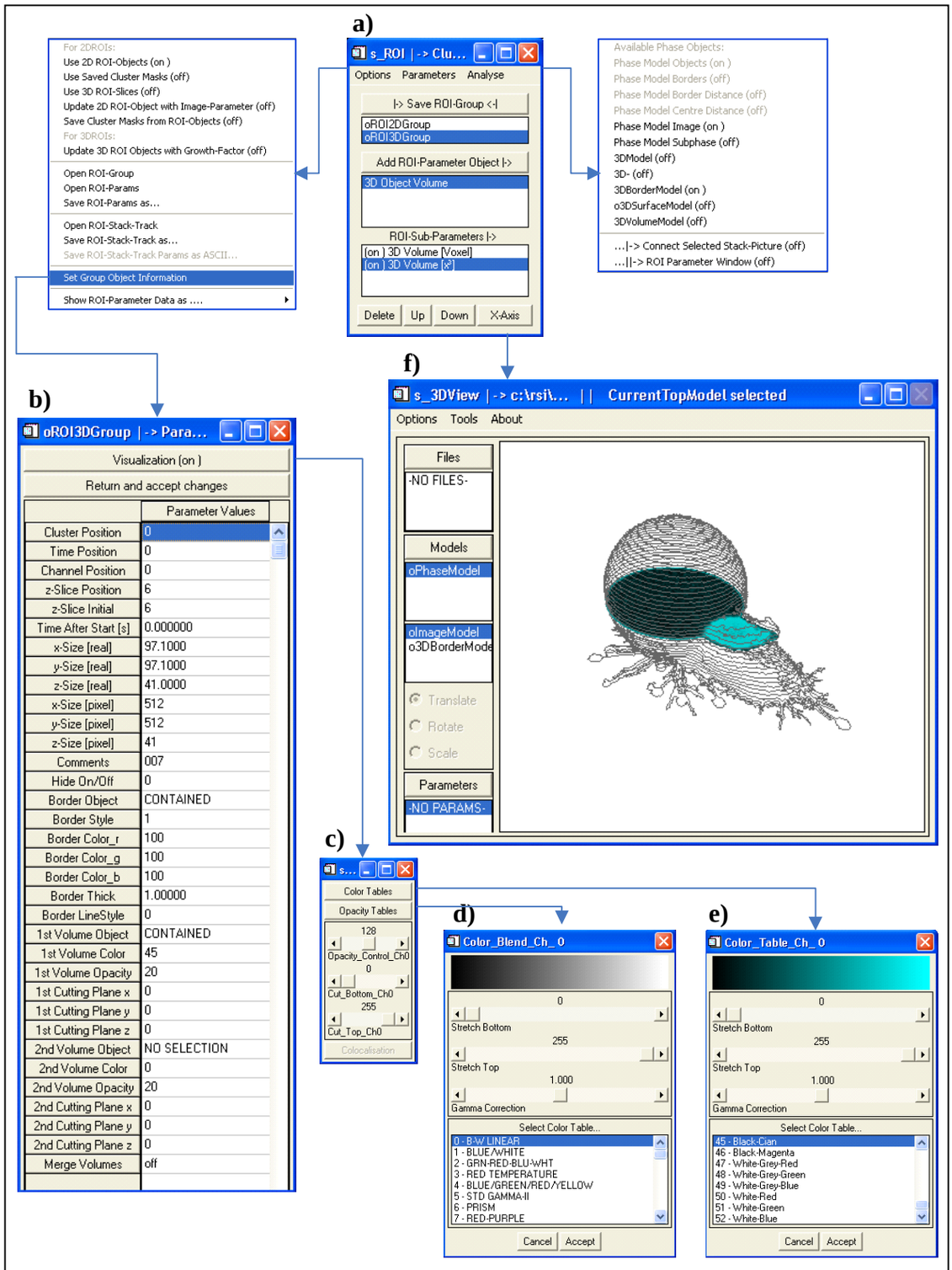


Figura 5.7: Ambiente interactivo para el acceso a los parámetros de los objetos visualizados.

Capítulo 6: Aplicaciones de Objetos Gráficos Tridimensionales

En el presente capítulo se presentan aplicaciones de reconstrucciones tridimensionales realizadas con los objetos introducidos en el capítulo 4. Se calibra la distancia entre cortes en el eje z (Z-Slices) (Cap. 6.1), se mide volumen celular (Cap. 6.2) y se realizan ejemplos de reconstrucciones en uno y dos canales de fluorescencia (Cap. 6.3 y 6.4).

6.1 Calibración de la Distancia Optima entre Cortes en el Eje z (Z-Slices)

En la siguiente aplicación se calibra la distancia entre cortes en el eje z (altura) a través de reconstrucciones tridimensionales. El objetivo de este experimento es establecer un criterio para la distancia óptima entre cortes en el eje z que representa un balance entre el número de cortes, la medición del volumen celular y la determinación de la morfología, sin dañar a la célula a través del tiempo. Para la calibración se utilizó como modelo una célula muerta, debido a que ésta presenta una morfología invariable. En estas condiciones se aplicaron múltiples cortes en eje z, generando inicialmente 51 cortes con distancia z de 0,3 μm para una reconstrucción con alto detalle morfológico, obteniendo un volumen exacto de 1,78 pl (Fig. 6.1A). La segunda columna de reconstrucciones en B fue realizada con 19 cortes, con una distancia de 0,8 μm en el eje z. La tercera columna de reconstrucciones en C fue realizada con 15 cortes, con una distancia de 1 μm en el eje z. La cuarta columna de reconstrucciones en D fue realizada con 13 cortes con una distancia de 1,3 μm en el eje z, y para la última columna de reconstrucciones en E se realizaron 12 cortes en el eje z con una distancia de 1,6 μm. Los resultados de la calibración de acuerdo a un análisis de varianza (Anexo C), determinan que la población E con una distancia de 1,6 μm entre cortes, es significativamente diferente a las otras poblaciones ($p<0,05$). El volumen 2,22 pl es alejado del valor original de 1,78 pl y presenta una morfología distorsionada. Debido a que con una distancia grande entre cortes en el eje z, se necesitan menos cortes para abarcar toda la

estructura, teniéndose menos información para reconstruir el objeto y en consecuencia una pérdida significativa en la resolución del eje z.

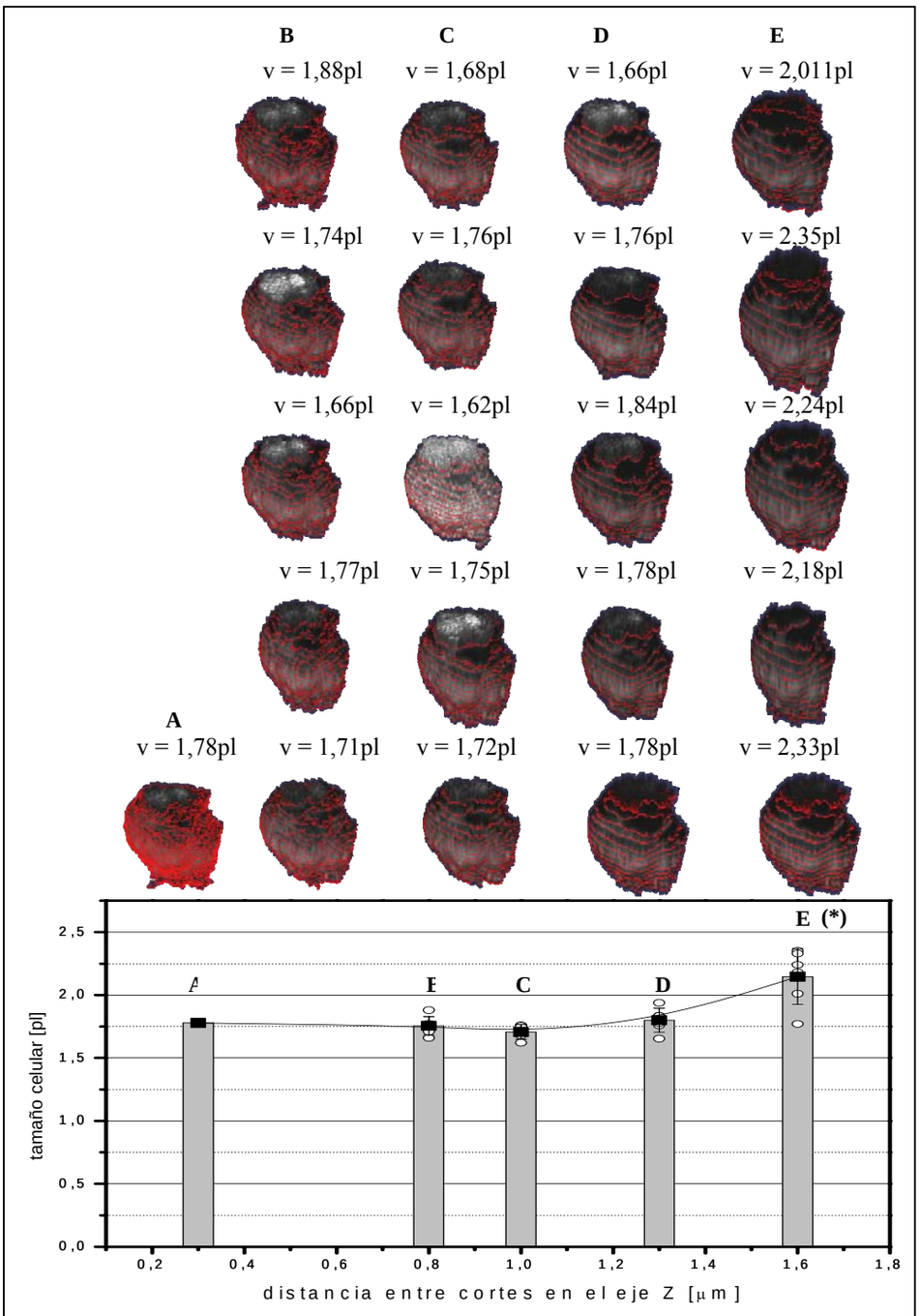


Fig. 6.1: Calibración de la distancia entre cortes en el eje z para determinar una balance entre el número de cortes, la medición del volumen celular y la determinación de la morfología sin dañar a la célula. En la parte superior de la figura se ilustran las reconstrucciones de una célula HeLa muerta sometida a diferentes cortes, con distancia entre cortes en el eje z de 0,3µm, 0,8µm, 1µm, 1,3µm, y 1,6µm respectivamente con su volumen propiamente tal. En la parte inferior hay mediciones individuales (o), promedios (■) y desviaciones estándar (⊥). El grafico describe la relación entre el promedio del volumen celular de cada población (B, C, D, E) vs. distancia entre cortes en el eje z. De acuerdo a un análisis de varianza se determinó que la población E es significativamente diferente a las otras poblaciones ($p<0,05$)(Anexo C).

Comparando los resultados de la determinación del volumen celular con la resolución de la morfología celular se puede concluir que con una distancia entre cortes en el eje z de 1.0 μm se logra un balance óptimo entre las características mencionadas.

En un experimento adicional (Mon05) se mostró que las células HeLa se podían observar sin problemas durante mucho tiempo, aplicando cortes con una distancia de 1.0 μm en el eje z.

6.2 Medición de Volumen en Células HeLa

La siguiente aplicación consta de 2 condiciones experimentales aplicada a células tipo HeLa en perfectas condiciones de cultivo. La membrana plasmática de las células fue marcada con el fluoróforo $DiIC_{18}$. En los 2 experimentos que se describen a continuación se mide el volumen celular.

En el 1^{er} experimento a través de una muestra celular se obtuvieron diversos Z-Stack de células individuales y grupales en condiciones normales. Se realizaron reconstrucciones tridimensionales con el objetivo de obtener una representación cualitativa de su morfología y cuantitativa de su volumen (ver Fig. 6.2.).

En el 2^{do} experimento se expuso la muestra celular a 32mM de H_2O_2 (agua oxigenada) durante 1 hora. La *necrosis* celular se manifiesta pocos minutos después de su aplicación. El peróxido induce a un desequilibrio en el control de volumen celular que a su vez conduce a la formación de vesículas o '*blebs*' en la membrana plasmática [Bar01]. En estas condiciones se obtuvieron diversos Z-Stack de células individuales y grupales. Se realizaron reconstrucciones tridimensionales con el objetivo de obtener una representación cualitativa de su morfología y cuantitativa de su volumen (ver Fig. 6.3).

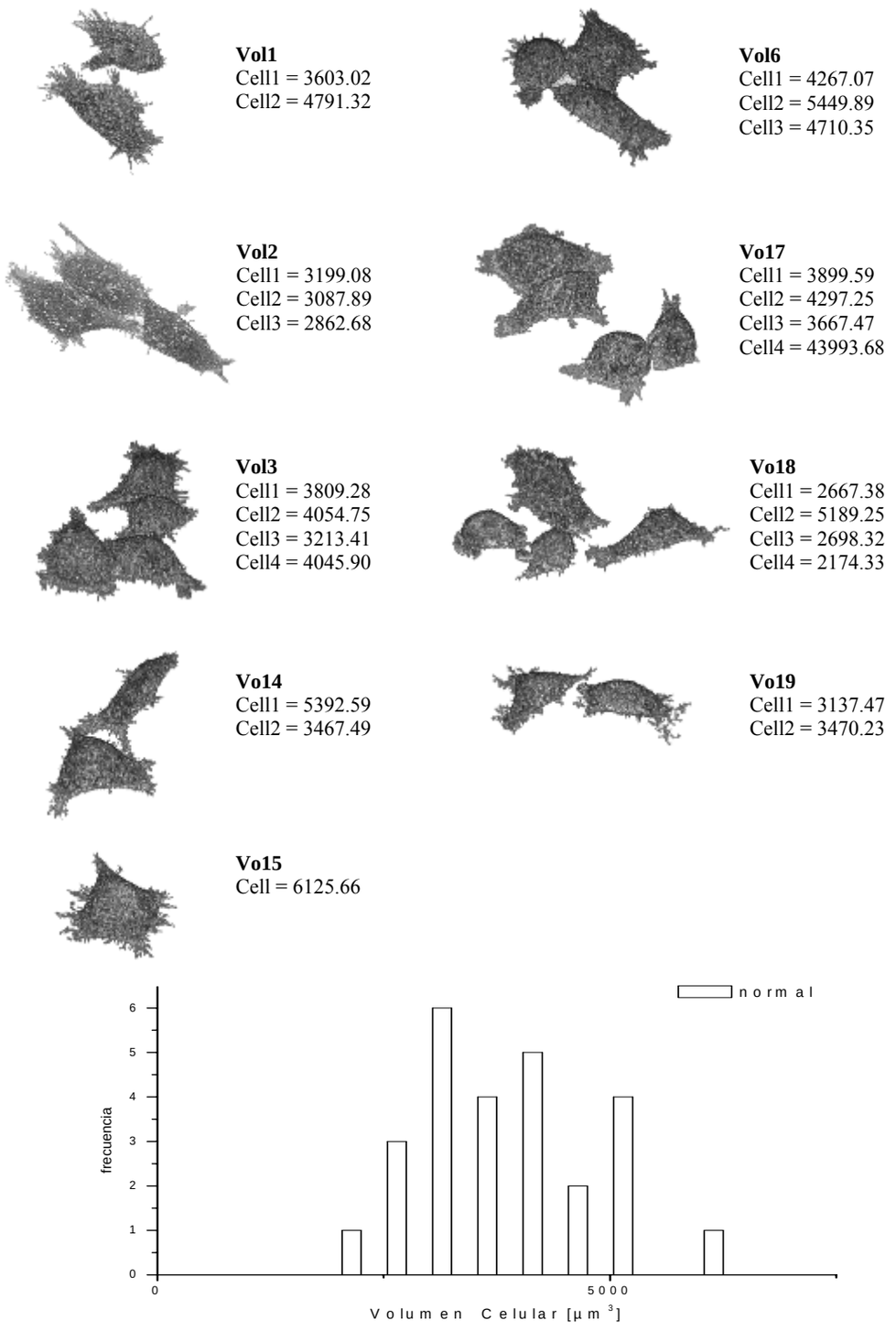
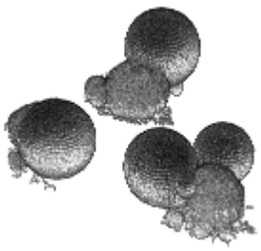


Fig. 6.9: Resultado de aplicar un '*Merge Volumes*' reconstrucciones en 2 canales con un corte en el eje z. Los sectores amarillos indican superposición entre canales.

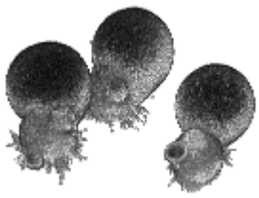
Bleb0
cell1 = 19867.2



Bleb1
Cell1 = 12152.4
Cell2 = 18837.5
Cell3 = 18145.8

Bleb2
Cell1 = 24168.9
Cell2 = 22113.8

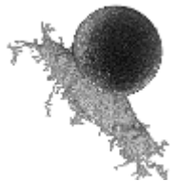
Bleb3
Cell1 = 13080.3
Cell2 = 9086.87



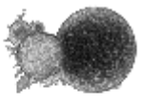
Blebb4
Cell1 = 18430.5
Cell2 = 15429.4
Cell3 = 12493.5

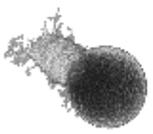
Bleb5
Cell1 = 10451.0
Cell2 = 11835.4

Bleb6
Cell1 = 19879.2
Cell2 = 21712.5

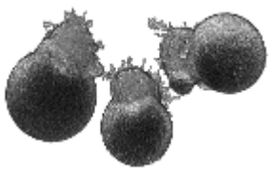


Bleb7
Cell1 = 19743.9

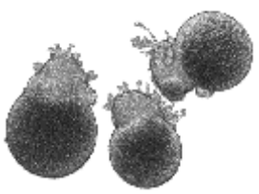




Bleb8
Cell1 = 13784.3
Cell2 = 15736.3

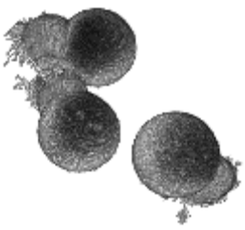


Bleb9
Cell1 = 17587.6
Cell2 = 12552.6
Cell3 = 10638.1



Bleb10
Cell1 = 18157.2
Cell2 = 13031.3
Cell3 = 10984.5

Bleb11
Cell1 = 23198.6



Bleb12
Cell1 = 3137.47
Cell2 = 3470.23

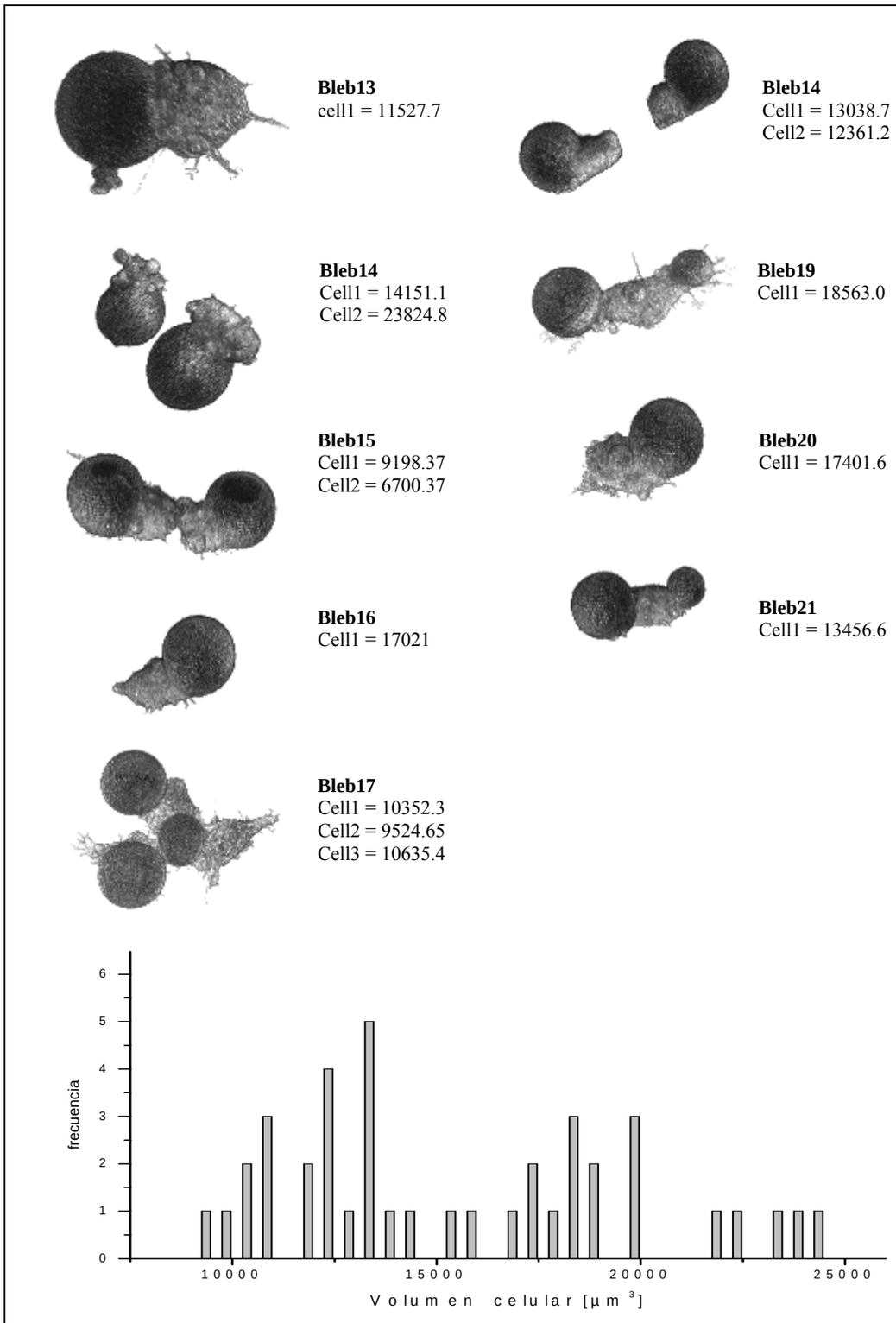


Fig. 6.3: Medición del volumen en células HeLa después de 1 hora de exposición a 32 mM H_2O_2. En la parte superior de la figura se ilustran las reconstrucciones tridimensionales con sus respectivos volúmenes. En el gráfico inferior se representa la relación entre el número de células a un cierto volumen.

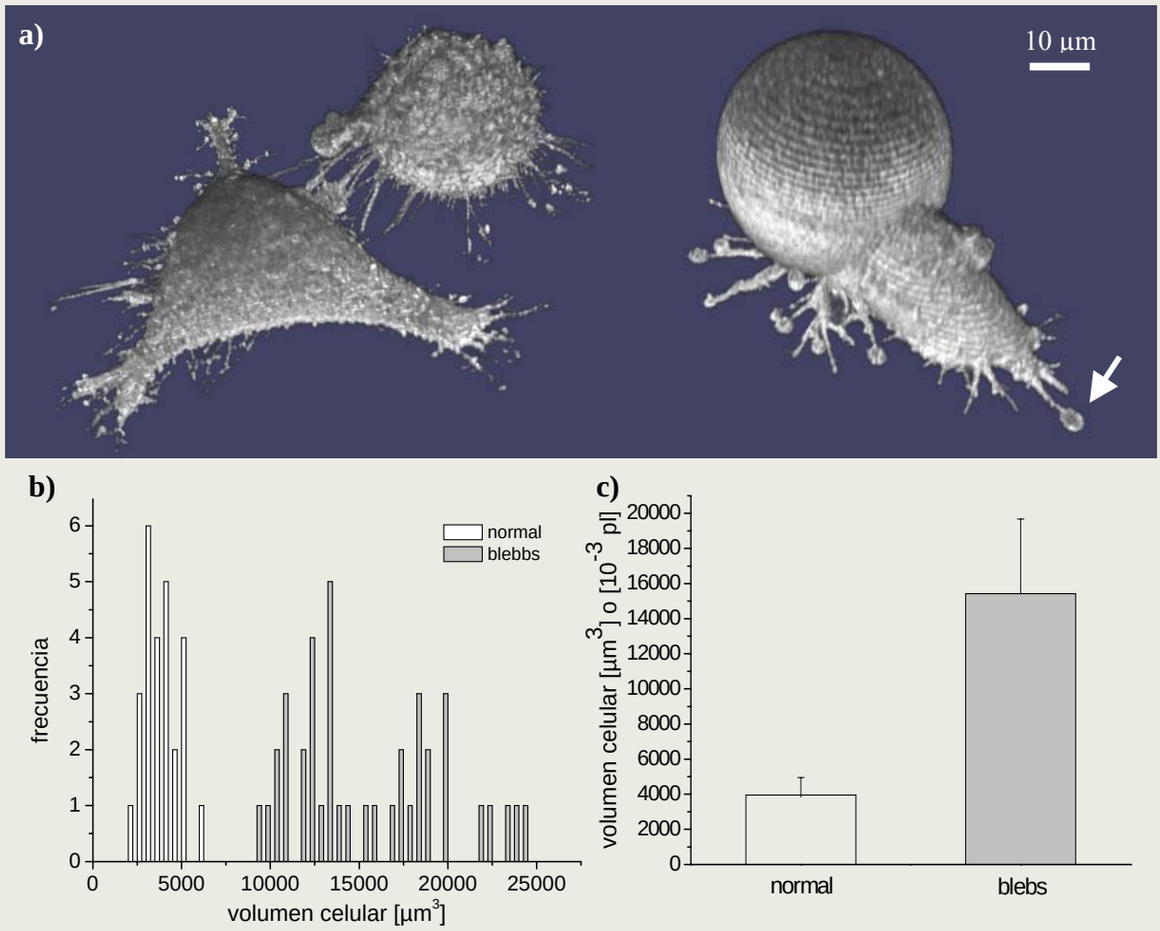


Fig. 6.4: Resultados de volúmenes de células HeLa en 2 condiciones experimentales.
[a] Reconstrucción tridimensional de dos células Hela representativas en el comienzo del experimento (izquierda), y 1h después de añadir 32 mM H_2O_2 (derecha).
[b] Distribución de Frecuencia de los volúmenes de células HeLa antes (columnas blancas) y 1h después de añadir 32 mM H_2O_2 (columnas grises).
[c] Valores promedios del volumen celular v = 3956± 992(sd) con su correspondiente desviación estándar para células normales (n = 26), o v = 15424± 671(sd) para células tratadas con H_2O_2(n = 38).

Los resultados de la aplicación experimental con las reconstrucciones tridimensionales, muestran que los volúmenes de células individuales pueden ser derivados con gran precisión. Cambios sutiles en la morfología celular, como la formación de pequeños *blebs* en diferentes posiciones de las filopodias celulares (ver flecha en Fig. 6.4), pueden ser identificados visualmente y cuantificados volumétricamente.

6.3 Ejemplo de Reconstrucción Tridimensional en 1 Canal de Fluorescencia

En el siguiente ejemplo se realiza una reconstrucción tridimensional en un canal de fluorescencia con 56 cortes en el eje z (Fig. 6.5). Las membranas de las células fueron marcadas con el fluoróforo $DiIC_{18}$. Para la reconstrucción se realizó un pretratamiento de deconvolución a la secuencia de imágenes.

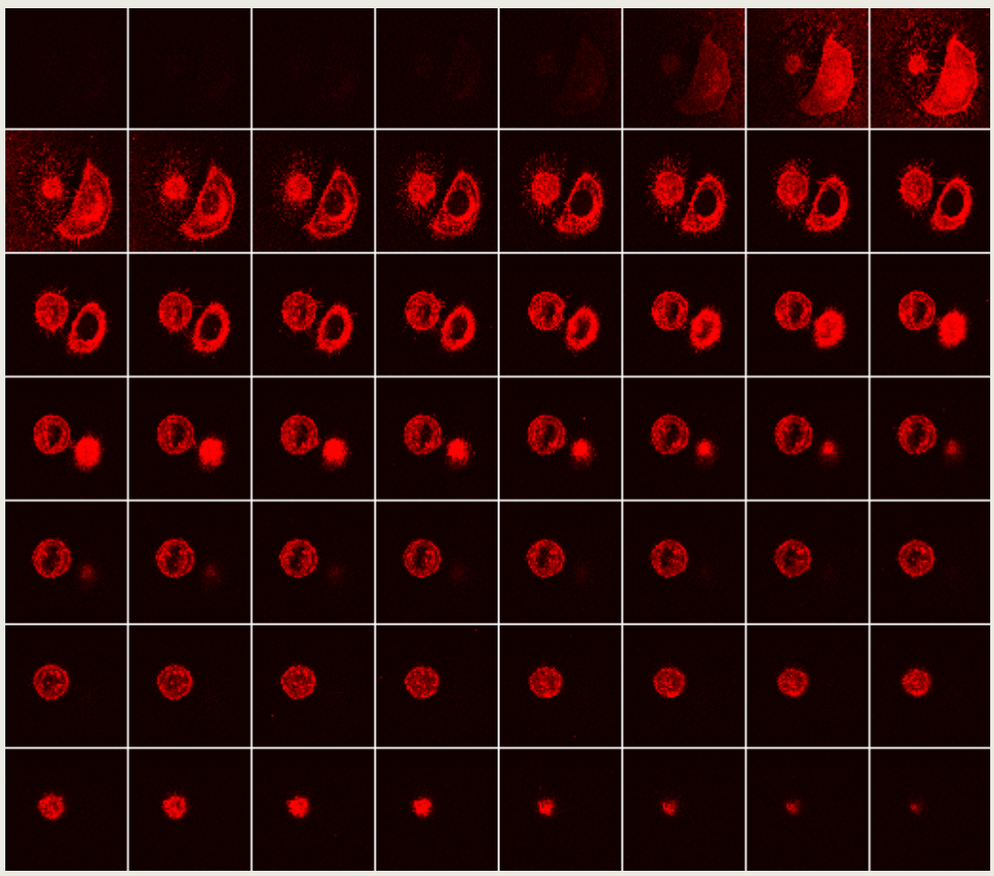


Fig. 6.5: Z-Stack de 56 cortes en el eje z con un microscopio confocal de fluorescencia. La imagen están tomadas en un canal de fluorescencia con el marcador de membrana $DiIC_{18}$, $I(x,y,z,c,t) = I(512,512,56,1,1) \in [0, 255]$.

Se observa en la Fig. 6.6 que la célula superior de las reconstrucciones tridimensionales esta preparando la división celular (*mitosis*), mientras que la célula inferior se encuentra en la interfase. Ambas células están unidas mediante varios nanotubos. Estos nanotubos fueron descubiertos por primera vez durante el tiempo en el que se realizo este trabajo [Rus04]. A

través de estos nanotubos, las células intercambian diferentes organelos o componentes de las membranas lipídicas.

El detalle de la morfología y las prolongaciones (*filopodias*) se destacan en ambas células gracias a los parámetros de la LUT y los altos valores de opacidad aplicado al objeto reconstruido (Fig. 6.6a). Se deja translucir el objeto (bajos valores de opacidad), se puede observar en detalle el núcleo celular, así como pequeñas estructuras perinucleares en el interior de la célula (Fig. 6.6b).

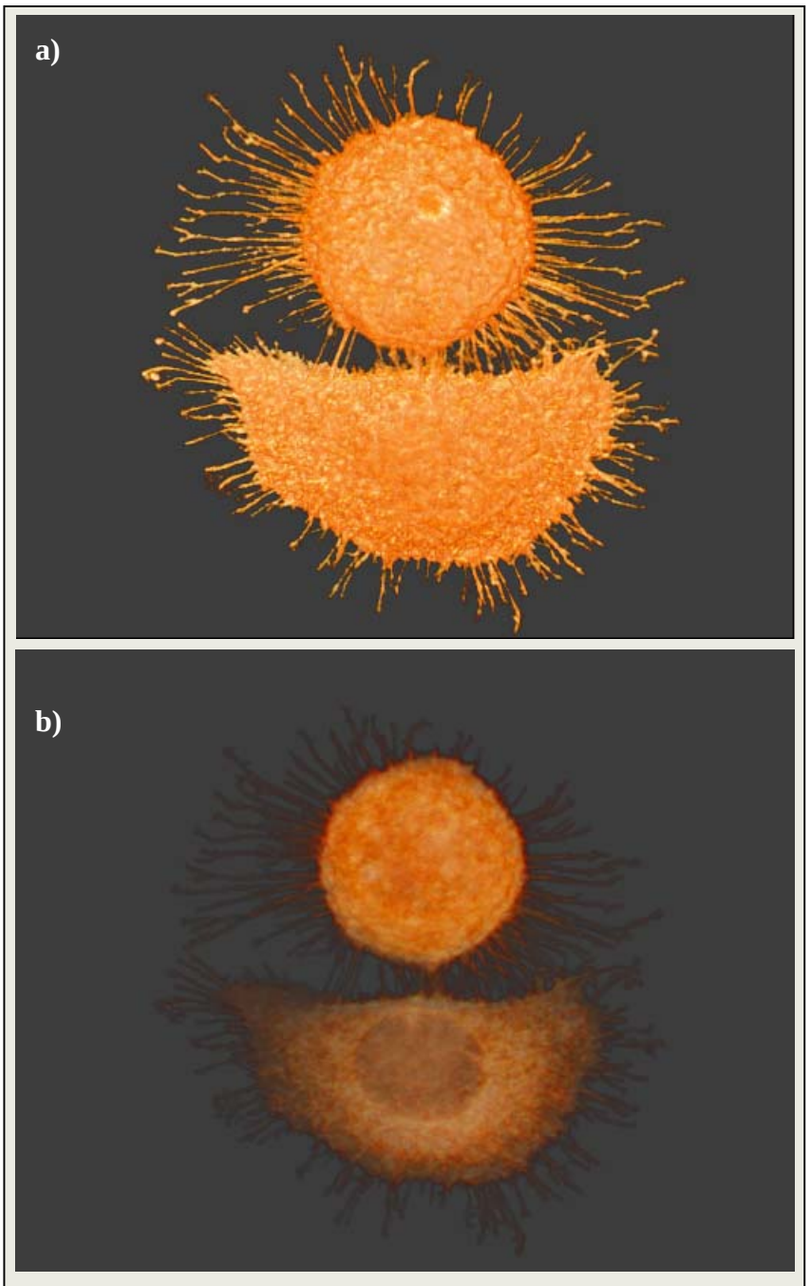


Fig. 6.6: Reconstrucción tridimensional de dos células de cultivo tipo HeLa con una LUT definida y distintos niveles de opacidad.
[a] LUT con el índice 41 que corresponde al color *Black-Red* y altos valores de opacidad. La opacidad asignada permite visualizar la morfología de ambas células, las prolongaciones y los nanotubulos.
[b] LUT con el índice 41 que corresponde al color *Black-Red* y bajos valores de opacidad. La opacidad asignada permite visualizar el núcleo y pequeñas estructuras perinucleares en el interior.

6.4 Ejemplo de Reconstrucción Tridimensional en 2 Canales de Fluorescencia

En la siguiente aplicación, se realizó una reconstrucción tridimensional en 2 canales de fluorescencia a células de cultivo tipo HeLa. Las células se marcan con los fluoróforos de membrana $DiIC_{18}$ y $DiOC_{16}$, Aparentemente, los fluoróforos se incorporan con preferencia en diferentes etapas de su ciclo celular. En estas condiciones, se generaron 67 cortes en el eje z con distancia de 0,4 μm entre cortes, para cada canal de fluorescencia. El canal rojo corresponde al fluoróforo $DiIC_{18}$, y el canal verde corresponde al fluoróforo $DiOC_{16}$,

Para la reconstrucción se realizó un pretratamiento de deconvolución. Cada canal se reconstruyó en forma independiente. El resultado de las reconstrucciones tridimensionales de ambos canales se muestra en la Fig. 6.7.

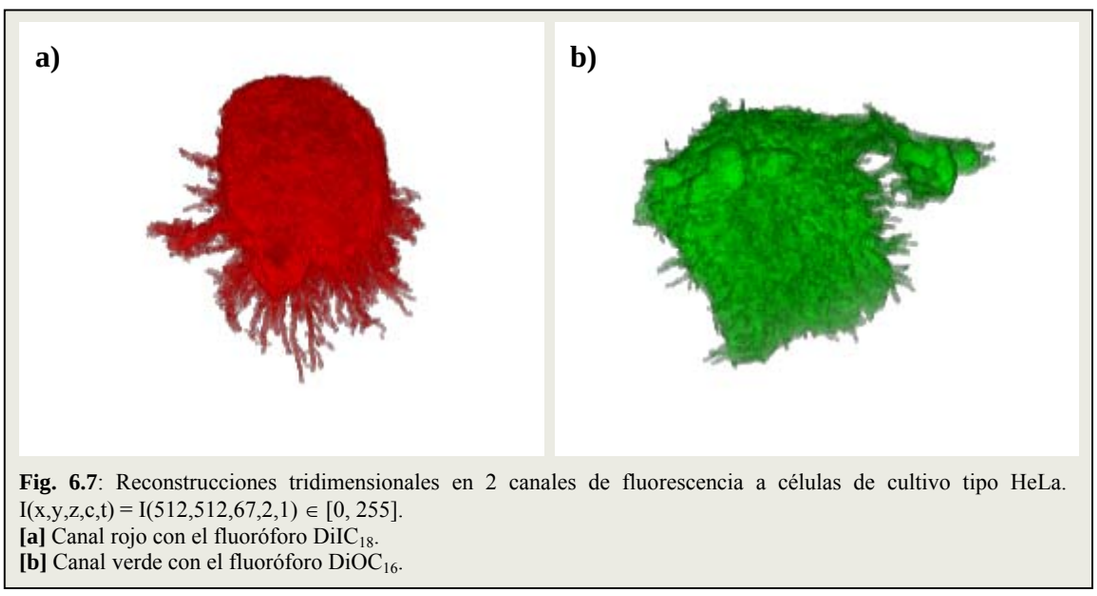


Fig. 6.7: Reconstrucciones tridimensionales en 2 canales de fluorescencia a células de cultivo tipo HeLa. $I(x,y,z,c,t) = I(512,512,67,2,1) \in [0, 255]$.
[a] Canal rojo con el fluoróforo $DiIC_{18}$.
[b] Canal verde con el fluoróforo $DiOC_{16}$.

Si bien las 2 células son objetos reconstruidos independientes, al momento de desplegarlas juntas en un espacio tridimensional, se descubrió que ambos modelos se traslaparán (Fig. 6.8b). Dentro de este trabajo, este error de IDL se corrigió presentando ambos objetos en un solo modelo. Se realizó una mezcla aditiva de las intensidades de cada objeto y se

implementó el parámetro '*Merge Volumes*', que está en la lista de parámetros interactivos del objeto (Fig. 6.8a/c).

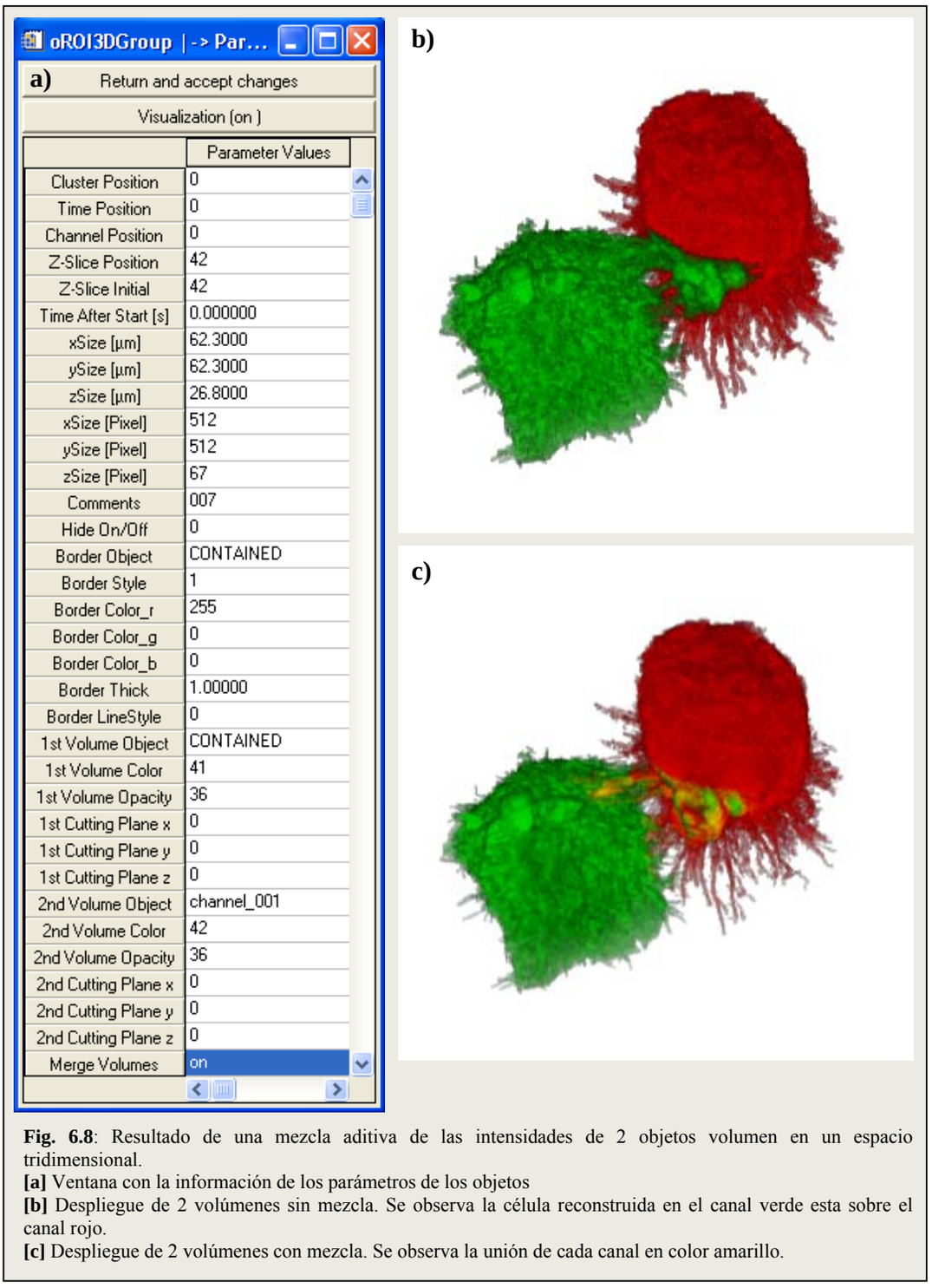


Fig. 6.8: Resultado de una mezcla aditiva de las intensidades de 2 objetos volumen en un espacio tridimensional.
[a] Ventana con la información de los parámetros de los objetos
[b] Despliegue de 2 volúmenes sin mezcla. Se observa la célula reconstruida en el canal verde esta sobre el canal rojo.
[c] Despliegue de 2 volúmenes con mezcla. Se observa la unión de cada canal en color amarillo.

En la Fig. 6.9 se puede observar el resultado de la implementación del parámetro '*Merge Volumes*' con un corte en el eje z. Los sectores amarillos en las células indican que hay una superposición entre un canal y otro. Este fenómeno es una característica importante en la microscopía confocal, que se conoce con el nombre de colocalización.

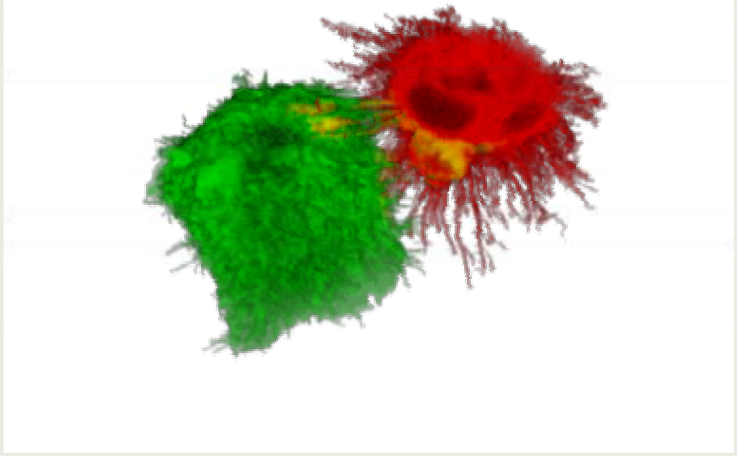


Fig. 6.9: Resultado de aplicar un '*Merge Volumes*' a reconstrucciones en 2 canales con un corte en el eje z. Los sectores amarillos indican superposición entre canales.

Capítulo 7 : Conclusiones

En primera instancia, se demostró que la deconvolución es un proceso de pretratamiento necesario para eliminar el ruido y las distorsiones introducidas por el sistema de adquisición (microscopio). Aplicando la deconvolución se obtiene una aproximación a las características reales del objeto observado. Una resolución óptima es esencial para un posterior análisis o una modelación tridimensional. La deconvolución se ha establecido como un proceso estándar en el CECS para los usuarios que trabajen con microscopía confocal de fluorescencia.

Se implementó y generó una gran variedad de LUT para la libre elección de una escala de color. De este modo, el usuario puede seleccionar la mejor escala que refleje las condiciones óptimas o visibles para representar el canal de color observado por el microscopio en la muestra biológica. La opacidad se descubrió como otra característica importante en la visualización. Diferentes grados de opacidad aplicados al objeto reconstruido hicieron posible distinguir la morfología celular y estructuras intracelulares con gran detalle.

Los algoritmos implementados dentro del sistema de objetos gráficos de IDL permitieron la reconstrucción y visualización de estructuras celulares en el margen de la resolución teórica del microscopio confocal de fluorescencia. La parametrización del volumen celular permitió monitorear por primera vez y con gran precisión la regulación del volumen en células únicas durante diferentes tipos de muerte celular (*apoptosis* y *necrosis*).

El desarrollo de un ambiente interactivo permitió acceder a parámetros intrínsecos de los objetos tridimensionales. Los objetos se pueden observar desde distintos ángulos, con cortes en xyz, cambios de escalas de color y opacidad, dando una mejor flexibilidad al momento de realizar la reconstrucción tridimensional.

Las técnicas implementadas han contribuido a establecer condiciones optimas para el análisis de la morfología y del volumen celular en diferentes condiciones experimentales. Se logró observar diferentes visualizaciones de estructuras celulares como filopodias, perturbaciones de la membrana o *tunneling nanotubes*.

Realizar un trabajo en un área relevante de investigación científica abre nuevas puertas para comprender y avanzar hacia el entendimiento del complejo mundo celular. Para muchos investigadores del área, este trabajo mejoró la posibilidad de visualizar y analizar tridimensionalmente estructuras biológicas en diferentes perspectivas, y medir el volumen celular.

Colaboración con otros proyectos de título y de investigación interdisciplinarios: informática, biología molecular, biofísica, bioquímica.

Resultados obtenidos del desarrollo de este trabajo han sido presentados en congresos nacionales e internacionales [Har04b/c] [Anc04] [Roj04].

Bibliografía

[Bar01] Barros, L., Hermosilla, T. and Castro, J. Necrotic volume increase and the early physiology of necrosis. *Comp. Biochem. Physiol. A* 130:401-409, 2001.

[Kem99] Kempen,V. M. (1999) Image Restauration in Fluorescence Microscopy. *PhD-thesis*, Technische Universiteit Delft.

[Gon02] Gonzalez R., Woods R., (2002), *Digital Image Processing*, Second Edition.

[Bar02] Barría P., Rojas R., Espinoza V. (2002).
"Plataforma de Identificación, Clasificación y Análisis de Imágenes, obtenidas a partir de Muestras Celulares", Universidad Austral de Chile.

[Jes02] Jessel, R., Haertel, S., Socaciu, C., Tykhonova, S., & Diehl H. (2002) *Kinetics of apoptotic markers in exogeneously induced apoptosis of EL4 cells*. *Journal of Cellular and Molecular Medicine*, 6(1), 82-92.

[Fan02] Fanani, M.L., Härtel, S., Oliveira, R.G., & Maggio, B. (2002) *Bidirectional control of sphingomyelinase activity and surface topography in lipid monolayers. Biophysical Journal,* 83(6), 3416-24.

[Oya03] Oyarzo, M, "Segmentación y análisis automatizado de objetos en movimiento aplicado al estudio de sistemas biológicos", Universidad Austral de Chile.

[Har03] Härtel, S., Zorn-Kruppa, M., Tikhonova, S., Heino, P., Engelke, M., & Diehl, H. (2003) *Staurosporine-induced Apoptosis in Human Cornea Epithelial Cells in Vitro*. *Cytometry*.

[Alv03a] Alvarez, M., Härtel, S. Godoy, R., & Heyser W. (2003a) *New Perspectives in the Determination of phoshatase activity in ectomycorrhiza of Nothofagus obliqua in forests of southern Chile. Gayana 60(1).*

[Alv03b] Alvarez, M., S. Godoy, R., & Heyser W., Härtel (2003b) *Structural variations of the surface bound phosphatase activity in living hyphae of ectomycorrhizal fungi of Nothofagus obliqua. In press, Mycologia.*

[Alv04] Alvarez, M., Godoy, R., Heyser, W., & Härtel, S. (2004) *Surface bound phosphatase activity in living hyphae of ectomycorrhizal fungi of Nothofagus obliqua. Mycologia* 96(3), 479 - 487.

[Har04a] Härtel, S., Rojas, R., Räth, C., Guarda, M. I., & Goicoechea, O. (2004) *Identification and Classification of Di- and Triploid Erythrocytes by Multi-parameter Image Analysis: A New Method for the Quantification of*

Triploidization Rates in Rainbow Trout (Oncorhynchus mykiss). Submitted to Archivos de Medicina Veterinaria.

[Rus04] Rustom A, Saffrich R, Markovic I, Walther P, Gerdes HH. *Nanotubular highways for intercellular organelle transport. Science.* 2004 Feb 13;303(5660):1007-10.

[Alv05] Alvarez, M., Härtel, S. Godoy, R., & Heyser W. (2005) *Anatomical-physiological determination of surface bound phosphatase activity in ectomycorrhiza of Nothofagus obliqua based on image processed confocal fluorescence microscopy. Soil Biology and Biochemistry*. 37(1), 125-132.

[Har05] Härtel, S., Fanani, M.L.,& Maggio, B. (2005*) Shape transitions and lattice structuring of ceramide-enriched domains generated by sphingomyelinase in lipid monolayers. Biophysical Journal*, Oct 15, [Epub ahead of print].

[Mon05] Montenegro, C. (2005). *"Modulación de TASK-2 por la acidificación extracelular y posible papel en la regulación de volumen celular"*, Universidad Austral de Chile.

Direcciones Electrónicas

[URL1] Huygens Professional User Guide (2003), Scientific Volume Imaging www.svi.nl

[URL2] Departamento de Física Matemática y Fluidos, "El ordenador en el laboratorio: Medir con imágenes".
http://dfmf.uned.es/actividades/no_reglada/laboratorio /segmentacion1.pdf

[URL3] Fernandez, M.M. (2002), Técnicas Clásicas de Segmentación de Imagen.
http://poseidon.tel.uva.es/~carlos/ltif10001/segmenclasica.pdf

Charla de Conferencia

[Har04b] Härtel, S., Verdugo, E., Montenegro, C., Niemeyer, M.I., Fanani, M., & L.F. Barros, *Time Resolved 3-Dimensional Image Processing of Microscopical Data and its Application in Membrane Related Cellular Processes*. XVIII Reunión anual, Sociedad de Biología Celular de Chile, Pucón, 13-17 Octubre, 2004.

[Anc04] Ancalao, J., Orta, G., Verdugo, E., Härtel, S., & R.Latorre. *Importance of the potential sensor in the structure of the channel hSlo. Joint meeting of the UK-Physiological Society with the Chilean Physiological Society*, King's College London, UK, 17-20 December, 2004.

Presentación de póster

[Har04c] Härtel, S., Verdugo, E., Montenegro, C., Niemeyer, M.I., Fanani, M., & L.F. Barros, *Deconvolution of Microscopal Data for Time Resolved 3-Dimensional Image Processing for the Study of Membrane Related Processes in Cell Biology*. *Representation of reality by brain and machines. Crossed views from neurosciences and computer vision*. Montevideo. Uruguay, 8-12 Noviembre, 2004.

[Roj04] Rojas, R., & S. Härtel, *Identification and Classification of Di- and Triploid Erythrocytes by Multi-parameter Image Analysis: A New Method for the Quantification of Triploidization Rates in Rainbow Trout. Representation of reality by brain and machines. Crossed views from neurosciences and computer vision*. Montevideo. Uruguay, 8-12 November, 2004.

Anexos

A. Lista de Parámetros de Objetos Implementados

A.1 IDLgrImage::Init

The IDLgrImage::Init function method initializes the image object.

Obj = OBJ_NEW('IDLgrImage' [, *ImageData*] [, BLEND_FUNCTION{Get,Set}=*vector*] [, CHANNEL{Get, Set}=*hexadecimal bitmask*] [, DATA{Get, Set}=*nxm*, *2xnxm*, *3xnxm*, or *4xnxm array of image data*] [, DIMENSIONS{Get, Set}=*[width*, *height]*] [, /GREYSCALE{Get, Set}] [, /HIDE{Get, Set}] [, INTERLEAVE{Get, Set}={0 | 1 | 2}] [, /INTERPOLATE{Get, Set}] [LOCATION{Get, Set}=*[x, y] or [x, y, z]*] [, NAME{Get, Set}=*string*] [, /NO_COPY{Get, Set}] [, /ORDER{Get, Set}] [, PALETTE{Get, Set}=*objref*] [, /RESET_DATA{Set}] [, SHARE_DATA{Set}=*objref*] [, SUB_RECT{Get, Set}=*[x*, *y*, *xdim*, *ydim]*] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])
or
Result = *Obj* -> [IDLgrImage::]Init([*ImageData*]) (solamente en una subclase del método Init)

A.2 IDLgrPolygon::Init

The IDLgrPolygon::Init function method initializes the polygons object.

Obj = OBJ_NEW('IDLgrPolygon' [, *X* [, *Y*[, *Z*]]] [, BOTTOM{Get, Set}=*index or RGB vector*] [, COLOR{Get, Set}=*index or RGB vector* | , VERT_COLORS{Get, Set}=*vector*] [, DATA{Get, Set}=*array*] [, /DOUBLE{Get, Set}] [, FILL_PATTERN{Get, Set}=*objref to IDLgrPattern object*] [, /HIDDEN_LINES] [, /HIDE{Get, Set}] [, LINESTYLE{Get, Set}=*value*] [, NAME{Get, Set}=*string*] [, NORMALS{Get, Set}=*array*] [, PALETTE=*objref*] [, POLYGONS{Get, Set}=*array of polygon descriptions*] [, REJECT{Get, Set}={0 | 1 | 2}] [, /RESET_DATA{Set}] [, SHADE_RANGE{Get, Set}=*array*] [, SHADING{Get, Set}={0 | 1}] [, SHARE_DATA{Set}=*objref*] [, STYLE{Get, Set}={0 | 1 | 2}] [, TEXTURE_COORD{Get, Set}=*array*] [, /TEXTURE_INTERP{Get, Set}] [, TEXTURE_MAP*{Get, Set}=objref to IDLgrImage object*] [, THICK{Get, Set}=*points*{1.0 to 10.0}] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*] [, ZERO_OPACITY_SKIP{Get, Set}={0 | 1}])
or
Result = *Obj* -> [IDLgrPolygon::]Init([*X*, [*Y*, [*Z*]]]) (solamente en una subclase del método Init)

A.3 IDLgrVolume::Init

The IDLgrVolume::Init function method initializes the volume object. At least one volume method must be specified, via arguments or keywords.

Obj = OBJ_NEW('IDLgrVolume' [, *vol0* [, *vol1* [, *vol2* [, *vol3*]]]] [, AMBIENT{Get, Set}=*RGB vector*] [, BOUNDS{Get, Set}=*[xmin*, *ymin*, *zmin*, *xmax*, *ymax*, *zmax]*] [, COMPOSITE_FUNCTION{Get, Set}={0 | 1 | 2 | 3}] [, CUTTING_PLANES{Get, Set}=*array*] [, DATA0{Get, Set}=*[dx*, *dy*, *dz]*] [, DATA1{Get, Set}=*[dx*, *dy*, *dz]*] [, DATA2{Get, Set}=*[dx*, *dy*, *dz]*] [, DATA3{Get, Set}=*[dx*, *dy*, *dz]*] [, DEPTH_CUE{Get, Set}=*[zbright*, *zdim]*] [, /HIDE{Get, Set}] [, HINTS{Get, Set}={0 | 1 | 2 | 3}] [, /INTERPOLATE{Get, Set}] [, /LIGHTING_MODEL{Get, Set}] [, NAME{Get, Set}=*string*] [, /NO_COPY{Get, Set}] [, OPACITY_TABLE0{Get, Set}=*256-element byte array*] [, OPACITY_TABLE1{Get, Set}=*256-element byte array*] [, RENDER_STEP{Get,

Set}=*[x, y, z]*] [, RGB_TABLE0{Get, Set}=*256 x 3-element byte array*] [, RGB_TABLE1{Get, Set}=*256 x 3-element byte array*] [, /TWO_SIDED{Get, Set}] [, UVALUE{Get, Set}=*value*] [, VOLUME_SELECT{Get, Set}={0 | 1 | 2}] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, /ZBUFFER{Get, Set}] [, ZCOORD_CONV{Get, Set}=*vector*] [, ZERO_OPACITY_SKIP{Get, Set}={0 | 1}])
or
Result = Obj -> [IDLgrVolume::]Init([*vol0* [, *vol1* [, *vol2* [, *vol3*]]]]) (solamente en una subclase del método Init)

B. Código de los Algoritmos Implementados

Modelo de Imagen

```
pro C_sROIGroupObject::getoImageModel, oObjectModel
        ZSlicePosition = *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'z-Slice Position'))[0]]

        ctab = self->getPalette(3)
        whereParam = (where(*(*self.pParamStruct).pNames eq 'Image Color'))[0]
        if (whereParam ne -1) then begin
                x = (*(*self.pParamStruct).pValues[whereParam] > 0) < 40
                *(*self.pParamStruct).pValues[whereParam] = x
                ctab=self ->getpalette(x)
        endif
        maskImage = self->getGroupMaskIntensity()
        imDim = size(maskImage, /dim)
        img = bytArr(4, imDim[0], imDim[1] )
        img[0,*,*] = ctab[0,maskImage]
        img[1,*,*] = ctab[1,maskImage]
        img[2,*,*] = ctab[2,maskImage]
        img[3,*,*] = (maskImage gt 0) * 255

        xp = [-0.5, 0.5]
        yp = [-0.5, 0.5]
        self->getProperty, xCoord_conv = xCoord_conv, yCoord_conv = yCoord_conv, zCoord_conv = zCoord_conv
        zp= zCoord_conv[1] * ZSlicePosition
        div = 1. * (imDim[0]<imDim[1]) / (imDim[0]>imDim[1])
        if (imDim[0] lt imDim[1]) then xp = xp * div - (.5 - .5 * div) else yp = yp * div - (.5 - .5 * div)

; Creación del objeto imagen
        oTextureImage = obj_new('IDLgrImage', img, $
        xCoord_conv = xCoord_conv, yCoord_conv = yCoord_conv, zCoord_conv = zCoord_conv, $
        interleave = 0,$
        dim=[512,512], $
        loc=[0,0], $
        hide = 1)
        oObjectModel->add, obj_new('IDLgrPolygon', $
        [[xp[0],yp[0],zp],[xp[1],yp[0],zp],[xp[1],yp[1],zp],[xp[0],yp[1],zp]], $
        texture_coord = [[0,0],[1,0],[1,1],[0,1]], $
        texture_map = oTextureImage, $
        /texture_int, $
        color=[255,255,255] )
end
```

Modelo de Bordes

```
pro C_sROIGroupObject::getoBorderModel, oObjectModel, color = color
        if not(keyWord_set(color)) then color = [0,255,0]

        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border Color_r'))[0]
        if (whereParam ne -1) then begin
                color = [ *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border Color_r'))[0]] , *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border Color_g'))[0]] , self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border Color_b'))[0]]] > [0,0,0] < [255,255,255]
        endif

        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border PolygonStyle'))[0]
        if (whereParam ne -1) then begin
                modelStyle = (*(*self.pParamStruct).pValues[whereParam] > 0) < 2
                *(*self.pParamStruct).pValues[whereParam] = modelStyle
        endif
```

```
        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border Thick'))[0]
        if (whereParam ne -1) then begin
                borderThick = *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq
                'Border Thick'))[0]] > 1. < 10.
                *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border
                Thick'))[0]] = borderThick
        endif else borderThick = 1.

        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border LineStyle'))[0]
        if (whereParam ne -1) then begin
                borderLineStyle = *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq
                'Border LineStyle'))[0]] > 0 < 6
                *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border
                LineStyle'))[0]] = borderLineStyle
        endif else borderLineStyle = 0
        ZSlicePosition = *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'z-
        Position [real]'))[0]]

        for i = 0, (self->count())-1 do (self->get(position = i))->addObjectBorderPolygon, color = color,
        oModel = oObjectModel, modelStyle = modelStyle, borderThick = borderThick, borderLineStyle =
        borderLineStyle, ZSlicePosition = ZSlicePosition
end
```

Modelo de bordes tridimensional

```
pro C_sROI3DGroupObject::getoBorderModel, oObjectModel
        if not(keyWord_set(color)) then color = [0,255,0]

        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border Color_r'))[0]
        if (whereParam ne -1) then begin
                color = [ *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border
                Color_r'))[0]], *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq
                'Border Color_g'))[0]] ,(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames
                eq 'Border Color_b'))[0]]] > [0,0,0] < [255,255,255]
        endif

        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border Style'))[0]
        if (whereParam ne -1) then begin
                modelStyle = (*(*self.pParamStruct).pValues[whereParam] > 0) < 2
                *(*self.pParamStruct).pValues[whereParam] = modelStyle
        endif

        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border Thick'))[0]
        if (whereParam ne -1) then begin
                borderThick = *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq
                'Border  Thick'))[0]] > 1. < 10.
                *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border
                Thick'))[0]] =      borderThick
        endif else borderThick = 1.

        whereParam = (where(*(*self.pParamStruct).pNames eq 'Border LineStyle'))[0]
        if (whereParam ne -1) then begin
                borderLineStyle = *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq
                'Border LineStyle'))[0]] > 0 < 6
                *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Border
                LineStyle'))[0]] = borderLineStyle
        endif else borderLineStyle = 0

        for i = 0, (self->count())-1 do (self->get(position = i))->makeZSliceObjectBorderPolygon, color =
        color, oModel = oObjectModel, modelStyle = modelStyle,borderThick = borderThick,
        borderLineStyle = borderLineStyle
end
```

Modelo de Volumen

```
pro C_sROI3DGroupObject::getoVolumeModel, oObjectModel, volumeColor=volumeColor, stack_tlb = stack_tlb

        xyzFramePixSize = [*(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'x-Size [pixel]'))[0]],*(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'y-Size [pixel]'))[0]], *(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'z-Size [pixel]'))[0]] ]
        ; conseguir primer objeto y parámetro de los objetos
        volData1 = bytArr(xyzFramePixSize[0], xyzFramePixSize[1], xyzFramePixSize[2])
        for i = 0, (self->count()-1) do volData1[*((self->get(position = i))->getpWherePoints())] = *((self->get(position = i))->getpPointValues())
        rgb_table0 = (bytArr(256,3) + transpose((*self.pvolState).rgbValues[0,*,*]))

        whereNo = where(volData1 lt ((*self.pVolState).bottomValues[0]))
        if (whereNo[0] ne -1) then volData1[whereNo] = 0
        whereNo = where(volData1 gt ((*self.pVolState).topValues[0]))
        if (whereNo[0] ne -1) then volData1[whereNo] = 0

        case ((*self.pVolState).opacFlag[0]) of
        0: begin
        whereParam =  (where(*(*self.pParamStruct).pNames eq '1st Volume Opacity'))[0]
        if (whereParam ne -1) then begin
                opacVal = (*(*self.pParamStruct).pValues[whereParam] > 0) < 255
                *(*self.pParamStruct).pValues[whereParam] = opacVal
                opacVect_0 = make_array(256, /byte, value = opacVal)
                opacVect_0[ 0 : ((*self.pVolState).bottomValues[0]) ] = 0
                if (((*self.pVolState).topValues[0]) le 254) then opacVect_0[
                ((*self.pVolState).topValues[0])+1 : 255 ] = 0
            endif
          endcase
       else: opacVect_0 = bytArr(256) + (*self.pVolState).opacValues[0,*]
        endcase

        whereParam = (where(*(*self.pParamStruct).pNames eq '1st Cutting Plane x'))[0]
        if (whereParam ne -1) then begin
                x1=fix((*(*self.pParamStruct).pValues[whereParam]    >    (-xyzFramePixSize[0]+1))    <    (xyzFramePixSize[0]-1))
                 *(*self.pParamStruct).pValues[whereParam] = x1
                        if (x1 le 0) then factorX1 = 1 else factorX1 = -1
                        endif else x1 = 0
                whereParam = (where(*(*self.pParamStruct).pNames eq '1st Cutting Plane y'))[0]

        if (whereParam ne -1) then begin
                y1=fix((*(*self.pParamStruct).pValues[whereParam]    >    (-xyzFramePixSize[1]+1))    <    (xyzFramePixSize[1]-1))
                *(*self.pParamStruct).pValues[whereParam] = y1
                        if (y1 le 0) then factorY1 = 1 else factorY1 = -1
        endif else y1 = 0
                whereParam = (where(*(*self.pParamStruct).pNames eq '1st Cutting Plane z'))[0]

        if (whereParam ne -1) then begin
                z1=fix((*(*self.pParamStruct).pValues[whereParam]    >    (-xyzFramePixSize[2]+1))    <    (xyzFramePixSize[2]-1))
                *(*self.pParamStruct).pValues[whereParam] = z1
                         if (z1 le 0) then factorZ1 = 1 else factorZ1 = -1
        endif else z1 = 0

    ; Conseguir segundo objeto y parámetros del objeto
        whereName = (where(*(*self.pParamStruct).pNames eq '2nd Volume Object'))[0]
        if (whereName ne -1) then begin
                strSelect = *(*self.pParamStruct).pValues[(where( *(*self.pParamStruct).pNames eq '2nd lume Object'))[0]]
```

```
        case 1 of
            ( (strPos(strSelect, 'Channel') ne -1) or (strPos(strSelect, 'channel') ne -1) ) :begin

                trNum = s_getRightNumberFromString(strSelect)

                clusterPosition=*(*self.pParamStruct).pValues(where(*(*
                self.pParamStruct ).pNames eq 'Cluster Position'))[0]]

                timePosition = *(*self.pParamStruct).pValues[(where(*(*
                self.pParamStruct).pNames eq 'Time Position'))[0]]

                oROI3DGroup = s_ISegM_GetROI3DGroup(stack_tlb = stack_tlb,
                timePosition = timePosition, channelPosition = strNum, clusterPosition =
                clusterPosition, fileName = fileName)
            endcase
            ( (strPos(strSelect, 'Time') ne -1) or (strPos(strSelect, 'time') ne -1) ):begin

                strNum = s_getRightNumberFromString(strSelect)

                clusterPosition = *(*self.pParamStruct).pValues[(where(*(*
                self.pParamStruct).pNames eq 'Cluster Position'))[0]]

                channelPosition = *(*self.pParamStruct).pValues[(where(*(*
                self.pParamStruct).pNames eq 'Channel Position'))[0]]

                oROI3DGroup = s_ISegM_GetROI3DGroup(stack_tlb = stack_tlb,
                timePosition = strNum, channelPosition = channelPosition, clusterPosition
                = clusterPosition, fileName = fileName)
            endcase
            ( (strPos(strSelect, 'Cluster') ne -1) or (strPos(strSelect, 'cluster') ne -1) ):begin
                strNum = s_getRightNumberFromString(strSelect)
                clusterPosition = *(*self.pParamStruct).pValues[(where(*(*
                self.pParamStruct).pNames eq 'Cluster Position'))[0]]
                timePosition = *(*self.pParamStruct).pValues[(where(*(*
                self.pParamStruct).pNames eq 'Time Position'))[0]]

                channelPosition = *(*self.pParamStruct).pValues[(where(*(*
                self.pParamStruct).pNames eq 'Channel Position'))[0]]

                channelPosition = 1

                oROI3DGroup = s_ISegM_GetROI3DGroup(stack_tlb = stack_tlb,
                timePosition = timePosition, channelPosition = channelPosition,
                clusterPosition = strNum, fileName = fileName)
            endcase
            else :
        endcase
endif

if (obj_valid(oROI3DGroup)) then begin
        volData2 = bytArr(xyzFramePixSize[0], xyzFramePixSize[1], xyzFramePixSize[2])
        for i = 0, (oROI3DGroup->count()-1) do volData2[*((oROI3DGroup->get(position = i))-
        >getpWherePoints())] = *((oROI3DGroup->get(position = i))->getpPointValues())
        rgb_table1 = (bytArr(256,3) + transpose((*self.pvolState).rgbValues[1,*,*]))

        whereNo = where(volData2 lt ((*self.pVolState).bottomValues[1]))
        if (whereNo[0] ne -1) then volData2[whereNo] = 0
        whereNo = where(volData2 gt ((*self.pVolState).topValues[1]))
        if (whereNo[0] ne -1) then volData2[whereNo] = 0

        case ((*self.pVolState).opacFlag[1]) of
            0: begin
                    whereParam =  (where(*(*self.pParamStruct).pNames eq '2nd
                    Volume Opacity'))[0]
```

```
                                        if (whereParam ne -1) then begin
                                                opacVal = (*(*self.pParamStruct).pValues[whereParam]
                                                > 0) < 255
                                                *(*self.pParamStruct).pValues[whereParam] = opacVal
                                                opacVect_1 = make_array(256, /byte, value = opacVal)
                                                opacVect_1[  0  :  ((*self.pVolState).bottomValues[1])  ]
                                                = 0
                                                if (((*self.pVolState).topValues[1]) le 254) then
                                                opacVect_1[  ((*self.pVolState).topValues[1])+1  :  255  ]
                                                = 0
                                        endif
                                endcase
                        else: opacVect_1 = bytArr(256) + (*self.pVolState).opacValues[1,*]
                endcase

                whereParam = (where(*(*self.pParamStruct).pNames eq '2nd Cutting Plane x'))[0]
                if (whereParam ne -1) then begin
                        x2 = fix((*(*self.pParamStruct).pValues[whereParam] > (-xyzFramePixSize[0]+1))
                        < (xyzFramePixSize[0]-1))
                        *(*self.pParamStruct).pValues[whereParam] = x2
                        if (x2 le 0) then factorX2 = 1 else factorX2 = -1
                endif else x2 = 0

                whereParam = (where(*(*self.pParamStruct).pNames eq '2nd Cutting Plane y'))[0]
                if (whereParam ne -1) then begin
                        y2 = fix((*(*self.pParamStruct).pValues[whereParam] > (-xyzFramePixSize[1]+1))
                        < (xyzFramePixSize[1]-1))
                        *(*self.pParamStruct).pValues[whereParam] = y2
                        if (y2 le 0) then factorY2 = 1 else factorY2 = -1
                endif else y2 = 0

                whereParam = (where(*(*self.pParamStruct).pNames eq '2nd Cutting Plane z'))[0]
                 if (whereParam ne -1) then begin
                        z2 = fix((*(*self.pParamStruct).pValues[whereParam] > (-xyzFramePixSize[2]+1))
                        < (xyzFramePixSize[2]-1))
                        *(*self.pParamStruct).pValues[whereParam] = z2
                        if (z2 le 0) then factorZ2 = 1 else factorZ2 = -1
                        endif else z2 = 0
        endif

self->getProperty, xCoord_conv = xCoord_conv, yCoord_conv = yCoord_conv, zCoord_conv =
zCoord_conv
        case (*(*self.pParamStruct).pValues[(where(*(*self.pParamStruct).pNames eq 'Merge
                Volumes'))[0]]) of
        'off':  begin
                        oObjectModel->add, obj_new('IDLgrVolume',   data0 = volData1, opacity_table0 =
                        opacVect_0, rgb_table0 = rgb_table0, /interpolate, uValue = 'o3DVolumeModel1',
                        /zero, xCoord_conv = xCoord_conv, yCoord_conv = yCoord_conv, zCoord_conv
                        = zCoord_conv, cutting_plane = [[factorX1,0,0, x1], [0,factorY1,0, y1],
                        [0,0,factorZ1, z1]], ambient = [255, 255, 255],lighting_model=1)
                if (obj_valid(oROI3DGroup)) then
                        oObjectModel->add, obj_new('IDLgrVolume',  data0 = volData2, opacity_table0 =
                        opacVect_1, rgb_table0 = rgb_table1, /interpolate, uValue = 'o3DVolumeModel2',
                        /zero, xCoord_conv = xCoord_conv, yCoord_conv = yCoord_conv, zCoord_conv
                        = zCoord_conv,   cutting_plane = [[factorX2,0,0, x2], [0,factorY2,0, y2],
                        [0,0,factorZ2, z2]], ambient = [255, 255, 255],lighting_model=1)
        endcase
        'on':   begin
                        if (x1 lt 0) then volData1[0: -(x1+1),*,*] = 0
                        if (x1 gt 0) then volData1[ x1:*,*,*] = 0
                        if (y1 lt 0) then volData1[*, 0: -(y1+1),*] = 0
                        if (y1 gt 0) then volData1[*, y1:*,*] = 0
                        if (z1 lt 0) then volData1[*,*, 0: -(z1+1)] = 0
                        if (z1 gt 0) then volData1[*,*, z1:*] = 0
```

```
volData_0 = (bytArr(256) + transpose((*self.pvolState).rgbValues[0,0,*]))[volData1]
volData_1 = (bytArr(256) + transpose((*self.pvolState).rgbValues[0,1,*]))[volData1]
volData_2 = (bytArr(256) + transpose((*self.pvolState).rgbValues[0,2,*]))[volData1]
case ((*self.pVolState).opacFlag[0]) of
0: begin
        whereParam =  (where(*(*self.pParamStruct).pNames eq '1st Volume Opacity'))[0]
        if (whereParam ne -1) then begin
        opacVal = (*(*self.pParamStruct).pValues[whereParam] > 0) < 255
                *(*self.pParamStruct).pValues[whereParam] = opacVal
        volData_3 = make_array(xyzFramePixSize[0], xyzFramePixSize[1],
        xyzFramePixSize[2], /byte, value = opacVal)
        whereI = where(volData1 le (*self.pVolState).bottomValues[0])
                if (whereI[0] ne -1) then volData_3[whereI] = 0
                        if (((*self.pVolState).topValues[0]) le 254) then begin
                        whereI = where(volData1 ge (*self.pVolState).topValues[0])
                                if (whereI[0] ne -1) then volData_3[whereI] = 0
                                endif
                        endif
                        endcase
                        else: volData_3 = (bytArr(256) +
                        transpose((*self.pvolState).opacValues[0,*]))[volData1]
endcase

        if (obj_valid(oROI3DGroup)) then begin
        if (x2 lt 0) then volData2[0: -(x2+1),*,*] = 0
        if (x2 gt 0) then volData2[ x2:*,*,*] = 0
        if (y2 lt 0) then volData2[*, 0: -(y2+1),*] = 0
        if (y2 gt 0) then volData2[*, y2:*,*] = 0
        if (z2 lt 0) then volData2[*,*, 0: -(z2+1)] = 0
        if (z2 gt 0) then volData2[*,*, z2:*] = 0

volData_0 = byte((fix(temporary(volData_0) + (bytArr(256) +
                transpose((*self.pvolState).rgbValues[1,0,*]))[volData2])) < 255)
volData_1 = byte((fix(temporary(volData_1) + (bytArr(256) +
                transpose((*self.pvolState).rgbValues[1,1,*]))[volData2])) < 255)
volData_2 = byte((fix(temporary(volData_2) + (bytArr(256) +
                transpose((*self.pvolState).rgbValues[1,2,*]))[volData2])) < 255)
case ((*self.pVolState).opacFlag[1]) of
0: begin
        whereParam =  (where(*(*self.pParamStruct).pNames eq '2nd Volume
                        Opacity'))[0]
        if (whereParam ne -1) then begin
                opacVal = (*(*self.pParamStruct).pValues[whereParam] > 0) < 255
                        *(*self.pParamStruct).pValues[whereParam] = opacVal
                volData2_3 = make_array(xyzFramePixSize[0], xyzFramePixSize[1],
                        xyzFramePixSize[2], /byte, value = opacVal)
        whereI = where(volData2 le (*self.pVolState).bottomValues[1])
        if (whereI[0] ne -1) then volData2_3[whereI] = 0
        if (((*self.pVolState).topValues[1]) le 254) then begin
        whereI = where(volData2 ge (*self.pVolState).topValues[1])
                if (whereI[0] ne -1) then volData2_3[whereI] = 0
                endif
        endif
        endcase
        else: volData2_3 = (bytArr(256) +
                transpose((*self.pvolState).opacValues[1,*]))[volData2]
endcase
volData_3 = temporary(volData_3) > volData2_3
endif
oObjectModel->add, obj_new('IDLgrVolume', volume_select = 2, data0 = volData_0, data1
= volData_1, data2 = volData_2, data3 = volData_3, /interpolate, uValue =
'o3DVolumeModel1', /zero, xCoord_conv = xCoord_conv, yCoord_conv = yCoord_conv,
zCoord_conv = zCoord_conv, ambient = [255, 255, 255], lighting_model=1)
endcase
```

Cálculo del Volumen Celular

```
; set Object Number Vector
*(*self.pParamStruct).pROINumberVect = (C_sROI3DGroupObject->getObjectNumberVector())

//hacer bucle para determinar el tamaño de cada objeto [voxel]
paramVect = fltArr(nObjects)
for i = 0, nObjects-1 do paramVect[i] = n_elements( *((C_sROI3DGroupObject->get(position = i))-
>getpWherePoints()) )

; Volumen 3D [Voxel]
if (whereParamActive[0]) then *(*(*self.pValueStruct)[whereParam[0]]).pROIParamVect =
paramVect

; Volumen 3D [x³]
if (whereParamActive[1]) then begin
pParamStruct = C_sROI3DGroupObject->getpParamStruct()
xSizePerPixel = *(*pParamStruct).pValues[(where(*(*pParamStruct).pNames eq 'x-Size
[real]'))[0]] * 1. / *(*pParamStruct).pValues[(where(*(*pParamStruct).pNames eq 'x-Size
[pixel]'))[0]]
ySizePerPixel = *(*pParamStruct).pValues[(where(*(*pParamStruct).pNames eq 'y-Size
[real]'))[0]] * 1. / *(*pParamStruct).pValues[(where(*(*pParamStruct).pNames eq 'y-Size
[pixel]'))[0]]
zSizePerPixel = *(*pParamStruct).pValues[(where(*(*pParamStruct).pNames eq 'z-Size
[real]'))[0]] * 1. / *(*pParamStruct).pValues[(where(*(*pParamStruct).pNames eq 'z-Size
[pixel]'))[0]]
(*(*(*self.pValueStruct)[whereParam[1]]).pValues)[(where(
*(*(*self.pValueStruct)[whereParam[1]]).pNames eq 'x-Size per Pixel'))[0]] = xSizePerPixel
(*(*(*self.pValueStruct)[whereParam[1]]).pValues)[(where(
*(*(*self.pValueStruct)[whereParam[1]]).pNames eq 'y-Size per Pixel'))[0]] = ySizePerPixel
(*(*(*self.pValueStruct)[whereParam[1]]).pValues)[(where(
*(*(*self.pValueStruct)[whereParam[1]]).pNames eq 'z-Size per Pixel'))[0]] = zSizePerPixel
*(*(*self.pValueStruct)[whereParam[1]]).pROIParamVect = paramVect *
(xSizePerPixel*ySizePerPixel*zSizePerPixel)
```

C. Análisis de Varianza para la Calibración de la Distancia entre Cortes en el Eje z (Z-slices)

ANOVA (análisis de varianza para comprobar la igualdad de las medias de dos o más muestras por medio de la distribución F)

Summary Statistics

Dataset	N	Mean	SD	SE
Data3_A	1	1,77713	1,03401	1,03401
Data3_B	6	1,755	0,07396	0,03019
Data3_C	5	1,706	0,05727	0,02561
Data3_D	6	1,78989	0,09356	0,0382
Data3_E	6	2,14683	0,22139	0,09038

Null Hypothesis: The means of all selected datasets are equal
Alternative Hypothesis: The means of one or more selected datasets are different

ANOVA

Source	DoF	Sum of Squares	Mean Square	F Value	P Value
Model	4	0,713340609	0,178335152	10,28964	1,31787E-4
Error	19	0,329299020	0,0173315274		

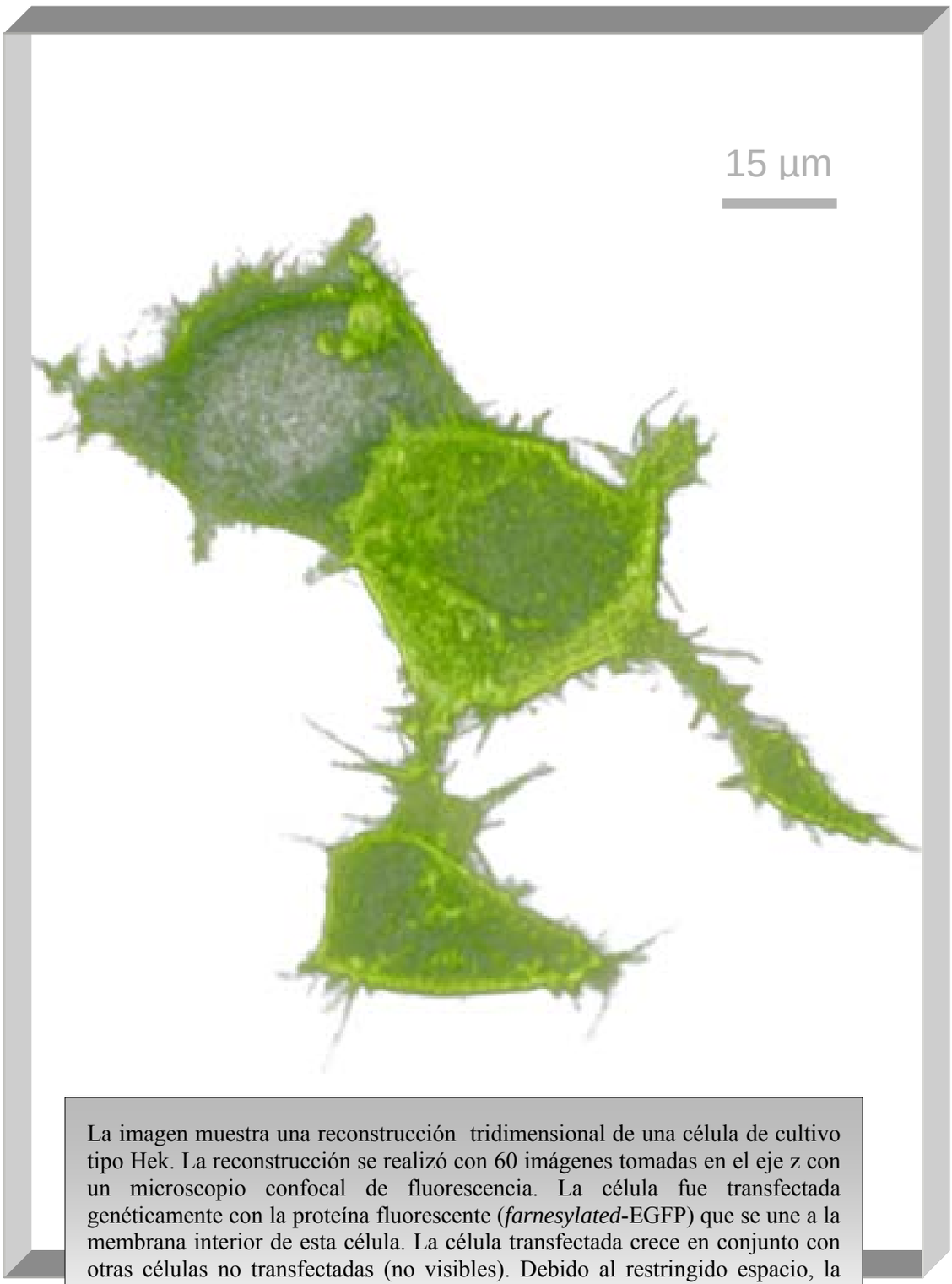
At the 0,05 level, the population means are significantly different.

Means Comparison using Bonferroni Test

Dataset	Mean	Difference between Means	Simultaneous Confidence Intervals Lower Limit	Simultaneous Confidence Intervals Upper Limit	Significant at 0,05 Level
Data3_A	1,77713				
Data3_B	1,755	0,02213	-0,42917	0,47343	No
Data3_C	1,706	0,07113	-0,38657	0,52883	No
Data3_D	1,78989	-0,01276	-0,46406	0,43853	No
Data3_E	2,14683	-0,3697	-0,821	0,08159	No
Data3_B	1,755				
Data3_C	1,706	0,049	-0,204	0,302	No
Data3_D	1,78989	-0,03489	-0,27612	0,20634	No
Data3_E	2,14683	-0,39183	-0,63306	-0,15061	Yes
Data3_C	1,706				
Data3_D	1,78989	-0,08389	-0,33689	0,16911	No
Data3_E	2,14683	-0,44083	-0,69383	-0,18783	Yes
Data3_D	1,78989				
Data3_E	2,14683	-0,35694	-0,59817	-0,11571	Yes

D. Galería de Reconstrucciones Tridimensionales

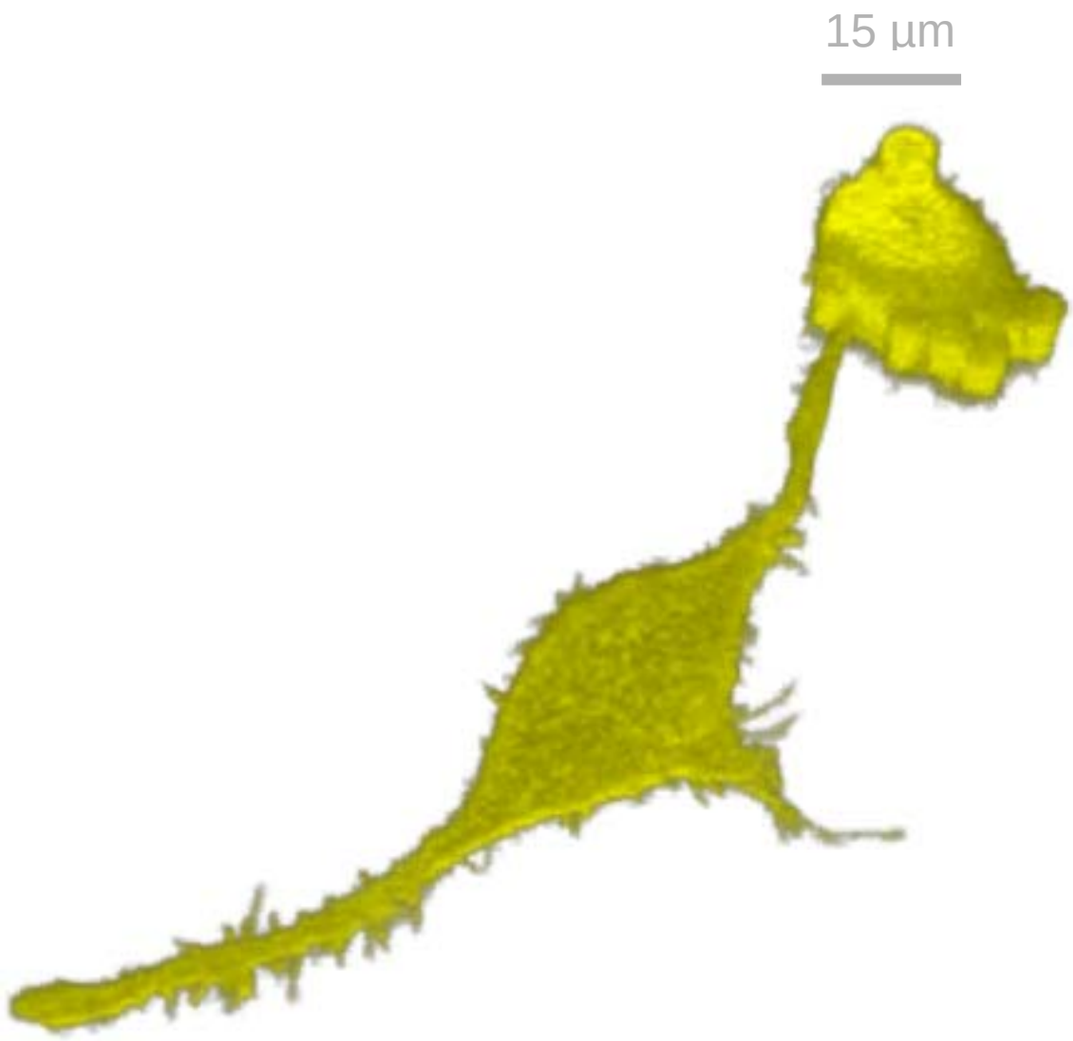
La imagen muestra una reconstrucción tridimensional de una célula necrótica (muerta) de cultivo tipo HeLa. La reconstrucción se realizó con 41 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. La membrana de la célula fue marcada con el fluoróforo $DiIC_{18}$. La muerte necrótica se manifiesta pocos minutos después de la aplicación de un estrés mediante peróxido (H_2O_2, 32 mM). En HeLa, el peróxido induce un desequilibrio en el control de volumen celular que a su vez conduce a la formación de vesículas (*blebs*) en la membrana plasmática. La barra negra en la esquina superior representa 10 µm.



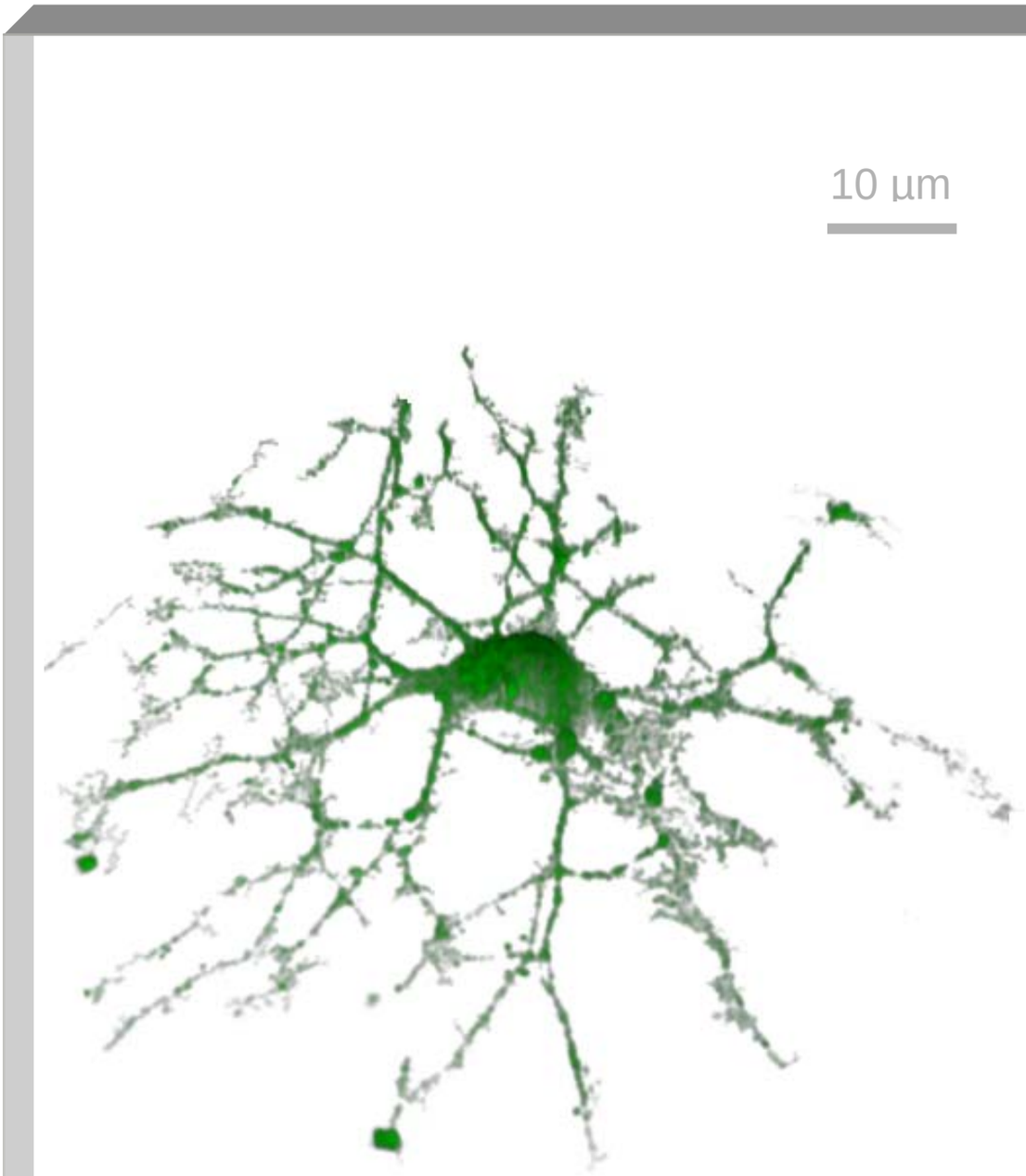
La imagen muestra una reconstrucción tridimensional de una célula de cultivo tipo Hek. La reconstrucción se realizó con 60 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. La célula fue transfectada genéticamente con la proteína fluorescente (*farnesylated*-EGFP) que se une a la membrana interior de esta célula. La célula transfectada crece en conjunto con otras células no transfectadas (no visibles). Debido al restringido espacio, la célula forma diversas prolongaciones que traspasan el espacio entre las células vecinas en una forma irregular. Se observan finas prolongaciones en la superficie de la célula (*filopodias*). La barra en la esquina superior representa 15 µm.

La imagen muestra una reconstrucción tridimensional de una célula de cultivo tipo Hek. La reconstrucción se realizó con 48 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. La célula fue transfectada genéticamente con la proteína fluorescente (*farnesylated*-EGFP) que se une a la membrana interior de esta célula. La célula transfectada crece en conjunto con otras células no transfectadas (no visibles). Debido al restringido espacio, la célula forma diversas prolongaciones que traspasan el espacio entre las células vecinas en una forma irregular. Se observan finas prolongaciones en la superficie de la célula (*filopodias*). La barra en la esquina superior representa 15 µm.

La imagen muestra una reconstrucción tridimensional del complejo pineal de pez cebra (*Danio rerio*). La reconstrucción se realizó con 76 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. Las células del complejo pineal expresan una versión de la proteína fluorescente verde unida a membrana (GFP-Gap43) bajo el control de un promotor específico. En esta vista dorsal se puede observar la complejidad de formas celulares presentes en esta estructura, así como la salida de manojos de proyecciones axones hacia ambos lados del complejo pineal, en un embrión *in vivo* de 24 horas postfertilización.



La imagen muestra una reconstrucción tridimensional de una célula de cultivo tipo Hek. La reconstrucción se realizó con 35 imágenes tomadas en el plano z con un microscopio confocal de fluorescencia. La célula fue transfectada genéticamente con la proteína fluorescente (*farnesylated*-EGFP) que se une a la membrana interior de esta célula. La célula transfectada crece en conjunto con otras células no transfectadas (no visibles). Debido al restringido espacio, la célula forma diversas prolongaciones que traspasan el espacio entre las células vecinas en una forma irregular. Se observan finas prolongaciones en la superficie de la célula (filopodias). La barra en la esquina superior representa 15 µm.



La imagen muestra una reconstrucción tridimensional de una microglia (célula neuroglial) en un cultivo primario de hipocampo de rata. La reconstrucción se realizó con 50 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. La membrana de la célula fue marcada con el fluróforo $DiIC_{18}$, observándose un núcleo pequeño con gran número de prolongaciones largas y ramificadas (dendritas). La barra en la esquina superior representa 10 µm.

La imagen muestra una reconstrucción tridimensional de dos células de cultivo tipo HeLa en condiciones normales. La reconstrucción se realizó con 56 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. Las membranas de las células fueron marcadas con el fluoróforo $DiIC_{18}$. La célula superior está preparando la división celular (*mitosis*) mientras la célula inferior se encuentra en la interfase (G1). Ambas células están conectadas mediante varios *nanotubos*. A través de estos *nanotubos*, las células intercambian diferentes organelos o componentes de las membranas lipídicas. Se observan finas prolongaciones en la superficie de ambas células (*filopodias*). La barra en la esquina superior representa 10 µm.

Comparación entre reconstrucciones tridimensionales de células HeLa sin (imagen superior) y con (imagen inferior) pretratamiento de deconvolución. Las reconstrucciones se realizaron con 49 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. Las membranas de las células fueron marcadas con el fluoróforo $DiIC_{18}$. La deconvolucion permite realizar una reconstrucción con buena resolución, definiendo mejor estructuras celulares (membrana, organelos intracelulares, filopodias). La barra en la esquina superior representa 10 µm.

Ejemplo de una reconstrucción tridimensional de una célula necrotica (muerta) de cultivo tipo HeLa con 2 filtros morfológicos. La reconstrucción se realizó con 30 imágenes tomadas en el eje z con un microscopio confocal de fluorescencia. La membrana de la célula fue marcada con el fluróforo $DiIC_{18}$. El filtro morfológico separa estructuras pequeñas de las grandes.. La barra en la esquina superior representa 10 µm.