Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Sistema de Visualización para un Tomógrafo de Rayos X

Tesis que presenta

María del Rosario Martínez Gómez

para obtener el Grado de

Maestra en Ciencias

en la Especialidad de

Ingenieria Eléctrica

Director de la Tesis

Dr. Luis Gerardo de la Fraga

México, D.F. Diciembre 2006

Resumen

En esta tesis se desarrolló un sistema de software que permite la reconstrucción de volúmenes a partir de imágenes obtenidas con un tomógrafo de rayos X. Éste, también permite la visualización de superficies de los volúmenes reconstruidos. Aunque se podría pensar que el único software involucrado en el problema de reconstrucción es el que que hace funcionar el tomógrafo, se mostrará que son cinco las partes de éste, no lígadas al hardware, necesarias para resolver el problema completo de reconstrucción: (1) adquirir las proyecciones, (2) centrar las proyecciones, (3) reconstruir el objeto, (4) segmentar las densidades del objeto reconstruido y, finalmente, (5) visualizar las isosuperficies de las densidades segmentadas. La parte de reconstrucción fue tomada de Xmipp, un software viejo diseñado para la reconstrucción en 3D de macromoléculas biológicas, la segmentación fue realizada usando el algoritmo de "k-means" y la visualización fue realizada por medio de la técnica de aplanamiento ("splatting", en inglés). Además se compara el aplanamiento con otras dos técnicas de visualización: voxels simples y mallas deformables. La mayor parte de los componentes de software fueron construidos en C, C++ y Perl. La interfaz gráfica fue construida en C++ con las bibliotecas Qt y OpenGL.

Abstract

A software tool-box has been developed, which allows, not only the volume reconstruction from X ray tomography images, but also the visualization of reconstructed isosurfaces. Although it may be thought that the software needed is tightly linked to the hardware, we show that five different hardware-independent software components are needed for the whole process: (1) acquisition, (2) projection's centering, (3) reconstruction, (4) isosurfaces segmentation and (5) visualization. The reconstruction component was taken from Xmipp, an old software for 3D reconstruction of biological macromolecules, segmentation was developed using k-means algorithm and visualization was built using the splatting technique. In addition, we compare splatting with another two surface visualization techniques such as simple voxels and deformable simplex meshes. Most of the software components were developed in C, C++ and Perl. The interface was developed in C++ using Qt and OpenGL libraries.

Agradecimientos

Rosario, *mi arbolito con estrellas*, gracias por tu lucha, por tu valor, por ahuyentar a los demonios, pero sobre todo por amarme tanto.
Víctor, gracias por confiar en mí y por ayudarme a alcanzar mis sueños.

A usted, mi director de tesis, Doc. Luis Gerardo de la Fraga, por su paciencia, por creer en mí, por apoyarme y brindarme la oportunidad de concluir mis estudios de maestría.

A mis hermanos Victoria y Juan Carlos, porque siempre me esperan y entienden.

A mis estrellas amigas, Doc. Tonatiuh Matos, M.C. Víctor Márquez, por escucharme, por creer en mí y ayudar a la conclusión de este proyecto de vida.

A todos mis amigos (gente), que hacen que mi vida sea tan maravillosa. Especialmente a Edgar, Luis Alberto y Abril, quienes han estado conmigo no sólo en los momentos más difíciles sino también en los más irrelevantes.

Al CINVESTAV IPN, a CONACyT y al proyecto 45306 de CONACyT, por otorgarme el apoyo para mis estudios y la elaboración de esta tesis.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Reconstrucción tridimensional y tomografía

La tomografía es la proyección de la sección transversal de un objeto, obtenida a partir de datos provenientes de diferentes direcciones [1]. La palabra *tomografía* se deriva de dos palabras griegas: *tomos* que significa corte o sección y *graphein* que significa escribir. El objetivo de la tomografía es el de permitir la visualización de la estructura interna de un objeto en estudio sin destruirlo. Ésta, se basa en la propiedad de que la densidad de la materia de un objeto puede absorber los rayos X.

Un tomógrafo de rayos X genera imágenes del objeto escaneado, a partir de la información obtenida al medir la atenuación de rayos X a lo largo de un número finito de líneas que pasan a través de éste. Al conjunto de imágenes obtenidas desde diferentes ángulos de rotación, se les conoce como *proyecciones* y con ellas se puede realizar la reconstrucción 3D del objeto en estudio.

La reconstrucción tridimensional de un objeto es muy útil ya que se puede conocer información acerca de la naturaleza y estructura de los materiales que conforman el interior de éste. Los problemas asociados con la reconstrucción de imágenes a partir de proyecciones han ido incrementando tanto en campos científicos cómo técnicos. Los más importantes se presentan en el campo de la medicina, al querer obtener la distribución de densidad dentro del cuerpo humano a partir de la incidencia de múltiples rayos X; en el campo de la biología, al reconstruir la estructura molecular de una bacteria y en el campo de la astronomía, al reconstruir la estructura de rayos X de una supernova a partir de los datos recolectados por cohetes enviados fuera de la atmósfera de la tierra [2].

Actualmente, existen diversos métodos para reconstruir un objeto tridimensional a partir de sus imágenes de proyección, los cuales se agrupan principalmente en dos grandes familias[3]:

- *Métodos directos:* Utilizan la relación matemática entre la densidad del objeto y sus proyecciones (por lo tanto trabajan en el espacio rea)l. Esta relación puede formularse como un sistema de ecuaciones lineales; pero en la práctica el número de variables es tan grande que imposibilita la solución directa a este problema. Una opción son las, técnicas de aproximación iterativas, que tratan de llegar a una solución mediante la minimización de la discrepancia entre las proyecciones de un volumen estimado (que se actualiza en cada iteración) y las proyecciones experimentales.

- *Métodos de Fourier:* Hacen uso del *Teorema de la Sección Central*. Este teorema establece que la transformada de Fourier de una imagen de proyección 2D es exactamente una sección central de la transformada de Fourier 3D del objeto. De tal manera que, podemos construir la transformada de Fourier 3D del objeto mediante el ensamblaje de las distintas secciones centrales que proponen las transformadas de Fourier de un número de proyecciones 2D del objeto. Finalmente, la distribución 3D de densidad del objeto se obtiene finalmente calculando la transformada inversa de Fourier 3D.

La diferencia principal de estas dos familias es el espacio donde recontruyen al objeto, el cual puede ser real (dominio del espacio) o recíproco (dominio de Fourier). Un conjunto de métodos que utiliza el dominio real para reconstruir un objeto son los de *retroproyección*, los cuales se han propuesto en versiones analógicas y digitales por un gran número de autores, son muy estudiados porque indican las partes principales de los más sofisticados procesos de reconstrucción.

1.2. Métodos de visualización

Tradicionalmente, un objeto (volumen) se representa como una malla poligonal (malla conformada por polígonos) para su manipulación y dibujado [4]; sin embargo conforme la complejidad del objeto aumenta se vuelve menos eficiente representar éste como malla poligonal u otro nivel de representación más alto (ya que se tiene que estar cambiando continuamente la información de conectividad para cada polígono si el observador cambia de posición). Dibujar volúmenes a partir de puntos es una de las técnicas que ha recibido gran atención en los últimos años porque permiten dibujar objetos con una alta complejidad sin necesidad de cambiar información de conectividad.

En 1985, casi al mismo tiempo que Lorensen y Cline describieron el algoritmo de dibujado de volumen *encajamiento de cubo (marching cubes 1987)* [5], Levoy y Whitted presentaron su primer trabajo de dibujado basado en puntos. En el que afirman que la coherencia provista por el dibujado de volumen basado en mallas de triángulos, polígonos u otro nivel de representación más alto disminuye conforme la complejidad de un objeto aumenta, también denotan que una superficie de un objeto puede ser representada por un conjunto de puntos $0-$dimensionales, siempre y cuando el conjunto sea diferenciable, estimando la tangente y la normal al plano a partir de un pequeño vecindario de puntos, de tal manera que la reconstrucción de la superficie se puede realizar estimando el alcance del número de puntos por el número de los puntos proyectados al área del núcleo de un filtro en cada píxel [6].

Sin embargo el dibujado de objetos basados en puntos se convirtió en una de las técnicas de representación más importantes hasta una década después, cuando surgió la necesidad de dibujar de manera eficiente objetos complejos capturados con escaners 3D. Una gran variedad de métodos de dibujado de volumen se han propuesto. Hasta ahora, el método más aceptado es el que implanta la *linealización por trozos de la superficie con aplanados*[7].

Usamos el término *aplanado* para referirnos a la palabra en inglés *splat* [8]. De la misma forma usamos *aplanamiento* para la palabra inglesa *splatting*.

1.3. Objetivo

Esta tesis tiene como objetivo desarrollar el sistema de software que permite la reconstrucción de volúmenes a partir de imágenes obtenidas con un tomógrafo de rayos X. Éste, también permite la visualización de superficies de los volúmenes reconstruidos por medio de la técnica de visualización llamada aplanamiento.

El proyecto integra diversas herramientas disponibles y algunas mas que se desarrollarón, con el fin de explotar el tomógrafo que se encuentra en el Departamento de Física del CINVESTAV.

1.4. Organización de la tesis

El resto de esta tesis está organizada de la siguiente manera:

- En el capítulo 2, desarrollamos todo lo referente a la reconstrucción de volúmenes a partir de proyecciones. Explicamos el proceso de adquisición de proyecciones de un objeto, resolvemos por medio de software el problema de centrado de proyecciones, reconstruimos el objeto comparando dos métodos de reconstrucción, retroproyección filtrada y ART y describimos la implantación de un algoritmo de segmentación basado en agrupamiento.

- En el capítulo 3, revisamos los conceptos fundamentales de las técnicas de dibujado de superficies basadas en puntos, introducimos el término y el método de *aplanamiento*, además describimos un algoritmo de dibujado de superficies que utiliza primitivas de OpenGL para representar *aplanados*.

- En el capítulo 4, hacemos pruebas con los algoritmos desarrollados.

- Concluimos la tesis de manera general y sugerimos el trabajo a futuro en el capítulo 5.

Capítulo 2

Reconstrucción de volúmenes

En este capítulo de la tesis explicamos como adquirir las proyecciones de un objeto utilizando el tomógrafo de rayos X del Cinvestav, resolvemos por medio de software el problema de centrado de proyecciones, revisamos la teoría básica de la reconstrucción de volúmenes a partir de proyecciones, reconstruimos un volumen artificial con el objetivo de comparar dos métodos de reconstrucción y describimos la implantación de un algoritmo de segmentación que usa K-means *como motor de agrupamiento.*

Para resolver el problema de reconstrucción de volúmenes a partir de imágenes obtenidas con un tomógrafo de rayos X y permitir la visualización de estas reconstrucciones, dividimos en cinco partes este problema: (1) adquirir las proyecciones, (2) centrar las proyecciones, (3) reconstruir el objeto, (4) segmentar las densidades del objeto reconstruido y, finalmente, (5) visualizar las isosuperficies de las densidades segmentadas.

2.1. Adquisición

La fase de adquisición consiste en obtener imágenes, llamadas proyecciones, de un objeto de densidad no homogénea al que llamamos *muestra* mediante el uso de un tomógrafo de rayos X. El tomógrafo con el que se cuenta está en el Departamento de Física del Cinvestav, mostrado en la Fig. 2.1, consta de cuatro partes principales: una cámara CCD (del término inglés *"charge-coupled device"*), una pantalla fosforescente, una fuente de rayos X y un plato giratorio. El proceso de adquisición comienza cuando la fuente genera un flujo de rayos X que atraviesa la muestra en línea recta, la cantidad de fotones del rayo que inciden perpendicularmente en la pantalla fosforescente son registrados para conformar una proyección. Algunos de los fotones que pasan a través del objeto serán absorbidos por la muestra o serán desviados en su interacción con ella, por lo que el brillo correspondiente a ese flujo de rayos será menor en la pantalla fosforescente. Los rayos que no sufren atenuación, es decir que no cruzan el objeto, serán los que causan el mayor brillo en la pantalla. Mediante el proceso de adquisición se puede capturar imágenes correspondientes a la densidad de la muestra. Las imágenes son recolectadas usando la cámara CCD, debido a que ésta se acopla a la pantalla fosforescente por medio de un dispositivo óptico.

Para adquirir las proyecciones nos basamos en la geometría de eje único, la cual consiste en rotar la muestra de estudio en torno a un eje fijo. El ángulo de rotación asociado a

Figura 2.1: Fotografía del tomógrafo de rayos X que existe en el Departamento de Física del Cinvestav.

la muestra varía en incrementos pequeños, de esa forma en cada orientación la cámara CCD toma una fotografía. En la Fig. 2.2 se ilustra ésta geometría, el eje z (localizado en el centro del plato giratorio) es el eje sobre el que se rota al objeto.

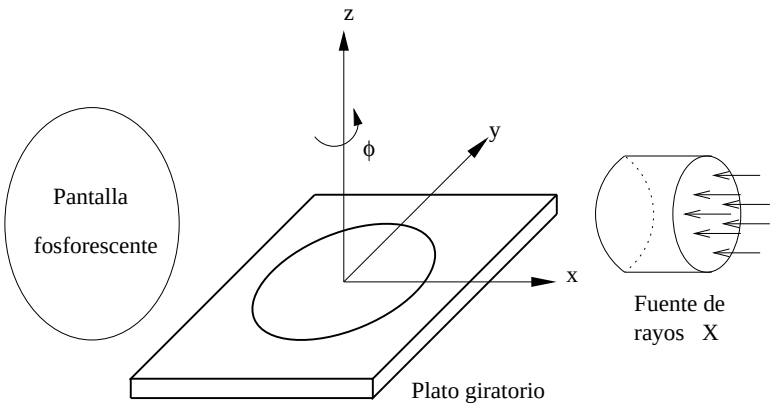


Figura 2.2: Geometría de eje único. La muestra se coloca en el centro del plato giratorio. ϕ es el ángulo en que se rota el objeto sobre el eje z. Los rayos X atraviesan la muestra y chocan con la pantalla fosforescente, y su atenuación depende de las densidades de la muestra.

2.2. Métodos de reconstrucción

La reconstrucción tridimensional de un objeto es muy útil ya que se puede conocer información acerca de la naturaleza y estructura de los materiales que conforman el interior de éste, sin necesidad de destruirlo. Desde que el primer tomógrafo fue construido [1], los problemas asociados con la reconstrucción a partir de proyecciones son comúnmente abordados en el campo de la medicina.

Un tomógrafo produce proyecciones generadas a partir de la información obtenida en la medición de la atenuación de un haz de rayos X a lo largo de una dirección. El tomógrafo con que se cuenta para el desarrollo de esta tesis entrega proyecciones conformadas por los valores de la atenuación de un haz de rayos, que pasan a través de la muestra (ver Fig. 2.3). De tal manera que, cada proyección corresponde a una matriz bidimensional donde cada uno de sus elementos es un número en punto flotante simple. Para propósitos

de visualización, esta matriz se transforma linealmente a valores entre 0 y 255, lo que produce una imagen bidimensional en tonos de gris.

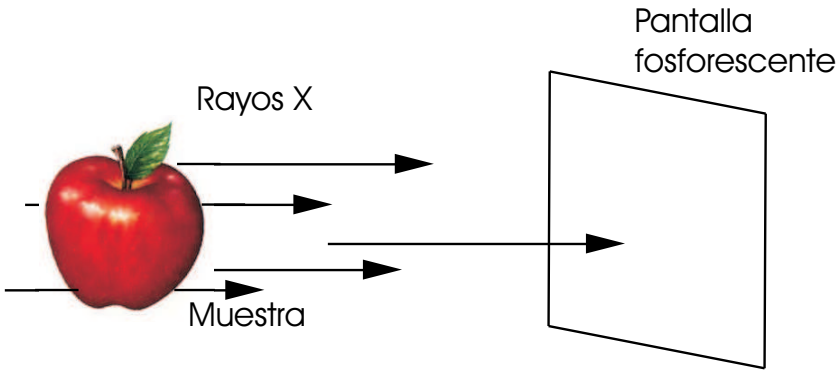


Figura 2.3: Un haz de rayos se atenúan al atravesar la muestra. La iluminación, provocada por el choque de los fotones contra la pantalla fosforescente, se digitaliza usando la cámara CCD.

Con el objetivo de explicar el proceso usado para generar una proyección, reducimos el problema de reconstrucción tridimensional a dos dimensiones mediante la reconstrucción de una sección transversal de la muestra. En otras palabras, suponemos que el objeto escaneado es bidimensional y que su densidad se define como una función $f : \Re^2 \rightarrow \Re$, que tiene como parámetro la pareja de números x, y, los cuales representan las coordenadas espaciales en el plano definido por el objeto.

Una proyección unidimensional $g_\phi(s)$ es el conjunto de valores de las integrales de línea de los rayos que pasa a través de la muestra en la dirección ϕ, como se ilustra en la Fig. 2.4, donde l representa la trayectoria de un rayo, ϕ es el ángulo que forma l con el eje x y s es la distancia que existe entre del rayo al origen.

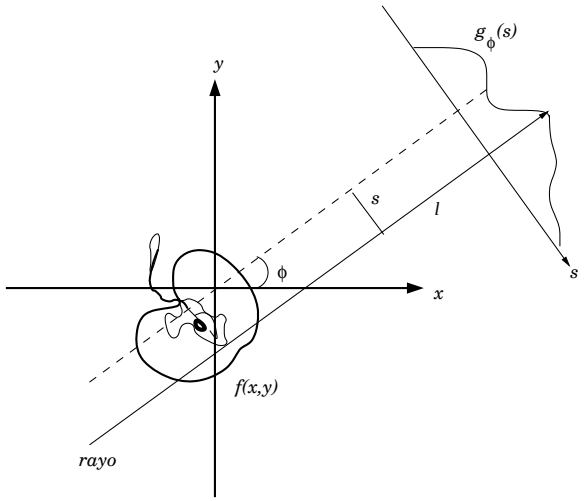


Figura 2.4: Los rayos pasan a través del objeto bidimensional $f(x, y)$, a lo largo de la línea l, a una distancia s del origen, a un ángulo de inclinación ϕ con el eje de las x. El vector unitario $[\cos\phi, \operatorname{sen}\phi]$ indica la dirección de los rayos. El vector unitario $[-\operatorname{sen}\phi, \cos\phi]$ indica la perpendicular a la dirección de los rayos.

A la colección de todas las proyecciones, cada proyección tomada a un ángulo ϕ distinto,

se le conoce como la *Transformada de Radon* de la muestra, la cual se define como

$$g(\phi, s) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\delta\left(s - x \operatorname{sen}\phi + y\cos\phi\right) dxdy, \tag{2.1}$$

donde δ es la función delta de Dirac [9]. Esta ecuación representa la integral de línea a lo largo de un rayo que se encuentra en la posición s orientado en un ángulo ϕ sobre el objeto bidimensional $f(x, y)$.

A pesar de que hasta ahora hemos revisado como se genera una proyección, nuestra finalidad es realizar el proceso inverso, es decir, a partir de una colección de proyecciones obtenidas en la fase de adquisición queremos reconstruir la densidad de la muestra en estudio. Para llevar a cabo esta tarea los sistemas tomográficos hacen uso de la *Transformada Inversa de Radon*, debido a que Radon estableció que, si se cuenta con un número suficiente de proyecciones, cada proyección tomada en un ángulo distinto, se puede reconstruir la densidad de un objeto con dicha ecuación [1]. Sin embargo, implantar un algoritmo de reconstrucción usando esta transformada no resulta muy práctico desde el punto de vista computacional, ya que se requiere de un arreglo continuo de proyecciones. Por esta razón se han creado un conjunto de métodos de reconstrucción alternativos, entre los más conocidos están los de *retroproyección*. Éstos, se han propuesto en versiones analógicas y digitales por varios autores desde 1974 y se estudian principalmente porque indican los procesos básicos de los métodos más sofisticados.

2.2.1. Retroproyección

Los métodos de retroproyección aproximan la reconstrucción de la densidad de un objeto bidimensional con la expresión

$$f(x, y) = \int_0^{\pi} g(x\cos\phi + y \operatorname{sen}\phi, \phi)d\phi. \tag{2.2}$$

El desarrollo matemático que sustenta esta ecuación se basa en el *teorema de la sección de Fourier* [1]. Dado que nuestro objetivo es el de explicar en términos computacionales como funciona un algoritmo que implanta el método de retroproyección daremos una explicación más intuitiva de como funciona éste, omitiendo el desarrollo matemático.

El método de retroproyección también es conocido como de *suma* debido a que trabaja con el principio básico de la tomografía tradicional. Esto es, para cada proyección el valor de la atenuación es expandido (retroproyectado) uniformemente a lo largo de la trayectoria del haz de rayos, si ésto se hace para varias proyecciones, la suma de la contribución de los valores de atenuación de cada proyección corresponderá a las características presentes en la estructura del objeto en estudio [10]. De tal manera que cuando el número de proyecciones aumenta la estructura de la densidad se convierte más definida.

Para reconstruir la densidad de un objeto tridimensional a partir de sus proyecciones bidimensionales, el algoritmo retroproyecta uniformemente los valores almacenados en las matrices de proyección en la matriz tridimensional que contendrá la densidad del objeto. Como se muestra en el procedimiento 1, la retroproyección en 3D necesita hacer un paso adicional rotando cada proyección antes de sumar su respectiva contribución, ésto se realiza debido a que cada proyección corresponde a una orientación distinta del objeto. La matriz de rotación usada por el algoritmo implantado en este trabajo es

$$M = R_x(\alpha) \cdot R_y(\beta) \cdot R_x(\gamma), \tag{2.3}$$

donde R_x es una rotación alrededor del eje x, R_y es una rotación alrededor del eje y, como se muestra a continuación

$$M = \begin{bmatrix} \cos\alpha & -\operatorname{sen}\alpha & 0 \\ \operatorname{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \operatorname{sen}\beta \\ 0 & 1 & 0 \\ -\operatorname{sen}\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\operatorname{sen}\gamma & 0 \\ \operatorname{sen}\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

Aunque la retroproyección es simple, presenta un problema muy importante descrito en la siguiente sección, por lo que ha sido necesario complementar el método, este ajuste es conocido como retroproyección filtrada.

2.2.2. Retroproyección filtrada

El principio con el que trabaja la retroproyección es fácil de implantar, sin embargo produce reconstrucciones de baja calidad. La retroproyección filtrada (Filtered Backprojection) es el método que aprovecha la simplicidad de la retroproyección y mejora sus resultados.

Con el fin de explicar brevemente porque la retroproyección genera objetos de baja calidad abordaremos el caso de la reconstrucción de la densidad de un punto, suponiendo que el punto tiene cierta densidad capaz de atenuar un rayo que pase a través de él, tal como se ilustra en la Fig. 2.5. Donde cada línea representa la atenuación registrada en cada una de sus proyecciones respectivamente.

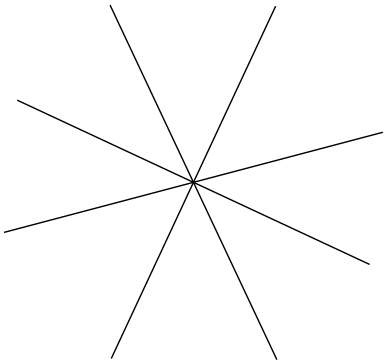


Figura 2.5: Reconstrucción de un punto simple usando el método de retroproyección. Una línea representa la atenuación registrada en una proyección.

Conforme se incrementa el número de proyecciones, la reconstrucción del punto se parece a una distribución de densidad proporcional a $\frac{1}{r}$, donde r es la distancia que existe entre una proyección y el punto [2]. Esto sucede debido a que la superposición de un número de líneas equidistantes a través de un punto en común, es equivalente a la rotación de una línea alrededor de dicho punto. Por lo tanto la contribución de una proyección se distribuye durante la rotación, sobre la longitud de la circunferencia $2\pi r$, lo que provoca que la reconstrucción sea una imagen brillante que tiene por centro el punto original. El método de retroproyección filtrada disminuye considerablemente la contribución $\frac{1}{r}$ apoyado en el Teorema de la Sección Central, el cual establece que la

Procedimiento 1 Retroproyección

Entrada: lista que contiene todas las proyecciones donde cada proyección tiene asociado un ángulo distinto y tamaño de la matriz de proyección (las matrices son cuadradas)

Salida: matriz tridimensional que contendrá la estructura de la densidad del objeto

1: $dim \leftarrow$ tamaño de la proyección
2: $proj \leftarrow$ crear matriz bidimensional de tamaño $dim \times dim$
3: $vol \leftarrow$ crear matriz tridimensional de tamaño $dim \times dim \times dim$
4: $A \leftarrow$ crear matriz de rotación de tamaño 3×3 {esta matriz corresponde a la de rotación calculada en la ecuación 2.3}
5: Se asigna un cero a cada elemento de la matiz tridimensional vol
6: **for** $0 \leq proActual <$ número total de proyecciones **do**
7: $proj \leftarrow$ leer la proyección número $proActual$
8: Leer el ángulo de proyección {el ángulo de rotación debe estar en términos de los ángulos de Euler [11]}
9: Calcular la matriz A {esta matriz se calcula como se muestra en la ecuación 2.4}
10: $dim2 \leftarrow dim/2$
11: $radio2 \leftarrow dim2 * dim2$
12: **for** $0 \leq i < dim$ **do**
13: $z \leftarrow -i + dim2$
14: $z2 \leftarrow z * z$
15: **for** $0 \leq j < dim$ **do**
16: $y \leftarrow j - dim2$
17: $y2 \leftarrow y * y$
18: $z2plusy2 \leftarrow z2 + y2$
19: $x \leftarrow -dim2$
20: $xp \leftarrow x * A[0][0] + y * A[1][0] + z * A[2][0] + dim2$
21: $yp \leftarrow x * A[0][1] + y * A[1][0] + z * A[2][0] + dim2$
22: **for** $0 \leq k < dim, xp+ = A[0][0], yp+ = A[0][1], x++$ **do**
23: Calcular $x2 = x * x$
24: **if** $x2 + z2plusy2 > radio2$ **then**
25: continue
26: **end if**
27: **if** xp y yp son coordenadas validas dentro de la proyección **then**
28: $vol[i][j][k]+ = proj[yp][xp]$
29: **end if**
30: **end for**
31: **end for**
32: **end for**
33: **end for**
34: Regresar vol

transformada de Fourier de una proyección $2D$ es un plano centrado en el origen del espacio recíproco [12].

En este trabajo se implantó el algoritmo de la retroproyección filtrada realizando los siguientes pasos:

- Paso 0: Creamos un filtro
- Paso 1: Calculamos la transformada de Fourier para cada proyección
- Paso 2: Filtramos las proyecciones, multiplicando el filtro diseñado con cada una de las proyecciones transformadas
- Paso 3: Calculamos la transformada inversa de Fourier de cada proyección
- Paso 4: Retroproyectamos la atenuación resgistrada en las proyecciones en el dominio del espacio

A pesar de que este algoritmo es elegante, no existe una fórmula exacta para diseñar el filtro del paso 0, provocando que las versiones implantadas con diferentes filtros arrojen resultados distintos. En esta tesis utilizamos dos filtros: rampa y Weiner.

2.2.3. Filtro rampa

El filtro rampa contemplando una dimensión es mostrado en la Fig. 2.6(a), es descrito como la diferencia entre un bloque Fig. 2.6(b) y un triángulo Fig. 2.6(c), en el espacio de Fourier hace cero muchas frecuencias altas y deja pasar las bajas. Este filtro debe ser limitado por la frecuencia de Nyquist [12] ya que no se comporta de manera estable cuando se le aplica la transformada inversa de Fourier.

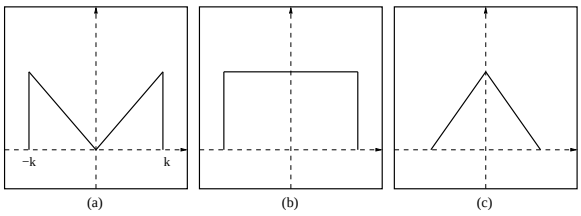


Figura 2.6: Descomposición del filtro rampa (a)diferencia de un bloque (b) y un triángulo (c).

Debido a que en este trabajo se manipulan imágenes (proyecciones), usamos el filtro rampa en dos dimensiones, representado en la siguiente ecuación

$$q(u,v) = \begin{cases} |r|, & \text{si } r \leq k, \\ 0, & \text{si } r > k, \end{cases} \tag{2.5}$$

donde $r = \sqrt{u^2 + u^2}$, k es una constante proporcional a la frecuencia de Nyquist y u, v son las frecuencias espaciales.

2.2.4. Filtro Wiener

Como mencionamos previamente, el método de retroproyección produce una versión de baja calidad de la reconstrucción deseada. En ausencia de ruido, la reconstrucción puede ser mejorada filtrando las proyecciones con el filtro rampa, desafortunadamente cuando existe ruido este filtro amplifica las frecuencias altas (ruido).

El filtro Wiener mejora la reconstrucción y no amplifica el ruido existente en las proyecciones. El diseño de éste requiere conocer la estadística del ruido y del objeto. La siguiente ecuación expresa dicho filtro en el dominio de Fourier, contemplando ruido de tipo blanco gaussiano

$$A(u,v) = \frac{|f|2\pi\alpha(\mathrm{S})}{2\pi\alpha(\mathrm{S}) + |f|(\alpha^2 + 4\pi^2 f^2)^{\frac{3}{2}}}, \tag{2.6}$$

donde: $f = \sqrt{u^2 + v^2}$, $\alpha = 0.5$, $\mathrm{S} = 60$, u, v son las frecuencias espaciales [13].

2.3. Técnicas de reconstrucción algebráica (ART)

Otra forma de reconstruir un objeto a partir de sus proyecciones es a través del uso de las técnicas de reconstrucción algebraica, ART (del término inglés *"techniques of reconstruction algebraic"*). Aunque en este trabajo utilizamos el algoritmo de ART implantado por un software diseñado para la reconstrucción en 3D de macromoléculas biológicas (Xmipp [14]), creemos conveniente explicar de manera muy general en que consisten éstas.

Las técnicas de reconstrucción algebraica contienen procesos iterativos que tratan de resolver el *problema de reconstrucción discreto*. Éste, consiste en estimar un vector-imagen x, a partir de: un vector de muestras y, una matriz de proyección R y un vector de error e. Su objetivo es el de satisfacer el criterio de optimización $y = Rx + e$ [2, cap. 6], es decir, ART trata de converger a un estimado del objeto, resolviendo el problema de encontrar el coeficiente de atenuación lineal de cada punto en la imagen. Debido a que cada integral de rayo provee una ecuación, el problema es visto como la resolución de un conjunto de ecuaciones simultáneas. La suma de los coeficientes de atenuación a lo largo del rayo es igual a la medida de absorción y el número de incógnitas en este conjunto de ecuaciones es igual al número de pixeles en la imagen [2, cap. 11].

2.3.1. Construcción de un fantasma

Con el objetivo de comprobar que los algoritmos de ART y retroproyección filtrada funcionan de manera adecuada sin considerar el ruido producido naturalmente en la fase de adquisición construimos un fantasma, que es un volumen sintético de objetos descritos matemáticamente. En otras palabras, un fantasma se crea al imponer las posiciones, orientaciones, tamaño y densidad de un número de objetos elementales de la manera que más nos convenga, sirven principalmente para investigar individualmente varios fenómenos que no pueden ser separados físicamente [2].

El fantasma diseñado está compuesto de dos esferas y dos elipsoides. Una esfera envuelve a todos los demás elementos, simulando una densidad que contiene otras en el interior. La Fig. 2.7 muestra los elementos del fantasma. La esfera D_1 tiene una densidad de 0.2, su centro está localizado en $(0, 0, 0)$ y su diámetro es de 57 pixeles. La elipsoide D_2 tiene

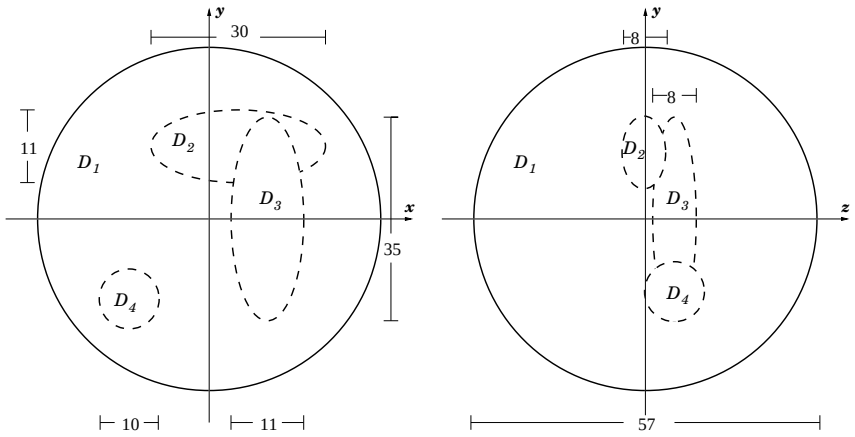


Figura 2.7: Diagrama del fantasma usado. Se pueden observar cuatro densidades (D_1, D_2, D_3, D_4). Las unidades están dadas en pixeles.

una densidad de 0.56, su centro está localizado en $(10, 25, 0)$ y sus diámetros son $30, 11, 8$ pixeles en x, y, z respectivamente. La elipsoide D_3 tiene una densidad de 0.86, su centro está localizado en $(20, 0, 10)$ y sus diámetros son $11, 35, 8$ pixeles en x, y, z respectivamente. La esfera D_4 tiene una densidad 0.7, su centro está localizado en $(-28, -28, 10)$ y su diámetro es de 10 pixeles.

Para crear el fantasma, nos hemos auxiliado del programa *phantom* de Xmipp [14], el cual nos entrega un volumen como una matriz tridimensional de dimensiones $m \times m \times m$ voxels. En este programa se pueden crear tres tipos de primitivas: cilindros, esferas y elipsoides. Para cada objeto se especifica: número de objeto, tipo, densidad, centro, radio y rotaciones (en radianes) sobre los ejes x, y, z. A continuación se muestra el contenido del archivo que el programa *phantom* necesita como argumento, la densidad y dimensiones del volumen se especifican en la línea 4, e indica primitivas tipo elipsoide, de esta forma las esferas D_1 y D_4 son creadas al especificar radios iguales en los 3 ejes (líneas 7 y 10). La línea 9 describe los parámetros del objeto D_3, asignando una densidad de 0.86, centro $(20, 0, 10)$, diámetros $35, 11, 8$ pixeles en x, y, z respectivamente y una rotación de 1.5827 rad sobre el eje z.

```
1.  # Phantom "phanprueba.spi" description file, (generated with phantom)
2.  # General Volume Parameters:
3.  #      Xdim      Ydim      Zdim   Density
4.        64        64        64    0.000000e+00
5.  # Feature Parameters:
6.  #Fea_N  cSph_Cyl cOr_And fDensity  iX_Cent iY_Cent iZ_Cent iX_Rad iY_Rad iZ_Rad iAlpha   iBeta    iGamma
7.  0001       e       o   0.200000e+00   0.000   0.000   0.000   57.00  57.00  57.00  0.00000  0.00000  0.0000
8.  0002       e       o   0.556666e+00   10.00   25.00   0.00    30.00  11.00  8.00   0.00000  0.00000  0.0000
9.  0003       e       o   0.855500e+00   20.00   0.000   10.00   35.00  11.00  8.00   0.00000  0.00000  1.5827
10. 0004       e       o   0.700000e+00   -28.00  -28.00  10.00   10.00  10.00  10.00  0.00000  0.00000  0.0000
```

2.3.2. Reconstrucción del fantasma

Debido a que los métodos de reconstrucción necesitan las proyecciones del objeto al que van a reconstruir y a que nosotros queremos reconstruir el fantasma de la sección anterior, generamos 72 proyecciones de éste (suponiendo que el ángulo asociado a la rotación de la muestra varía cada 5°) por medio del programa Xmipp. Los siguientes algoritmos se utilizaron para reconstruir la densidad del fantasma (modelo original):

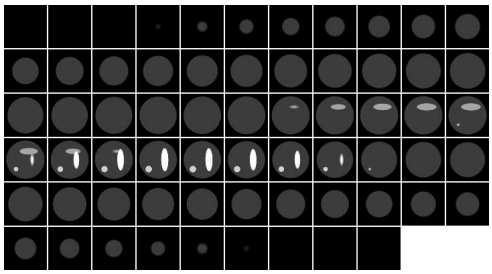


Figura 2.8: Visualización por medio de rebanadas del fantasma especificado en el archivo *phanprueba.spi*. Cada rebanada corresponde a un plano en el eje z. Un color de gris representa una densidad.

- Retroproyección
- Retroproyección Filtrada con rampa
- Retroproyección Filtrada con Weiner
- ART

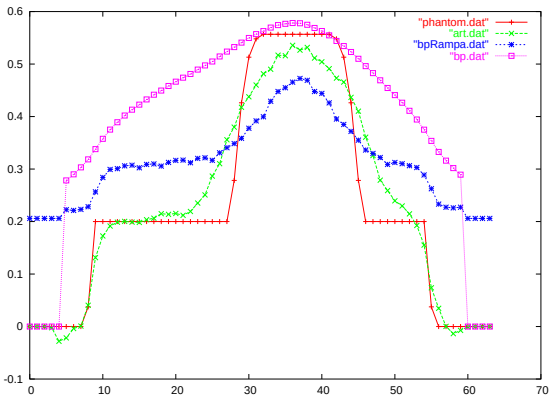


Figura 2.9: Visualización del perfil de la rebanada 30 sobre la línea 15 del fantasma de prueba. La línea simple representa el fantasma original, la reconstrucción hecha con retroproyección se representa por la línea con cuadros, la reconstrucción realizada con retroproyección filtrada es mostrada en la línea con asteriscos, la mejor reconstrucción es la realizada con ART representada por la línea con cruces.

Para conocer que algoritmo aproxima más su resultado al modelo original graficamos los cambios de densidad de cada volumen reconstruido, a lo largo de la línea 15, en el plano 30 del eje z. Las gráficas resultantes se muestran en la Fig. 2.9. La línea simple representa el fantasma original, la reconstrucción hecha con retroproyección se representa por la línea con cuadros, esta reconstrucción mejora cuando se aplica el filtro rampa (línea con asteriscos); sin embargo este filtro disminuye la amplitud dinámica, que se define como el intervalo de valores que existen entre el mínimo y máximo valor del volumen. Por lo tanto, la mejor reconstrucción se realiza con ART, la cual se representa por la línea con cruces.

2.4. Centrado de proyecciones

En la tomografía computarizada una gran variedad de defectos pueden presentarse en las imágenes de proyección, propagando errores al objeto reconstruido. Uno de los defectos más importantes es causado debido al impreciso conocimiento de la localización del centro de rotación del plato giratorio del sistema tomográfico [10]. Ésto sucede dado que es difícil determinar el origen de coordenadas del objeto escaneado o por variaciones en el centro del plato causadas por un mecanismo imperfecto, además de que la colocación exacta del objeto en el centro de éste es difícil de obtener.

Es posible establecer el origen de coordenadas del sistema tomográfico por medio de un proceso de prueba y error, el cual consiste en rotar o transladar el objeto mientras los centros (del plato y del objeto) no se encuentren exactamente alineados; sin embargo este proceso resulta ser lento, por lo que es preferible alinear las proyecciones vía software.

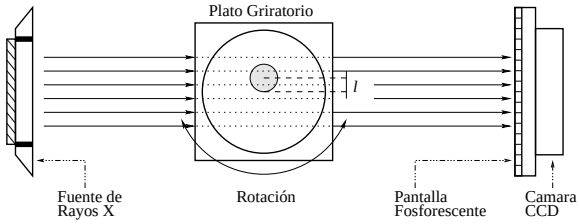


Figura 2.10: El objeto no se encuentra centrado en el plato. Existe una distancia l del centro del plato al origen de coordenadas de la muestra.

Resolvimos el problema de centrado de proyecciones dividiéndolo en dos casos: (a) cuando el objeto es pequeño, (b) cuando el objeto es grande. Las técnicas que se utilizaron para solucionar ambos casos son descritas en las siguientes secciones y se basan en que la distancia l (Figura 2.10) que existe del centro de un objeto al centro del plato giratorio se puede calcular por medio de la expresión $l \cos \phi$, donde ϕ es el ángulo asociado la proyección.

2.4.1. Cuerpos pequeños

Para solucionar el problema de centrado de proyecciones donde el objeto es pequeño, definimos que el origen de coordenadas de un objeto es su centro de masa (centroide), el cual se puede estimar como el centro de la proyección debido a que las proyecciones están libres de ruido. Con el objetivo de estimar este centro desarrollamos un programa que ejecuta los siguientes pasos:

- Paso 0: Calcula el centroide del objeto
- Paso 1: Calcula el desplazamiento $k = \textit{centro de la proyección} - \textit{centroide del objeto}$
- Paso 2: Translada la proyección k pixeles. La translación se realizó con un algoritmo incremental que calcula las posiciones y el tono de gris de los pixeles transladados (ver subsection 2.4.1) usando la interpolación bilineal

El programa que realiza el centrado de proyecciones considerando objetos pequeños, recibe como argumento un archivo que contiene el nombre de todas las proyecciones,

desplazamiento corrector	desplazamiento real	error
29.70941	30	0.290588
29.709412	29.885841	0.176429
29.709412	29.544231	-0.165181
28.709412	28.977774	0.268362
27.709412	28.190779	0.481367
26.709412	27.189232	0.47982
25.709412	25.980762	0.27135
24.709412	24.574562	-0.13485
22.709412	22.981333	0.271921
20.709412	21.213203	0.503791
18.709412	19.283628	0.574216
16.709412	17.207294	0.497882
14.709412	14.999999	0.290587
12.709412	12.678547	-0.030865
9.709412	10.260605	0.551193
7.709412	7.764572	0.05516
4.709412	5.209447	0.500035
2.709412	2.614674	-0.094738
-0.290592	-0.000001	0.290591
-3.290592	-2.614673	0.675919
-5.290592	-5.209446	0.081146
-8.290592	-7.764571	0.526021
-10.290592	-10.260604	0.029988
-13.290592	-12.67855	0.612042
-15.290592	-15.000002	0.29059
-17.290592	-17.207294	0.083298
-19.290592	-19.283628	0.006964
-21.290592	-21.213203	0.077389
-23.290592	-22.981333	0.309259
-25.290592	-24.574562	0.71603
-26.290592	-25.980762	0.30983
-27.290592	-27.189232	0.10136
-28.290592	-28.190779	0.099813
-29.290592	-28.977774	0.312818
-30.290592	-29.544231	0.746361
-30.290592	-29.885841	0.404751

Cuadro 2.1: Error del programa de centrado de proyecciones cuando los objetos son pequeños.

produce las mismas proyecciones; pero todas alineadas con respecto a su centro. Para probar que este programa funciona, cada proyección del fantasma de prueba (ver Fig. 2.8), se desplazó sobre el eje de las x, $30\cos\phi$ pixeles, donde ϕ es el ángulo de la proyección. Tal como se muestra en la Tabla 2.1 el programa obtiene un error menor a un píxel al transladar la cantidad *desplazamiento corrector* pixeles para centrar cada proyección.

Translación de una imagen

Para transladar una imagen se programó un algoritmo incremental y se uso la aproximación bilineal con el objetivo de calcular las posiciones y los tonos de gris de los pixeles transladados.

El proceso de translación de una imagen sucede cuando cada punto $P(x, y)$ se mueve dx unidades paralelas al eje x y dy unidades paralelas al eje de las y, al punto nuevo $P'(x', y')$, éste es expresado en la siguiente ecuación

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{2.7}$$

Sin embargo, el algoritmo de desplazamiento incremental lleva a cabo el proceso inverso de translación, esto es

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}, \tag{2.8}$$

debido a que cuando transladamos los pixeles de la imagen original obtenemos pixeles con coordenadas no enteras. Es decir, el algoritmo supone que tenemos los pixeles transladados en coordenadas enteras exactas, para poder calcular de que coordenadas se originaron, de tal manera que la ecuación (2.8) se puede escribir en dos ecuaciones como

$$x = x' - dx, \tag{2.9}$$
$$y = y' - dy. \tag{2.10}$$

Estas ecuaciones para un tiempo i se reescriben como

$$x_i = x'_i - dx, \tag{2.11}$$
$$y_i = y'_i - dy. \tag{2.12}$$

Si consideramos un desplazamiento en el eje x, tenemos que $x'_{i+1} = x'_i + 1$, $y'_{i+1} = y'_i$ por lo que las ecuaciones (2.11) y (2.12) son iguales a

$$x_{i+1} = (x'_i + 1) - dx = x'_i - dx + 1 = x_i + 1, \tag{2.13}$$
$$y_{i+1} = y'_i - dy = y_i. \tag{2.14}$$

Pero, si consideramos un desplazamiento en el eje y, tenemos que $y'_{i+1} = y'_i + 1$, $x'_{i+1} = x'_i$, por lo que las ecuaciones (2.11) y (2.12) son iguales a

$$y_{i+1} = (y'_i + 1) - dy = y'_i - dy + 1 = y_i + 1, \tag{2.15}$$
$$x_{i+1} = x'_i - dx = x_i. \tag{2.16}$$

Una de las ventajas de utilizar el algoritmo incremental es, por ejemplo, que para calcular cada píxel transladado sólo se necesita una suma, en vez de dos como en las ecuaciones (2.9) y (2.10).

Por otro lado, la interpolación bilineal determina el valor de un píxel en forma ponderada usando los cuatro vecinos más cercanos de la siguiente manera

$$\begin{aligned} k =& a(1-x)(1-y) + b(x)(1-y) \\ &+ c(1-x)(y) + d(x)(dy), \end{aligned} \tag{2.17}$$

dado que el área de un píxel es 1, cada uno de ellos se pondera con el valor del área opuesta. Ésta se utiliza en una gran variedad de métodos, por ejemplo en la interpolación de mallas de control, en el cálculo de iluminación, en el mapeo de texturas, etc.; porque es computacionalmente simple y produce un mapeo suave, preservando continuidad y conectividad [15]. Esta interpolación En la Fig. 2.11 el píxel k se interpolará bilinealmente usando los valores de los pixeles a,b,c y d.

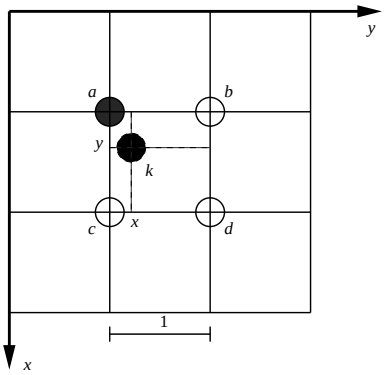


Figura 2.11: Interpolación bilineal. El valor del tono de gris para k es una interpolación de los tonos de gris de los cuatro pixeles a, b, c y d, como muestra la ecuación. (2.17).

El procedimiento 2 muestra el algoritmo de desplazamiento incremental usando la interpolación bilineal. El desplazamiento se lleva a cabo sólo sobre el eje de las x.

2.4.2. Cuerpos grandes

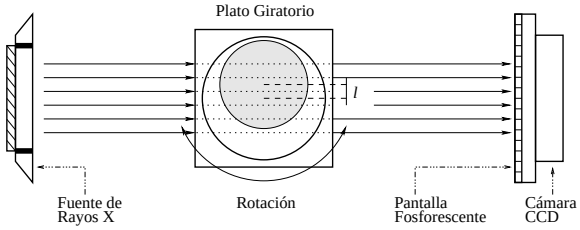


Figura 2.12: La proyección no muestra la información completa cuando el objeto es grande.

El centroide de la proyección es un buen punto de referencia cuando las proyecciones muestran la información completa del objeto escaneado, sin embargo cuando el objeto es grande las proyecciones no muestran la información completa del objeto, por lo que es necesario encontrar otra manera de solucionar el problema de centrado de proyecciones de objetos grandes. En la Fig. 2.12 se puede observar un objeto grande no centrado. El centro del objeto está a una distancia l del centro del plato giratorio. Dado que conocemos el ángulo de la proyección, podemos calcular la distancia desplazada por medio de la expresión $l \cos \phi$, donde ϕ es el ángulo de la proyección.

Para solucionar este problema diseñamos un programa que calcula, por medio de correlación cruzada, el desplazamiento que existe entre el centro de una proyección y otra, también definimos como proyección de referencia la que tiene asociado un ángulo de rotación de 90° (en esta proyección no hay desplazamiento, l es igual a cero), con la finalidad de alinear el resto de las proyecciones con respecto al centro de ésta. Además partimos

Procedimiento 2 Desplazamiento incremental

Entrada: imagen,número entero que indica el desplazamiento deseado (dx)
Salida: imagen desplazada

1: $inputImg$ = Leer imagen de entrada()
2: $cols \leftarrow$ columnas de la imagen de entrada
3: $rows \leftarrow$ filas de la imagen de entrada
4: **for** $0 \leq i < rows$ **do**
5: **for** $0 \leq j < cols$ **do**
6: $outputImg[i][j] \leftarrow 0.0$;
7: **end for**
8: **end for**
9: Calcular $cy = cx = (cols - 1.0)/2.0$ {las coordenadas del centro cx y cy son iguales porque la imagen es cuadrada}
10: Calcular $x_i = -cx - dx$ {desplazamiento en x}
11: $y_i \leftarrow -cy$
12: **for** $0 \leq i < rows$ **do**
13: $x \leftarrow x_i$
14: $y \leftarrow y_i$
15: **for** $0 \leq j < cols$ **do**
16: Calcular $intx = (int)(x + cx + 0.5)$
17: Calcular $inty = (int)(y + cy + 0.5)$
18: **if** $intx \leq 0$ y $intx < rows$ **then**
19: **if** $inty \leq 0$ y $inty < cols$ **then**
20: $outputImg[i][j] \leftarrow (1-x)(1-y)inputImg[inty][intx] +$
 $(y)(1-x)inputImg[inty][intx+1] +$
 $(x)(1-y)inputImg[inty+1][intx] +$
 $(x)(y)inputImg[inty+1][intx+1]$ {interpolación bilineal}
21: **end if**
22: **end if**
23: $x+ = 1.0$
24: **end for**
25: $y_i+ = 1.0$
26: **end for**
27: Regresar $outputImg$

de la suposición de que se han adquirido proyecciones con un incremento pequeño en el ángulo de rotación y que el objeto sólo se encuentra desplazado con respecto al eje de las x.

Los resultados de centrar 50 proyecciones de un objeto grande con la solución explicada en el párrafo anterior se muestran en la Tabla 2.2. El desplazamiento entre una proyección y otra se presenta en la columna *dx relativo*. La columna *dx proy.18* denota la distancia que existe del centro de la $i-$ésima proyección a la proyección que tiene un ángulo de 90°, es decir a la proyección 18 (proyección de referencia). Se encontró que el ángulo de rotación entre cada proyección no debe ser mayor a 5° para tener un error en el desplazamiento corrector (cantidad de pixeles que se debe de mover la proyección para que ésta se encuentre alineada con la proyección de referencia) menor a 1 píxel.

Búsqueda automática de la proyección de referencia

En la captura real de las proyecciones no sabemos de antemano cuál es la proyección de referencia, es decir, qué proyección fue tomada a un ángulo de inclinación de 90° con respecto a la línea que une la pantalla y la fuente de rayos X. Sin embargo, podemos calcular fácilmente cuál es esta proyección a partir de los valores de desplazamiento relativo por proyección mostrados en la Tabla 2.2. Para realizar este cálculo, nombramos a x como el desplazamiento relativo y a j como la proyección de referencia, de tal manera que el desplazamiento absoluto d, para la proyección k se calcula como

$$d_k = \begin{cases} -\sum_{i=k}^{j-1} x_i & \text{si } k < j, \\ 0 & \text{si } k = j, \\ \sum_{i=j+1}^{k} x_i & \text{si } k > j. \end{cases} \tag{2.18}$$

Para aplicar la ecuación (2.18), suponemos que las proyecciones están en el mismo orden en que fueron adquiridas y que ya tenemos el desplazamiento x calculado, el cual es el desplazamiento de la proyección en el eje-x de la proyección actual con respecto a la anterior.

Con el fin de saber cuál es la proyección de referencia calculamos la suma de los desplazamientos absolutos, para todas las proyecciones, de la siguiente manera

$$s_i = \sum_{k=0}^{n-1} d_k, \tag{2.19}$$

donde n denota el número total proyecciones y d_k es calculado como se muestra en la ecuación (2.18). Por lo que la proyección de referencia es la correspondiente al menor valor de la función s.

En la Fig. 2.13 se puede ver un ejemplo de los desplazamientos absolutos calculados según la ecuación (2.18). Para generar esta imagen escogimos las proyecciones 10, 17, 18, 19 y 25 como la proyección de referencia. La gráfica con menor error s es la (línea con asteriscos) generada cuando se utiliza la proyección 18 como referencia, ya que la gráfica es simétrica con respecto al eje de las x.

número proyección	dx relativo	dx proy.18	ángulo proy.	desplazamiento real	error
1	0.023161	8.481113	5	9.961947	1.480834
2	0.023661	8.504274	10	9.848077	1.343803
3	0.024102	8.527935	15	9.659258	1.131323
4	-0.952216	8.552037	20	9.396926	0.844889
5	0.026003	7.599821	25	9.063078	1.463257
6	0.026258	7.625824	30	8.660254	1.03443
7	-0.950354	7.652082	35	8.191521	0.539439
8	0.026844	6.701728	40	7.660444	0.958716
9	-0.950379	6.728572	45	7.071068	0.342496
10	-0.96275	5.778193	50	6.427876	0.649683
11	0.016746	4.815443	55	5.735765	0.920322
12	-0.964051	4.832189	60	5	0.167811
13	-0.965843	3.868138	65	4.226182	0.358044
14	-0.968245	2.902295	70	3.420202	0.517907
15	0.006908	1.93405	75	2.588191	0.654141
16	-0.969963	1.940958	80	1.736482	-0.204476
17	-0.970995	0.970995	85	0.871558	-0.099437
18	-0.972451	0	90	0	0
19	-0.975139	-0.975139	95	-0.871558	0.103581
20	-0.97936	-1.954499	100	-1.736482	0.218017
21	-0.985375	-2.939874	105	-2.58819	0.351684
22	-0.012354	-2.952228	110	-3.420201	-0.467973
23	-0.993002	-3.94523	115	-4.226183	-0.280953
24	-0.998174	-4.943404	120	-5.000001	-0.056597
25	-1.005994	-5.949398	125	-5.735765	0.213633
26	-0.022123	-5.971521	130	-6.427876	-0.456355
27	-1.002874	-6.974395	135	-7.071068	-0.096673
28	-1.009304	-7.983699	140	-7.660444	0.323255
29	-0.023663	-8.007362	145	-8.191521	-0.184159
30	-1.009956	-9.017318	150	-8.660254	0.357064
31	-0.032378	-9.049696	155	-9.063078	-0.013382
32	-0.032334	-9.08203	160	-9.396926	-0.314896
33	-1.009026	-10.091056	165	-9.659258	0.431798
34	-0.030836	-10.121892	170	-9.848077	0.273815
35	-0.030705	-10.152597	175	-9.961947	0.19065
36	-0.030472	-10.183069	180	-10	0.183069
37	-0.030527	-10.213596	185	-9.961947	0.251649
38	-0.030443	-10.244039	190	-9.848077	0.395962
39	-0.030325	-10.274364	195	-9.659258	0.615106
40	0.946506	-9.327858	200	-9.396926	-0.069068
41	-0.031209	-9.359067	205	-9.063078	0.295989
42	-0.030988	-9.390055	210	-8.660254	0.729801
43	0.945944	-8.444111	215	-8.191521	0.25259
44	-0.031064	-8.475175	220	-7.660445	0.81473
45	0.959164	-7.516011	225	-7.071068	0.444943
46	0.959351	-6.55666	230	-6.427875	0.128785
47	-0.020159	-6.576819	235	-5.735763	0.841056
48	0.960708	-5.616111	240	-4.999999	0.616112
49	0.962644	-4.653467	245	-4.226182	0.427285
50	0.965247	-3.68822	250	-3.420201	0.268019

Cuadro 2.2: Se muestran los datos para centrar 50 proyecciones de un objeto grande. Por medio de correlación cruzada se puede saber el desplazamiento entre una proyección y otra *(dx relativo)*. La columna *dx proy.18* denota la distancia que existe del centro de la $i-$ésima proyección a la proyección que tiene un ángulo de 90°. El error es casi siempre menor a un píxel.

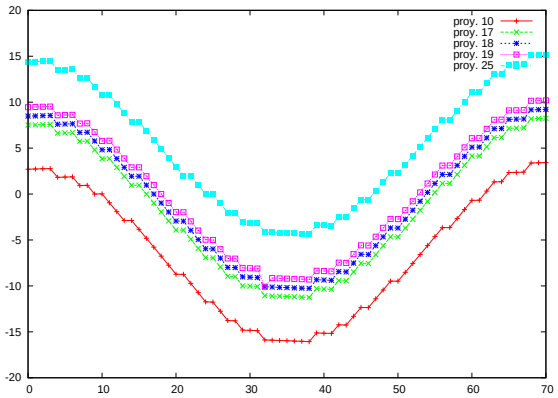


Figura 2.13: Se tomarón a las proyecciones 10, 17, 18, 19 y 25 como las proyecciones de referencia. La proyección 18 (línea con asteriscos) es la que presenta el menor error, por lo tanto ésta es la que se debe definir como proyección de referencia.

2.5. Segmentación volumétrica

Segmentar las estructuras de un volumen es una tarea fundamental en el procesamiento y análisis de imágenes. Especialmente en el caso de imágenes medicas o en la práctica radiológica, donde se desea aislar regiones anatómicas de interés a partir de la Tomografía Computarizada (TC) y la Resonancia Magnética Nuclear (RM).

La segmentación de imágenes puede ser descrita como el particionamiento de una imagen en las regiones que la constituyen. Estas regiones son homogéneas con respecto a una o varias características tales como la densidad o la textura. Idealmente, un método de segmentación separa las distintas estructuras anatómicas o regiones de interés de una imagen sin asignar un significado a cada región (proceso de etiquetación), ya que éste es un proceso aparte del de segmentación [16].

Existe una gran variedad de métodos de segmentación, entre los más importantes podemos mencionar [17]:

- Aproximaciones basadas en umbral
- Crecimiento de regiones
- Aproximaciones basadas en agrupamiento
- Modelos aleatorios de Markov
- Redes neuronales artificiales
- Aproximaciones guiadas por atlas

Las aproximaciones basadas en umbral, segmentan las estructuras de una imagen usando un valor (umbral) que se obtiene al analizar el histograma de ésta. Un histograma proporciona el número de pixeles que existen para cada valor de gris en una imagen 2D en escala de grises. De tal manera que el umbral corresponde a un valor de gris que separa las estructuras deseadas. Este método es fácil de implantar cuando solo se requieren dos estructuras; sin embargo, cuando se desean encontrar más estructuras se vuelve complicado elegir los valores de umbral además, no utiliza información acerca de las coordenadas

espaciales de los datos, por lo que en presencia de ruido no separa de manera adecuada las regiones principales de una imagen.

El método de crecimiento de regiones extrae una estructura o región que se encuentra agrupada de acuerdo a cierto criterio, por ejemplo el color de los pixeles o por los bordes de los objetos en la imagen. Éste, requiere que el usuario proporcione el punto de comienzo (*semilla*), para extraer los pixeles que estén conectados a él y que cumplan con el mismo criterio (previamente establecido). Es decir, *"crece una región"* a partir de una semilla sembrada por el usuario. Las desventajas de este método radican en que por cada estructura que se quiera segmentar se necesita una semilla sembrada por el usuario, además, éste no funciona para segmentar regiones pequeñas (por ejemplo tumores o lesiones) y puede ser sensible al ruido.

Las aproximaciones basadas en agrupamiento utilizan métodos de agrupamiento como *K-means* [18], *fuzzy c-means* [18], *EM* (del término ingles "Expectation-Maximization") [18], etc.; Éstos son conocidos como métodos no supervisados porque no necesitan un entrenamiento previo para agrupar los datos, esto es, utilizan la información disponible para entrenarse ellos mismos de acuerdo a un parámetro de similitud proporcionado. Los métodos requieren que se especifique de antemano cuantas estructuras o grupos se van a encontrar, son sensibles al ruido dado que no contemplan las coordenadas espaciales de los datos; sin embargo los algoritmos que implantan estos métodos son fáciles de programar.

Los métodos aleatorios de Markov no son métodos de segmentación propiamente; pero se consideran como tales porque utilizan un modelo estadístico para segmentar. Éstos, modelan propiedades de las imágenes utilizando correlaciones locales, generadas al suponer que los vecinos de un píxel pertenecen a la misma clase (región a segmentar) de éste. La segmentación se obtiene mediante la adición de modelos bayesianos a métodos iterativos como *K-means*. Aunque los métodos aleatorios de Markov obtienen a menudo mejores resultados que los que se basan en agrupamiento, entregan imágenes suavizadas permitiendo que se pierdan algunos detalles de las regiones de interés.

Las redes neuronales artificiales son nodos de procesamiento que simulan el aprendizaje biológico. Los métodos que las utilizan para segmentar imágenes pueden usarlas en modo supervisado, esto es, las redes necesitan entrenarse primero con datos previamente calculados con el objetivo de preparar a la red para clasificar nuevos datos. En el caso de que no se cuente con datos de entrenamiento disponibles, las redes se pueden utilizar de manera no supervisada agregando información de las coordenadas espaciales al proceso de aprendizaje.

Las aproximaciones guiadas por atlas son una poderosa herramienta cuando existe disponible una plataforma llamada atlas, el cual es generado compilando información de la anatomía de la estructura a segmentar. El atlas funciona como referencia para segmentar nuevas imágenes. Este método necesita que se asigne un significado a las regiones que queremos segmentar con el objetivo de recolectar información acerca de ellas, lo cual puede convertirse en una desventaja si no se requiere el proceso de etiquetación.

Debido a que en esta tesis implantamos un algoritmo de segmentación basado en agrupamiento, en la siguiente sección explicamos las ventajas y desventajas que presentan los más conocidos algoritmos de agrupamiento.

2.5.1. Agrupamiento

Agrupamiento es la herramienta que permite asociar objetos diferentes en grupos (también llamados cúmulos), de tal manera que si dos objetos pertenecen al mismo grupo el grado de asociación que existe entre éstos es máximo; de lo contrario este grado es mínimo. Esta herramienta puede ser usada para descubrir estructuras en datos sin que sea necesario proveer de antemano una interpretación de éstos, el área de aplicación es muy diversa, por ejemplo, es muy útil en el campo de la medicina; agrupando enfermedades, o en el campo de la arqueología; al establecer taxonomías de herramientas de piedra, objetos funerarios, etc.

Existen dos principales métodos para realizar agrupamiento: particionamiento (que en inglés se conoce como *K-clustering*) y agrupamiento jerárquico [19]. Éstos, pertenecen a los métodos no supervisados ya que no usan datos de entrenamiento como referencia para agrupar otros nuevos, es decir, se entrenan así mismos usando los datos disponibles.

Los algoritmos que implantan el agrupamiento jerárquico organizan los datos en una estructura mediante el cálculo de la proximidad que existe entre éstos, la estructura resultante es conocida como dendograma. Este tipo de agrupamiento se utiliza cuando se requiere de una descripción informativa de como se relacionan todos los datos entre sí o cuando se quiere visualizar la estructura del agrupamiento [19].

A diferencia de los algoritmos jerárquicos, los algoritmos de particionamiento asignan un conjunto de objetos en K cúmulos sin ninguna estructura jerárquica. Éstos requieren de algunos parámetros proporcionados por el usuario, como el número de cúmulos que deseamos obtener.

Los algoritmos de particionamiento más conocidos son: *K-means*, *EM* y *fuzzy c-means*.

K-means particiona mediante la minimización de la función de error de la suma de mínimos cuadrados. El algoritmo (descrito en la siguiente sección) es simple y puede ser fácilmente implantado en muchos problemas prácticos, sin embargo presenta algunas desventajas como: (1) no existe un método universal que permita conocer el número de cúmulos, (2) el algoritmo no converge a los mismos centros de los grupos cuando se dan diferentes centros iniciales, (3) el proceso iterativo de *K-means* no garantiza que el algoritmo alcance el óptimo global, (4) es sensible al ruido, (5) la aplicación del término *medias* (del término inglés "means") sólo se aplica cuando los datos son numéricos y (6) debido a que el algoritmo utiliza la distancia euclidiana tiende a generar cúmulos con forma esférica [18].

Maximización de la expectación (EM), es un algoritmo iterativo que hace suposiciones (expectaciones) acerca de la función de densidad de la probabilidad que describe al conjunto de datos, esto es, utiliza alguna función de densidad (normal, Poisson, Gaussiana, etc.) para describir los datos, tratando de maximizar la probabilidad de que los éstos sean descritos por la función de densidad usada. El particionamiento se realiza cuando se encuentra la mezcla de funciones que describe a todo el conjunto de datos, estos es posible debido a que encontrar los cúmulos en un conjunto de datos es equivalente a encontrar sus funciones de distribución. Las desventajas principales del algoritmo son: (1) no existe un método universal que permita conocer el número de cúmulos, (2) es mucho más sensible a los parámetros iniciales (centros de los grupos) que *K-means* y *fuzzy c-means*, (3) el proceso iterativo de *EM* no garantiza que el algoritmo alcance el óptimo global, (4) es sensible al ruido, (5) converge lentamente y (6) cuando utiliza una mezcla de Gaussianas esféricas el algoritmo genera cúmulos con forma esférica [18].

A diferencia de los dos algoritmos anteriores, donde un dato pertenece únicamente a un cúmulo, *fuzzy c-means* supone que un dato pertenece a todos los cúmulos con cierto grado de membresía. Éste, es el algoritmo de agrupamiento difuso más popular, es particularmente útil cuando las fronteras entre los cúmulos no son bien definidas, además permite descubrir relaciones sofisticadas entre los datos; sin embargo también padece de las mismas limitantes para conocer el número cúmulos y es sensible al ruido [18].

Escoger un algoritmo de agrupamiento que resuelva un problema en particular, puede ser una tarea difícil debido a que éstos han surgido en contextos muy específicos. Nosotros elegimos *K-means* porque: (1) pertenece a los algoritmos de particionamiento (no nos interesa conocer la relación que existe entre todos los datos de nuestras reconstrucciones, ni tampoco visualizarlos en un árbol), (2) es simple, (3) nuestros datos son numéricos y no contienen ruido, (4) porque basados en el histograma de las reconstrucción de un fantasma podemos conocer el número de cúmulos.

2.5.2. Algoritmo de segmentación basado en agrupamiento usando *K-means*

Debido a que la fase de reconstrucción arroja una matriz tridimensional, que contiene diferentes valores de grises (cada valor de la matriz representa una densidad), en este trabajo desarrollamos un algoritmo de segmentación que utiliza, como motor de agrupamiento, el método *K-means*. El objetivo de este algoritmo es el de segmentar volúmenes con la misma densidad.

El propósito de *K-means* es el de clasificar n datos en k cúmulos disjuntos, basado en las características de los datos. El agrupamiento es realizado al minimizar la suma de cuadrados de la distancia que existe entre un dato y el centro de un cúmulo. El algoritmo que implante este método producirá exactamente k cúmulos diferentes con la mayor distinción posible. Los siguientes pasos son realizados por *K-means* [20]:

- Paso 0: Se asignan aleatoriamente los elementos a cada cúmulo
- Paso 1: Cada elemento es asignado al cúmulo cuya distancia al centro de éste sea menor
- Paso 2: Para cada cúmulo se recalcula su centro con los nuevos elementos asignados
- Paso 3: Se vuelve al paso 2 mientras haya cambios en los elementos que conforman los cúmulos

El programa de segmentación desarrollado por nosotros implanta *K-means* por medio de la función *Kcluster*, que pertenece a la biblioteca *The C Clustering Library*[21]. La rutina *Kcluster* ejecuta repetidamente el algoritmo *K-means*, guardando la solución óptima y la frecuencia con que la solución es encontrada. El centro de un cúmulo y la distancia de cada elemento a éste, se calculan de acuerdo a los argumentos enviados a *Kcluster*, los cuales son:

- nclusters
  Número de cúmulos k.

- nrows
  Número de filas de la matriz de datos M.

- ncols
  Número de columnas de la matriz de datos N.

- data
  Matriz de datos (MxN).

- mask
  Matriz de enteros (MxN) que muestra cuales datos en *data* están faltando (en caso de que se esten usando matrices donde no todos los datos existan), si $mask[i][j] == 0$, entonces el dato i, j está faltando.

- transpose
  Valor que indica si se deben analizar las columnas o las filas de la matriz de datos, si es igual a cero se agrupan las filas, pero si es igual a uno se agrupan las columnas.

- weight
  Los pesos son usados para calcular las distancias. Si *transpose* es igual a cero, la longitud de este arreglo debe ser igual al número de columnas de la matriz de datos, pero si es igual a uno, la longitud de este arreglo deberá ser igual al número de filas de la matriz de datos. Si el arreglo tiene longitud diferente todo el arreglo será ignorado.

- npass
  Número que indica las veces que el algoritmo *K-means* es ejecutado. Si $npass > 0$ el algoritmo *K-means* se ejecuta *npass* veces, asignando aleatoriamente los elementos iniciales de cada cúmulo en cada corrida. Si $npass == 0$, entonces el algoritmo es ejecutado una sola vez, usando inicialmente los centros de los cúmulos especificados por el usuario.

- method
  Bandera que especifica si la media aritmética (*'a'*) o la mediana (*'m'*), es usada para calcular el centro del cúmulo.

- dist
  Bandera que indica que función de distancia es usada:

  - *'c'*: Correlación.
  - *'a'*: Valor absoluto de la correlación.
  - *'u'*: Correlación no centrada.
  - *'x'*: Valor absoluto de la correlación no centrada.
  - *'s'*: Correlación rango de Spearman [21].
  - *'k'*: Medida no paramétrica de Kendall [21].
  - *'e'*: Distancia Euclidiana.

  La distancia Euclidiana es usada para otros valores de distancia.

- clusterid
  Este arreglo será usado para almacenar el número de cúmulo al que cada elemento fue asignado por el algoritmo de *K-means*. Si $npass == 0$, entonces el contenido de *clusterid*, como parámetro de entrada, es usado como el asignamiento inicial de los elementos pertenecientes a los centros de cada cúmulo; como salida, *clusterid* contiene la solución óptima encontrada por el algoritmo. Si $transpose == 0$ su dimensión es igual a *nrows*, de lo contrario es igual a *ncols*.

- error
  La suma de distancias de los elementos con sus respectivos centros una vez que se ha encontrado la solución óptima. Puede ser usado como un criterio para comparar las soluciones producidas en diferentes ejecuciones de *kcluster*.

- nfound
  Determina la frecuencia con la que la solución óptima fue encontrada.

En este trabajo, alimentamos a la función *Kcluster* de tal manera que la *distancia euclidiana* se usa para calcular la distancia de un elemento al centro de un cúmulo, la *media aritmética* se usa para determinar el centro de un cúmulo, la matriz *data* se llena (un dato por cada fila) con los valores de la matriz que contiene la reconstrucción del objeto que queremos segmentar, el número de filas *nrows* es el número de elementos de esta matriz, *ncols* se asigna a 1. Indicamos, por medio de *transpose=0*, que el agrupamiento se realiza por filas, además consideramos pesos iguales (*weight[i]=1*) y que ningún dato esta faltando. Debido a que la biblioteca *The C Clustering Library* asigna aleatoriamente los centros de los cúmulos, permite comparar los resultados de diferentes ejecuciones del algoritmo *K-means*, con la finalidad de obtener cual ejecución se comporta mejor. Sin embargo nosotros establecemos que el algoritmo se ejecute sólo una vez, es decir *npass=0*, por lo que *clusterid* contiene los valores iniciales de los centros de los cúmulos.

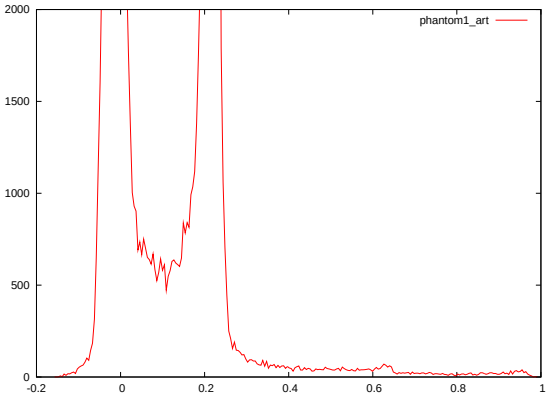


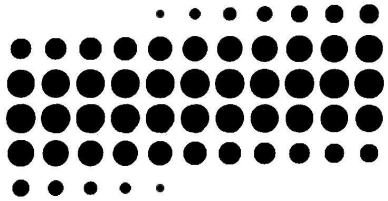
Figura 2.14: Histograma de la reconstrucción de un fantasma hecha con ART.

Para conocer el número de cúmulos con los que cuenta una reconstrucción e identificar que elementos pertenecen a los centros de éstos, analizamos el histograma de las densidades de la reconstrucción. De tal manera que, el número de cúmulos que debemos segmentar corresponde al número de picos en éste, cada centro corresponde a una densidad cuyo valor está alrededor de cierto umbral definido por cada pico. Los valores que limitan este umbral los llamamos límite superior e inferior.

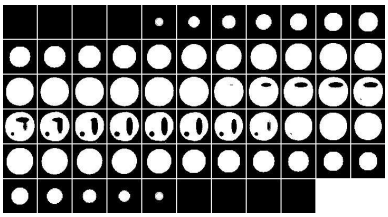
El programa de segmentación recibe como argumento un archivo, en el cual se deben especificar el número de cúmulos y el límite superior e inferior del umbral que define cada densidad, además imprime el número de iteraciones que k-means ejecutó en el proceso de segmentación. Éste, produce tantas matrices tridimensionales binarias como cúmulos se hayan especificado, debido a que cada densidad segmentada se almacena en una matriz. A continuación se muestra el contenido del archivo que este programa necesita para segmentar la reconstrucción hecha con ART del fantasma de prueba que hemos usado a lo largo de éste capítulo (ver sección 3.3.5). Este archivo es creado basado en la información proporcionada por el histograma de la reconstrucción, el cual es ilustrado en la Fig. 2.14.

```
1. $#nClusters$
2.      3
3. $#limite Superior  Limite Inferior$
4.   0.0                    0.1
5.   0.15                   0.25
6.   0.6                    0.7
```

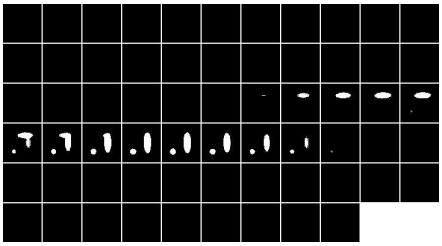
En la línea 2 se especifica el número de cúmulos, el umbral que define al centro del cúmulo uno se muestra en la línea 4, el umbral que define al centro del cúmulo dos se muestra en la línea 5 y así sucesivamente. Las densidades segmentadas de la reconstrucción realizada con ART del fantasma de prueba se muestran en la Fig. 2.5.2. Los pixeles blancos representan la densidad agrupada.



Fondo del fantasma de prueba.



Densidad D_1 del fantasma de prueba.



Densidad D_2, D_3 y D_4 del fantasma de prueba.

Capítulo 3

Visualización con aplanados

En este capítulo describimos un algoritmo que utiliza aplanados para visualizar las superficies de los volúmenes generados en la fase de segmentación (capítulo 2). Primero definimos las características de las superficies punto-muestreadas, después describimos los fundamentos del aplanado, finalmente explicamos la implementación del algoritmo de visualización de superficies (basado en aplanados) que se desarrolló en este trabajo.

3.1. Superficies punto muestreadas

Para visualizar la información obtenida al escanear un objeto por medio de tomógrafos, muchos métodos se han desarrollado. Entre los más novedosos destacan aquellos que utilizan puntos muestra para representar esta información.

El término *punto muestra* es comúnmente usado para referirse a una muestra de una superficie dos-dimensional que se encuentra en un espacio tridimensional (por ejemplo una muestra de superficie de un objeto tridimensional), la cual incluye información (posición, normal y color) acerca de la geometría y apariencia de la superficie de un objeto escaneado. En esta tesis un punto muestra es el punto central de un voxel que incluye la misma información.

Tradicionalmente, los puntos muestra se agrupan como mallas poligonales (mallas conformadas por polígonos) para su manipulación y dibujado [4]; sin embargo conforme la complejidad del objeto escaneado aumenta se vuelve menos eficiente representar los puntos como una malla poligonal u otro nivel de representación más alto (ya que se tiene que estar cambiando continuamente la información de conectividad para cada triángulo o polígono si el observador cambia de posición). Como alternativa muchos métodos para dibujar representaciones de puntos muestra se ha desarrollado.

Se le llama *representación de puntos muestra* a la forma de representar (dibujar y manipular) superficies de objetos punto-muestreadas. Éstas, discretizan al mismo tiempo la geometría y apariencia de la superficie de un objeto escaneado, consisten de un conjunto arbitrario de puntos muestra sin ningún dato adicional[22] y se caracterizan por dos propiedades fundamentales:

- Puntos no conectados:
  No existe información de conectividad entre los puntos muestra. En otras palabras, cada punto es almacenado individualmente sin información explícita de los vecinos

de los puntos en la superficie, a diferencia de las mallas triangulares, donde cada triángulo es definido por tres vértices conectados [22].

- Muestreo no uniforme:
  El conjunto de puntos que define la superficie es distribuido arbitrariamente, o no uniformemente, en el espacio. Es decir, el patrón de muestreo no está restringido a mallas uniformes; al contrario de las superficies basadas en NURBS [**?**] que requieren de la existencia de una malla estructurada de puntos de control. [22].

Los métodos que utilizan representaciones basadas en puntos para visualizar la superficie de un objeto se clasifican de acuerdo a cómo reconstruyen la superficie, por lo menos cuatro métodos se han propuesto[6]:

- Detección de hoyos y llenado del espacio de pantalla. Los puntos muestra individuales son proyectados en la pantalla, de tal manera que los pixeles que no reciben puntos son detectados. La superficie se interpola a partir de los pixeles vecinos.
- Generación de más puntos. Una superficie se interpola en el espacio del objeto para garantizar que cada píxel recibe por lo menos un punto muestra.
- Aplanamiento [1]. Un punto muestra de superficie se proyecta sobre la pantalla, su contribución es extendida sobre los pixeles vecinos para garantizar que se cubre toda la superficie. Métodos de calidad mayor promedian las contribuciones sobre el píxel.
- Enlazado. Una malla de polígonos se usa para interpolar los puntos muestra de la superficie.

En la siguiente sección describimos los fundamentos del aplanamiento, debido a que implantamos este método para visualizar las superficies de un objeto escaneado.

3.2. Aplanamiento

El aplanamiento es una técnica usada para dibujar superficies de objetos. El nombre de ésta se debe a la primitiva de dibujado que utiliza llamada aplanado. Esta técnica linealiza por trozos la superficie del objeto a visualizar por medio del aplanado, esto es, por cada punto muestra el algoritmo asigna un aplanado orientado de acuerdo a la normal que tenga la superficie en las coordenadas del punto.

Los aplanados permiten expandir la contribución de un punto muestra alrededor de él. Fueron propuestos como primitiva de dibujado con el objetivo de cubrir hoyos entre los puntos muestra vecinos. Cada aplanado tiene asociada una posición, una normal, un color y un área. El área de un aplanado se representa por un radio, de tal manera que cada aplanado es un disco circular en el espacio del objeto que se quiere visualizar.

Los aplanados pueden ser opacos o difusos, ambos tienen la misma forma e información asociada, sin embargo en los difusos el color va difuminándose conforme se alejan del centro como se ilustra en la Fig. 3.1, permitiendo que la transición de color en un objeto parezca más suave, ya que dónde los aplanados se traslapen sus colores serán mezclados.

[1] En este trabajo usamos el término *aplanado* para referirnos a la palabra en inglés *splat* [8]. De la misma forma usamos *aplanamiento* para la palabra inglesa *splatting*.

Figura 3.1: Aplanado difuso.

La proyección de un aplanado circular en el espacio del objeto es similar a una elipse en el espacio de pantalla, tal como se muestra en la Fig. 3.2. A pesar de que comúnmente se utilizan elipses, un algoritmo de visualización que implante aplanamiento puede mezclar estos dos tipos de geometrías (elíptica y circular) para visualizar el mismo objeto. La geometría elíptica de los aplanados provee una adaptación localmente óptima a la curvatura principal de la superficie del objeto [23].

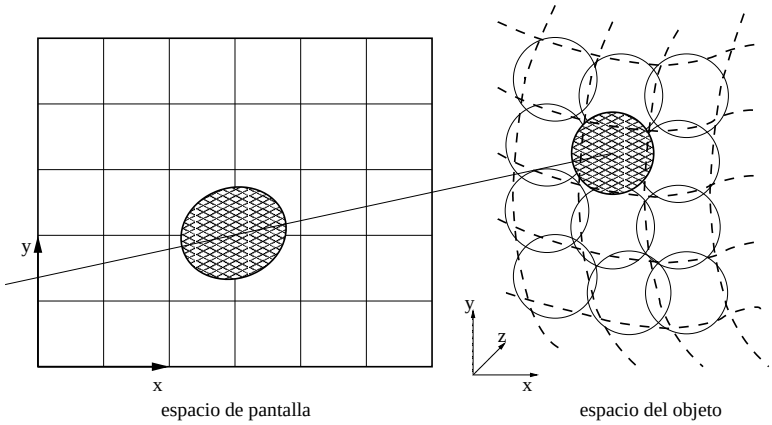


Figura 3.2: Un aplanado que en el espacio del objeto es circular se proyecta como una elipse en el espacio de la pantalla.

A continuación se describe el algoritmo de visualización que implantamos usando aplanados.

3.3. Algoritmo de visualización usando aplanados

La fase de segmentación aisla las densidades que conforman el objeto escaneado, produciendo una serie de volúmenes binarios, cada volumen representa una densidad. Con el objetivo de visualizar la superficie de cada una de las densidades, implantamos un algoritmo que utiliza aplanados para reconstruir la superficie. Tal como lo hace aplanamiento, representamos una superficie por un conjunto de puntos, cada punto se asocia a un aplanado orientado a lo largo de la superficie del objeto. El algoritmo que desarrollamos se conforma principalmente por los módulos mostrados en la Figura 3.3. La extracción de

puntos en la superficie se realiza usando morfología matemática, el cálculo de la normal de cada punto se realiza mediante un ajuste a mínimos cuadrados. La interfaz gráfica asocia un aplanado a cada punto, orientado de acuerdo a su normal.

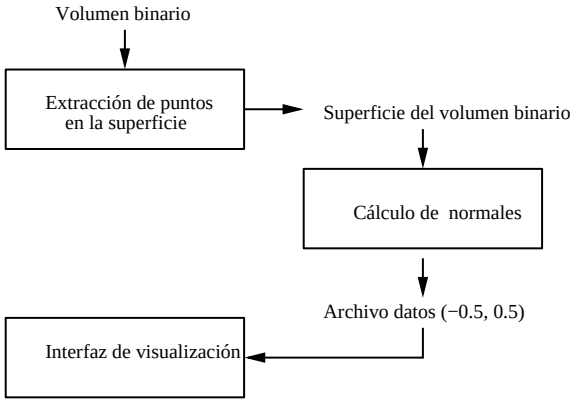


Figura 3.3: Fases del algoritmo de visualización.

3.3.1. Extracción de puntos en la superficie

Desarrollamos un programa para extraer los puntos en la superficie de un volumen binario. El programa obtiene la superficie (perímetro de un volumen) restando la erosión del volumen a éste, tal como se muestra en el procedimiento 3.

La erosión es una de las operaciones que conforman la morfología matemática, ésta se desarrolla a partir de la teoría de conjuntos. Comúnmente se aplica a imágenes dos-dimensionales, sin embargo puede ser aplicada a volúmenes tridimensionales. Ésta, contemplando una imagen en 2D binaria, se define como

$$Erosion = B \ominus S = \{x, y | S_{x,y} \subseteq B\}, \tag{3.1}$$

donde B es la imagen binaria, S es el elemento estructural, $S_{x,y}$ es el elemento estructural con el origen trasladado al punto (x, y). El programa desarrollado utiliza el elemento estructural mostrado en la Fig. 3.4.

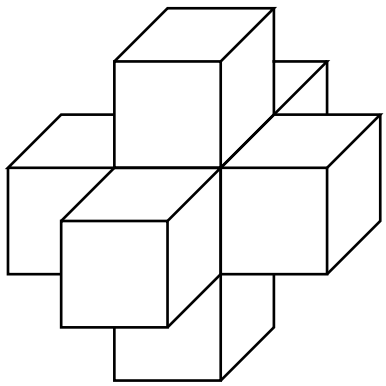


Figura 3.4: Elemento estructural en tres dimensiones. Este elemento es usado en el programa de extracción de puntos en la superficie.

Procedimiento 3 Perímetro de un volumen
Entrada: volumen binario
Salida: perímetro del volumen binario
1: $v_{entrada}$ ←Leer volumen()
2: *erosión*← Aplicar la operación de erosión sobre $v_{entrada}$
3: **for** todas las rebanadas (i) del volumen **do**
4: **for** todas las filas (j) del volumen **do**
5: **for** todas las columnas (k) del volumen **do**
6: Calcular *perímetro[i][j][k]* = $v_{entrada}[i][j][k]$ - *erosión[i][j][k]*
7: **end for**
8: **end for**
9: **end for**
10: Regresar *perímetro*

3.3.2. Cálculo de normales 2D

Debido a que el cálculo de las normales de los puntos en la superficie de un objeto tridimensional es complejo, primero explicaremos un algoritmo que aproxima las normales de los puntos en el perímetro de un objeto bidimensional (contenido en una imagen binaria). Este algoritmo no ajusta los puntos a un plano, como se hace en el cálculo de las normales para un objeto tridimensional; pero muestra algunas partes medulares que se escalaron para el cálculo en tres dimensiones.

El algoritmo que realiza el cálculo de las normales en 2D se implanta por medio de un programa en C, descrito en el procedimiento 5. Éste, recibe como parámetro de entrada una imagen binaria y entrega como salida un archivo. Cada renglón de el archivo contiene 3 números, los primeros dos corresponden a la coordenada de un punto del perímetro, mientras que el tercero corresponde al ángulo de su respectiva normal. La visualización de las normales asociadas a cada punto se ejecuta mediante un programa ($visualNormales$) en *Perl* [24] que procesa la información generada por el cálculo de normales. Éste recibe como argumento el archivo de puntos y ángulos e imprime como salida datos que pueden ser utilizados para graficar las normales en *gnuplot* [25].

El programa que calcula las normales ($normales2D$) obtiene los primeros vecinos para cada punto en el perímetro, en una vecindad de 8 pixeles y asocia un ángulo a cada vecino del punto, tal como se muestra en la Fig. 3.5, con el objetivo de calcular la dirección de la normal al punto k en base al ángulo de sus vecinos.

El ángulo asociado a la normal de un punto se calcula como el promedio de los ángulos de sus vecinos. Debido a que el cálculo se realiza para puntos que pertenecen al perímetro, podemos saber de antemano que un punto tiene dos vecinos. De tal manera que el ángulo de la normal de un punto es descrito por

$$a = \frac{a_{vecino1} + a_{vecino2}}{2}, \tag{3.2}$$

donde a es el ángulo de la normal en el punto k, $a_{vecino1}$ es el ángulo del primer vecino y $a_{vecino2}$ es el ángulo del segundo vecino. Un ejemplo de como se obtiene el ángulo del punto a partir de sus vecinos se muestra en la Fig. 3.6. De acuerdo a la ecuación 3.2 el ángulo del punto k es igual a 90°.

Una vez que conocemos la dirección de la normal (descrita por el ángulo), procedemos a encontrar su sentido. Primero, obtenemos las coordenadas rectangulares de la normal

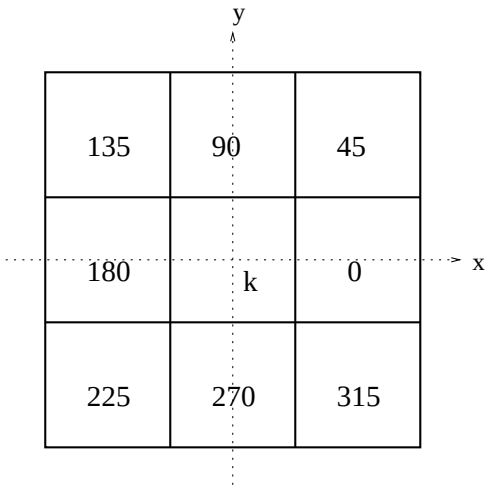


Figura 3.5: ángulos asociados a los 8 vecinos.

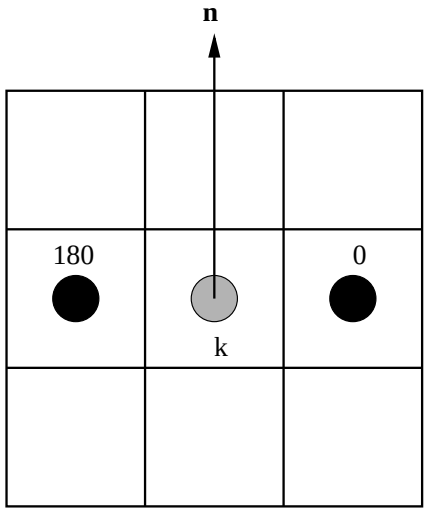


Figura 3.6: Vecindario del punto k. Los puntos negros representan los vecinos del punto k, el valor del ángulo de la normal se calcula con la ecuación 3.2, el cual es igual a 90°.

(presuponiendo que la magnitud de la normal es unitaria), después exploramos la imagen original (la que contiene el objeto) en dirección que éstas indiquen. Si el valor de la imagen en esta dirección es igual a 0 el sentido es correcto, de lo contrario es corregido tal como se muestra en el procedimiento 4.

Procedimiento 4 Verificar el sentido de las normales

Entrada: punto(x, y), ángulo de la normal, imagen original
Salida: ángulo correcto de la normal asociada al punto (x, y)

1: $img \leftarrow$ Leer imagen de entrada()
2: *ángulo* $\leftarrow$ ángulo de la normal
3: Calcular $x_{nvoPixel} = Round(\cos(ángulo * \pi/180.0))$
4: Calcular $y_{nvoPixel} = Round(\text{sen}(ángulo * \pi/180.0))$ {la función *Round* redondea un valor flotante}
5: **if** $img[y + y_{nvoPixel}][x + x_{nvoPixel}] == 0$ **then**
6: $\quad sentido \leftarrow 0$
7: **else**
8: $\quad sentido \leftarrow 180$
9: **end if**
10: Calcular $ángulo += sentido$
11: Regresar *ángulo*

Finalmente, la aproximación del ángulo de la normal para cada punto es correcta, sin embargo los cambios de las normales entre puntos vecinos es muy drástica, al sólo contemplar ángulos cada 45°. Con el fin de mejorar dicha aproximación promediamos el ángulo del punto en estudio con los ángulos de sus vecinos.

3.3.3. Pruebas

Calculamos las normales para los puntos en la superficie de la elipse que se muestra a continuación.

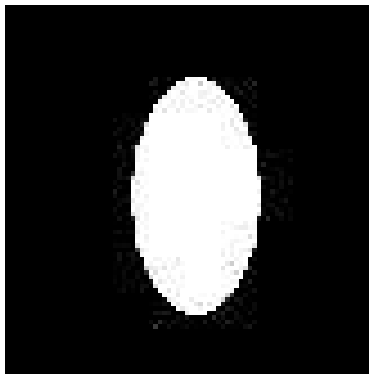


Figura 3.7: Imagen binaria de una elipse.

El comando que indica que se procese el cálculo de las normales es *./normales2D elipse.pgm > archivo.dat*. Tras la ejecución de dicho programa obtenemos un archivo que contiene los puntos que conforman el perímetro de la elipse con los ángulos de sus respectivas normales. El perímetro de la elipse se muestra en la siguiente figura.

Procedimiento 5 Cálculo de normales 2D

Entrada: imagen binaria que contiene el objeto bidimensional (formato *pgm*)

Salida: archivo de puntos y ángulos

1: Inicializar el arreglo $angulos8Vecinos[9] = \{225, 270, 315, 180, 0, 0, 135, 90, 45\}$
2: Inicializar el arreglo $vecinos[8] = 0$
3: $IMG \leftarrow$ Leer la imagen de entrada()
4: Apartar memoria para el arreglo *angulo* de tamaño $filas * columnas$ de la imagen de entrada
5: Calcular el perímetro de la imagen()
6: $j \leftarrow 0$
7: **for** cada punto del perímetro **do**
8: $k \leftarrow 0$
9: **for** $i = 0; i < 8; i++$ **do**
10: **if** el punto i es un vecino **then**
11: $vecinos[k] \leftarrow 1$
12: $k++$
13: **end if**
14: **end for**
15: Contar el número de vecinos
16: **if** el número de vecinos es igual a 2 **then**
17: **for** $i = 0; i < 8; i++$ **do**
18: **if** $vecinos[k] == 1$ **then**
19: Calcular $angulo[j] += angulos8Vecinos[k]$
20: Calcular $angulo[j] /= 2.0$
21: Calcular $angulo[j] += sentido(punto, angulo[j], IMG)$ {ver procedimiento 4}
22: **end if**
23: **end for**
24: **else**
25: Imprimir existen puntos con un número de vecinos diferente a 2
26: exit(1);
27: **end if**
28: $j++$
29: **end for**
30: $j \leftarrow 0$
31: **for** cada punto del perímetro **do**
32: $anguloPromedio \leftarrow angulo[j]$ {$angulo[j]$ es el ángulo de la normal asociada al punto j}
33: **for** $i = 0; i < 8; i++$ **do**
34: **if** el punto i es un vecino al punto j y su ángulo esta en el mismo cuadrante ($\pm 45°$) **then**
35: Calcular $anguloPromedio +=$ *ángulo del vecino[i]*
36: **end if**
37: **end for**
38: Calcular $anguloPromedio /= 3$ {2 vecinos + el punto en estudio}
39: **if** $anguloPromedio \geq 360°$ **then**
40: $anguloPromedio -= 360$
41: **end if**
42: $j++$
43: Desplegar a pantalla las coordenadas del punto en el perímetro y el su ángulo $anguloPromedio$
44: **end for**

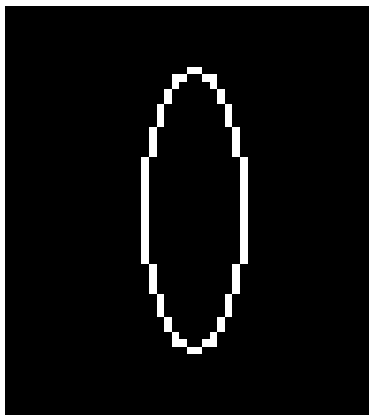


Figura 3.8: Perímetro de la elipse de prueba.

La instrucción *./visualNormales2D.pl archivo.dat > salida.gnu* ejecuta el programa para visualizar las normales de los puntos en el perímetro. Recibe como argumento el archivo de puntos y ángulos (*archivo.dat*) e imprime como salida datos que son utilizados para graficar las normales en *gnuplot*, éstos son almacenados en *salida.gnu*. En la Fig. 3.9 se muestran las normales, representadas por las flechas, de cada punto.

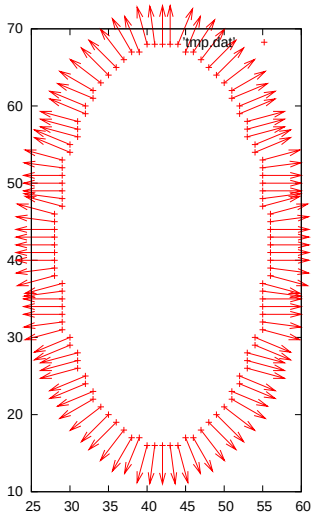


Figura 3.9: Visualización de las normales de la elipse de prueba.

3.3.4. Cálculo de normales 3D

Una vez que se cuenta con los puntos de la superficie de un volumen (densidad contenida en una matriz tridimensional), se procede a calcular las normales de cada uno de ellos. Este cálculo requiere que para cada punto se obtengan los primeros vecinos, en una vecindad de 26 voxels, con el objetivo de ajustarlos (punto y vecinos) a un plano. La dirección de la normal de la superficie en cada punto corresponde a la normal asociada al plano ajustado.

Los algoritmos de visualización basados en puntos obtienen el vecindario de éstos, calculando la *distancia euclidiana* que existe entre todos los puntos, porque suponen que no existe información disponible de conectividad entre ellos [23]. A diferencia de estos algoritmos, en esta tesis obtenemos el vecindario explorando los voxels alrededor

de él, debido a que los puntos con los que contamos están contenidos en una matriz tridimensional.

Para explicar los pasos realizados en el cálculo de las normales 3D abordamos la definición de la ecuación de un plano que tiene un vector normal diferente de cero, $n = (A, B, C)$ que pasa a través del punto $\mathbf{x}_0 = (x_0, y_0, z_0)$, la cual es $n \cdot (\mathbf{x} - x_0) = 0$, donde $\mathbf{x} = (x, y, z)$; de tal manera que, la ecuación general del plano se puede definir como

$$Ax + By + Cz + D = 0, \tag{3.3}$$

donde $D \equiv -ax_0 - by_0 - cz_0$ [26]. Para encontrar el plano que minimice la distancia cuadrática entre cada uno de los puntos de la superficie de un volumen, es necesario resolver el sistema sobre-determinado que se forma con las coordenadas del punto y sus vecinos, el cual es

$$\begin{pmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ . & & & \\ . & & & \\ . & & & \\ x_{26} & y_{26} & z_{26} & 1 \end{pmatrix} \cdot \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix}, \tag{3.4}$$

donde el vector de variables se conforma por los coeficientes de la ecuación del plano A, B, C, D. La matriz de coeficientes se conforma por las coordenadas x_i, y_i, z_i de los vecinos y el punto en estudio. Para resolver este sistema utilizamos SVD (Sigular Value Decomposition). SVD es el nombre con el que se conoce a un conjunto de técnicas que trabajan con ecuaciones o matrices singulares, pueden ser usadas para encontrar la solución de un sistema lineal bajo el criterio de mínimos cuadrados. Estas técnicas se basan en el siguiente teorema: Cualquier matriz $A_{M \times N}$, cuyo número de filas M es mayor o igual al número de columnas N, puede ser escrita como el producto de: una matriz ortogonal $M \times N$ llamada U, una matriz $N \times N$ llamada W (la cual contiene elementos positivos o ceros que corresponden a los valores característicos de A) y la transpuesta de una matriz ortogonal $N \times N$ llamada V. La matriz $U_{M \times N}$ es ortogonal en el sentido de que sus columnas son ortonormales [27].

$$\begin{pmatrix} \\ A \\ \\ \end{pmatrix} = \begin{pmatrix} \\ U \\ \\ \end{pmatrix} \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & ... & \\ & & & w_N \end{pmatrix} \begin{pmatrix} \\ V^T \\ \\ \end{pmatrix}. \tag{3.5}$$

De tal manera que cuando el ajuste al plano se realiza por SVD, podemos encontrar la dirección de la normal del punto de estudio se como una combinación lineal en una de las columnas de $V_{N \times N}$. El número de esta columna corresponde al número de columna de la matriz $W_{N \times N}$ donde se encuentre el valor característico más pequeño. El procedimiento 6 muestra cómo extraer la dirección de la normal en un punto.

Corregimos el sentido de la normal explorando los voxels vecinos del punto en estudio sobre el volumen original. Las coordenadas del voxel que debemos explorar son obtenidas

Procedimiento 6 Extracción de la normal

Entrada: matriz $A_{M\times N}$ que contiene las coordenadas de los vecinos y el punto en estudio
Salida: vector direccional de la normal asociada a la superficie del objeto en el punto en estudio

1: Apartar memoria para las matrices que nos regresa SVD: $V_{N\times N}$, $W_{N\times N}$, $U_{M\times N}$
2: svdcmp(A, U, V, W) {Singular Value Decomposition}
3: **for** todos los valores en la diagonal de la matriz $W_{N\times N}$ **do**
4: **if** i es el menor valor característico **then**
5: Asignar $column = i$ {la columna i es la que tenemos que explorar}
6: **end if**
7: **end for**
8: **for** $0 \leq j <$todos los elementos en la columna *column* de $V_{N\times N}$ **do**
9: $normal[j] \leftarrow V[j][columna]$
10: **end for**
11: Normalizar el vector *normal*
12: Regresar el vector *normal*

al sumar las componentes de la normal con las coordenadas del punto, ésto es posible gracias a que sabemos que sólo se están calculando las normales a los puntos en la superficie y que las coordenadas de la normal están normalizadas. La exploración averigua si el valor del volumen, en estas coordenadas es igual a 0, si es así el sentido de la normal es correcto.

Finalmente, implantamos un programa que calcula las normales (*SVDnormal*) 3D. Éste, entrega como salida un archivo que contiene el número de puntos en la superficie, la distancia mayor entre los voxels (tamaño de la matriz cuadrada) y las coordenadas de cada punto como de sus respectivas normales. A continuación se muestra un ejemplo del archivo de salida. La línea 1 indica que se trabajará con una matriz de $64 \times 64 \times 64$. Como es cuadrada sólo se escribió 64, la línea 2 indica que se cuentan con 2688 puntos en la superficie, de la linea 3 a la 8 se muestran las coordenadas de 6 puntos con las coordenadas de sus normales.

```
1. 64
2. 2688
3. 30 18 27  -0.143580 -0.430739 -0.808373
4. 31 18 27  -0.000000 -0.449099 -0.844030
5. 32 18 27  -0.000000 -0.449099 -0.844030
6. 33 18 27  0.143580 -0.430740 -0.808373
7. 29 19 27  -0.308135 -0.308134 -0.848853
8. 30 19 27  -0.175472 -0.175473 -0.962058
```

El procedimiento 7 muestra el algoritmo que se sigue para calcular las normales a los puntos en la superficie.

3.3.5. Visualizando aplanados

Para esta fase del algoritmo de visualización diseñamos y programamos una interfaz (*VisualTomography*) en *Qt* [28] y *OpenGL* [29] con el fin de visualizar los volúmenes por medio de la primitiva de dibujado aplanado.

Procedimiento 7 Cálculo de normales 3D

Entrada: matriz tridimensional que contiene el volumen original, matriz tridimensional que contiene los puntos en la superficie del volumen (perímetro)

Salida: archivo que contiene los puntos en la superficie del volumen y sus respectivas normales

1: Contar cuantos puntos existen en el perímetro()
2: Escribir a archivo el número de puntos
3: Escribir a archivo el tamaño de la matriz cuadrada
4: **for** cada punto del perímetro (j) **do**
5: **for** $i = 0$; $i < 27$; $i++$ **do**
6: **if** el punto i es un vecino de j **then**
7: Almacenar sus coordenadas x, y, z en la matriz de coeficientes $A_{M\times N}$ {el punto i pertenece a una matriz volumétrica por lo tanto tiene coordenadas x, y, z}
8: **end if**
9: **end for**
10: Calcular $normal = SVD(A_{M\times N})$ {ver procedimiento 6}
11: Verificar el sentido de la normal del punto j con el volumen original y la normal()
12: Regresar la coordenada del punto j y las coordenadas de su normal
13: **end for**

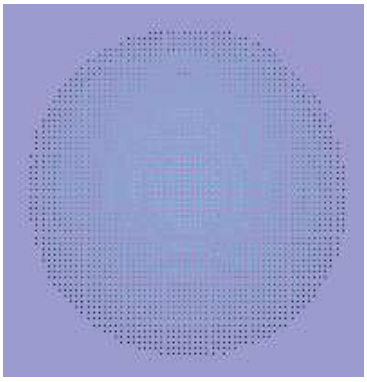
OpenGL (Open Graphics Library) es una interfaz de software (API) para el hardware gráfico, la cual permite al programador producir imágenes de alta calidad de objetos tridimensionales; no solo provee el dibujado para graficar 2D y 3D, también incluye el modelado, trasformaciones, color, iluminación, sombreado, entre otras. *Qt* es una biblioteca de desarrollo de prototipos rápidos desarrollada en $C++$, tiene una excelente documentación y permite asignar un *widget* [30] de *OpenGL* para la visualización.

Para representar un aplanado opaco utilizamos cuatro tipos de primitivas geométricas de *OpenGL*: puntos, triángulos, cuadrados y círculos, también se permite visualizar puntos como una ayuda rápida para el usuario. La interfaz gráfica que desarrollamos permite el intercambio la geometría de los aplanados. Ésta recibe como entrada el archivo arrojado por el programa que calcula las normales (ver subsection 3.3.4). Aunque la normalización de los datos del archivo no es un proceso inherente a la visualización, para nuestra interfaz es necesario normalizar el archivo por medio del programa *normalizeData*, quien lo translada al origen y escala para que quepa en una caja de -0.5 a 0.5 en las tres dimensiones x, y, z. El programa de normalización recibe el archivo como argumento y genera como salida otro archivo; pero con los datos normalizados. De tal manera que, la información (número de puntos en la superficie, el tamaño de la malla y las coordenadas de cada punto con su normal) que proporciona este archivo la usa la interfaz para conocer la posición, la normal y el radio de cada aplanado. El radio se calcula como la distancia máxima entre dos puntos vecinos con la finalidad de evitar huecos en la visualización, esto es

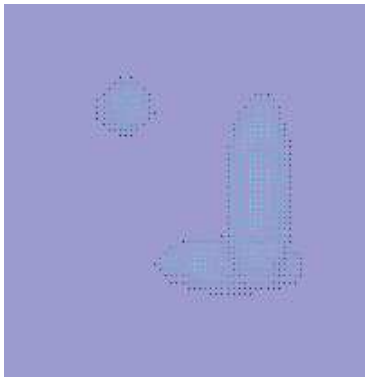
$$r = \frac{\sqrt{3}}{s}, \tag{3.6}$$

donde s es el tamaño de la malla.

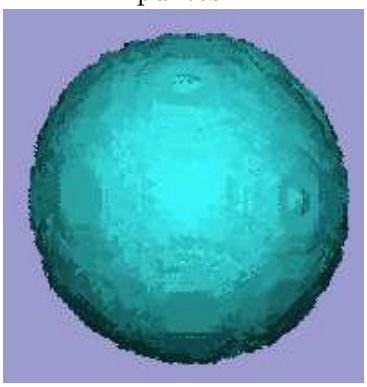
A continuación se muestran, por medio de aplanados, las densidades segmentadas del fantasma diseñado en la sección . La mejor visualización se observa cuando se usan círculos para representar los aplanados.



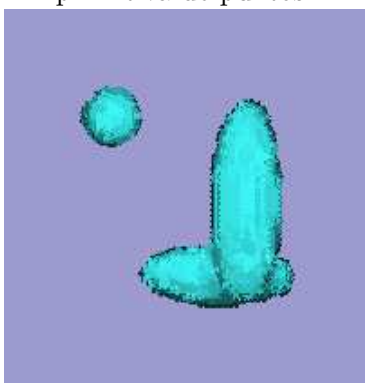
Visualización de la densidad D_1 utilizando la primitiva de puntos.



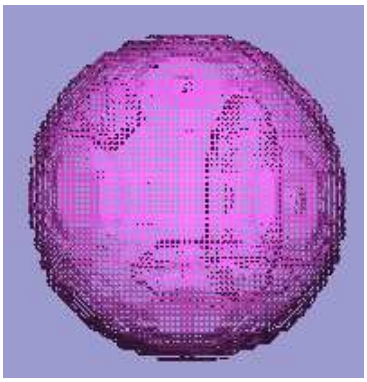
Visualización de la densidad D_2, D_3 y D_4 utilizando la primitiva de puntos.



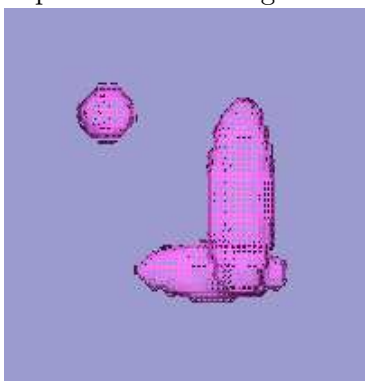
Visualización de la densidad D_1 utilizando la primitiva de triángulos.



Visualización de la densidad D_2, D_3 y D_4 utilizando la primitiva de triángulos.

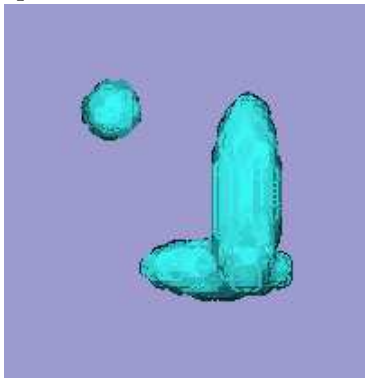


Visualización de la densidad D_1 utilizando la primitiva de cuadriláteros.



Visualización de la densidad D_2, D_3 y D_4 utilizando la primitiva de cuadriláteros.

Visualización de la densidad D_1 utilizando la primitiva de círculos.



Visualización de la densidad D_2, D_3 y D_4 utilizando la primitiva de círculos.

Figura 3.10: Visualización por medio de aplanados del fantasma diseñado en la sección 3.3.5.

Capítulo 4

Pruebas de los programas desarrollados

En este capítulo de la tesis probamos como se comportan los programas realizados cuando los fantasmas a visualizar presentan diferentes simetrías.

Los datos de las pruebas diseñadas se obtienen a partir de dos fantasmas. El primero de éstos ($flor$) se conforma por una sola densidad, ya que se diseñó para observar el resultado que el algoritmo de visualización entrega cuando se utilizan densidades perfectamente segmentadas. Además comparamos este resultado contra la visualización (de la flor) que se obtiene con *voxels simples* [31] y una *malla deformable* [32] ajustada a la superficie.

El segundo fantasma (*helicoide*) se conforma por dos densidades, una densidad simula tener otra en el interior. Con éste se diseñaron dos pruebas: (1) la visualización de sus densidades segmentadas y (2) la visualización de la reconstrucción de sus densidades segmentadas.

4.1. Visualización de una flor

Se diseñó un fantasma semejante a una flor (ver Fig. 4.1), que consiste de cuatro elipses enlazadas, las cuales representan una sóla densidad.

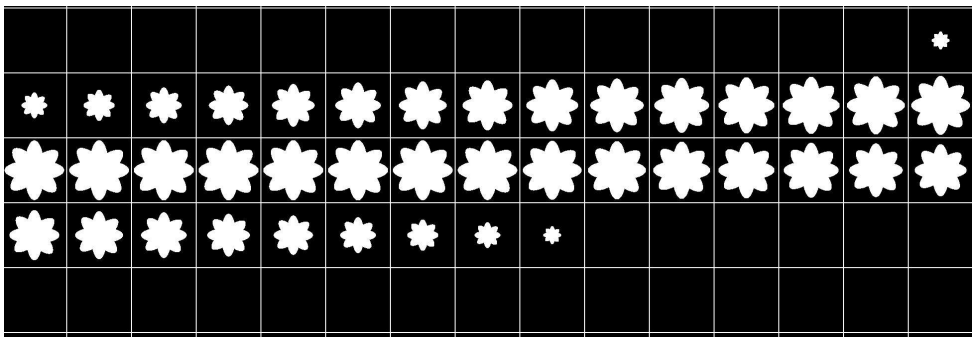


Figura 4.1: Visualización por medio de rebanadas de un fantasma consistente de cuatro elipses enlazadas.

Para visualizar esta flor comenzamos extrayendo los puntos de su superficie utilizando el comando *./perimeter flor128.raw pflor128.raw 128 128 128*. Donde *flor128.raw* es el

volumen binario que contiene la flor (fantasma original), *pflor128.raw* es el nombre del volumen binario de salida que contendrá la superficie, los tres siguientes números especifican la dimensión del volumen ($128 \times 128 \times 128$). La superficie que arroja este programa se muestra por rebanadas en la Figura 4.2.

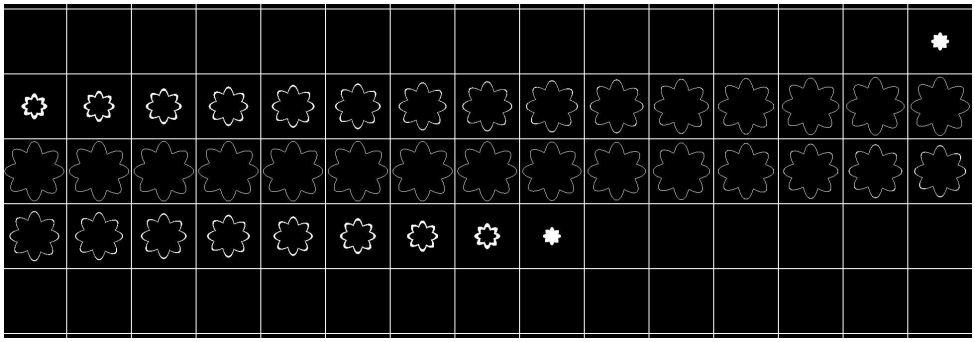
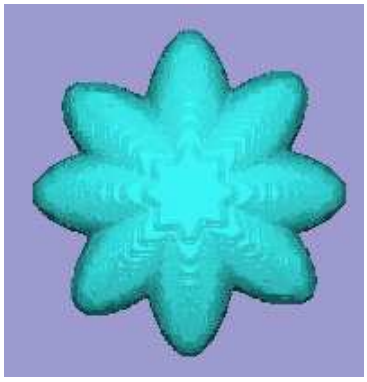


Figura 4.2: Visualización por medio de rebanadas de la superficie del fantasma.

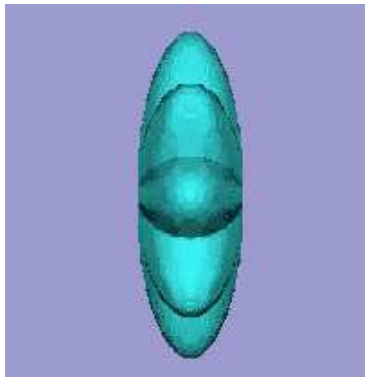
El archivo de puntos (sobre la superficie) y normales se utiliza como argumento por la interfaz de visualización, se genera por medio del comando *./SVDnormal flor128.raw pflor128.raw 128 128 128 > salida.dat*. Donde *flor128.raw* es el volumen binario que contiene al fantasma original, *pflor128.raw* es el nombre del volumen binario que contiene la superficie del fantasma, los tres siguientes números especifican el tamaño del volumen ($128 \times 128 \times 128$), *salida.dat* es el nombre del archivo que contendrá la salida del programa. Los datos de este archivo se transladan al origen y escalan para que quepan en una caja de -0.5 a 0.5 en las tres dimensiones x, y, z, mediante un programa que normaliza los datos. Para ejecutar el programa de normalización (*normalizeData*) es necesario enviar como parámetro: el nombre del archivo de entrada y el nombre del archivo de salida. Aunque los parámetros anteriores son suficientes para ejecutar el programa, es posible normalizar todos las densidades de un mismo volumen con respecto al valor máximo y mínimo de los volúmenes segmentados, por lo que el usuario puede ingresar dos valores correspondientes al valor máximo y mínimo con los que trabajará el proceso de normalización.

Finalmente, en la Figura 4.3 mostramos la visualización de la flor usando aplanados círculares. Éstos son los que proveen la mejor visualización. Las imágenes que ilustran la flor usando: puntos, triángulos y cuadriláteros son encontradas en el apéndice D, Fig. D.1. El dibujado por triángulos debe ser el más barato computacionalmente, aunque por el tipo de fantasma usado, que es pequeño, no se notó una pérdida de rendimiento computacional.

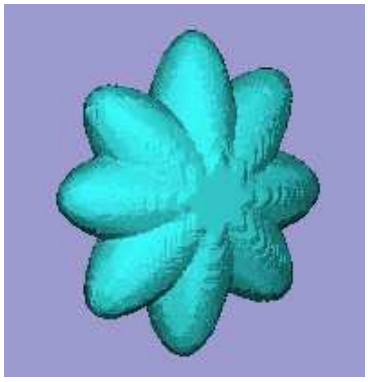
Por otro lado, comparamos la visualización de la flor generada mediante aplanados circulares con: simples voxels (Scubes) y una malla deformable (Proto) ajustada a la superficie. La Figura 4.1 ilustra dicha comparación. Debido a que la realizada con voxels tiene calculada la normal a cada cara del voxel no entrega una buena visualización. La realizada con la malla deformable es buena, pero conocer los parámetros de la malla y ajustarla no es un proceso sencillo. En comparación, la obtención con aplanados es mucho más sencilla y la visualización es de mayor calidad.



Vista frontal de la flor utilizando la primitiva de círculos.

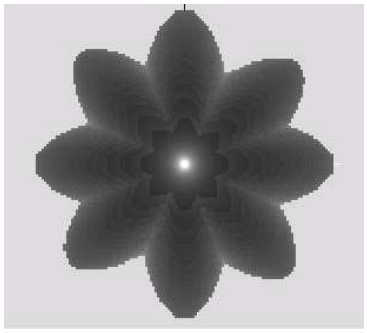


Vista lateral de la flor utilizando la primitiva de círculos.

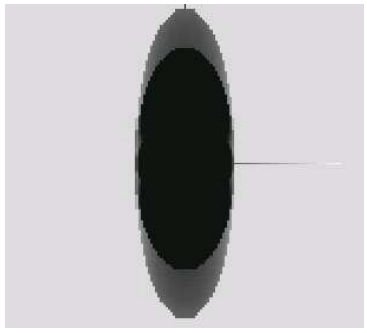


Vista general de la flor utilizando la primitiva de círculos.

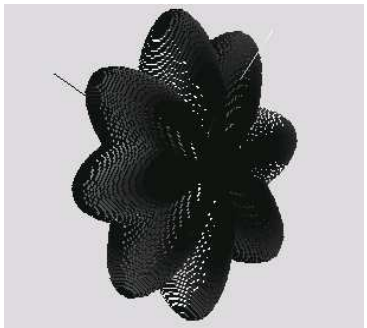
Figura 4.3: Visualización de una flor por medio de aplanados circulares.



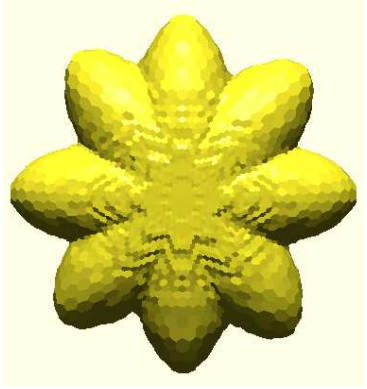
Vista frontal de la flor utilizando la herramienta Scubes.



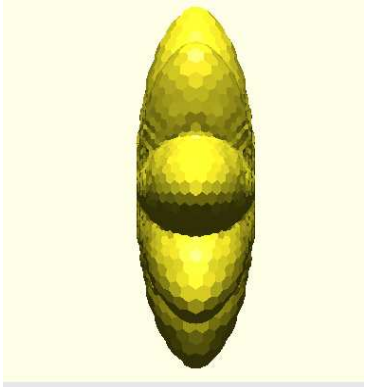
Vista lateral de la flor utilizando la herramienta Scubes.



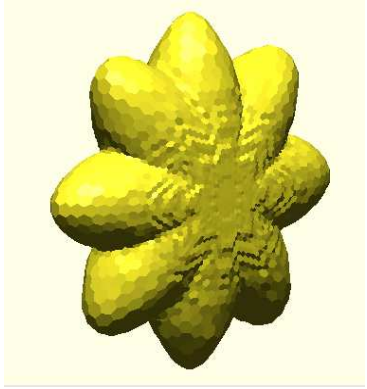
Vista general de la flor utilizando la herramienta Scubes.



Vista frontal de la flor utilizando la herramienta Proto.



Vista lateral de la flor utilizando la herramienta Proto.



Vista general de la flor utilizando la herramienta Proto.

4.2. Visualización de un helicoide

Para esta prueba se diseñó un fantasma compuesto por un cilindro y un helicoide (ver apéndice A), tal como se muestra por rebanadas en la Fig. 4.4, el helicoide envuelve al cilindro simulando una densidad (densidad 1) que contiene otra en el interior (densidad 2). El proceso que se tiene que seguir para visualizar este fantasma es básicamente el mismo que el que se describió en la visualización de la flor (sección 4.1); sin embargo,

debido a que el helicoide conforma una densidad y el cilindro otra, se debe incluir la fase de segmentación antes de extraer los puntos en la superficie.

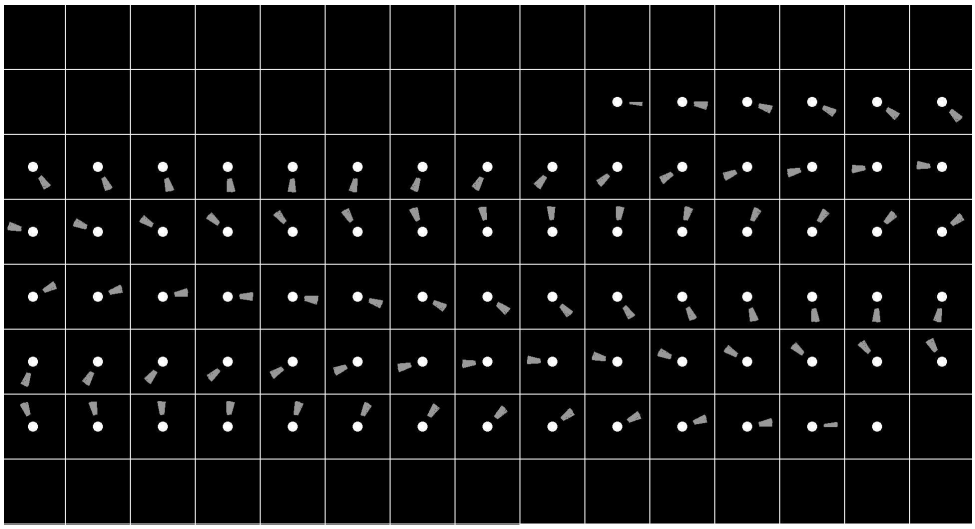


Figura 4.4: Visualización por medio de rebanadas de un cilindro y un helicoide. El helicoide envuelve al cilindro simulando una densidad que contiene otra en el interior.

La instrucción *./kmeansCluster helicoide.raw file.cfg* ordena al programa de segmentación que separe el volumen *helicoide.raw* de acuerdo a lo que indique el archivo de configuración *file.cfg*, el cual se crea a partir de la información que proporciona el histograma de las densidades del volumen (Fig. 4.5). El programa entrega tres volúmenes ya que en el histograma observamos tres cúmulos bien definidos. Los volúmenes que contienen las densidades del fantasma se muestran por medio de rebanadas en el la Fig. D.2 del apéndice D.

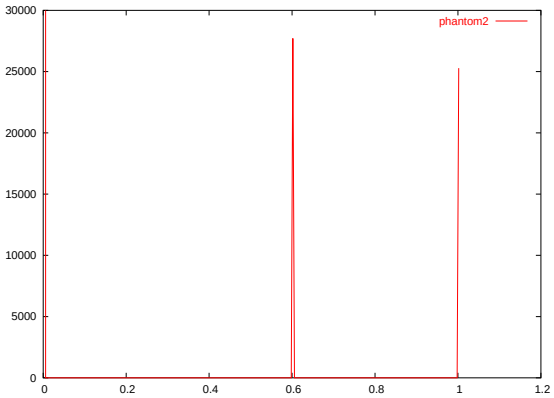
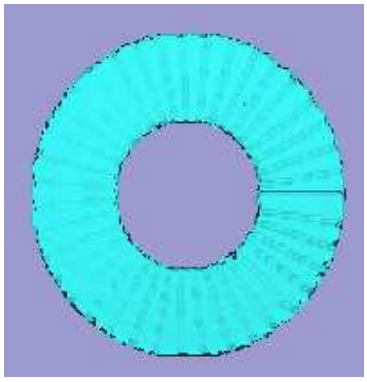
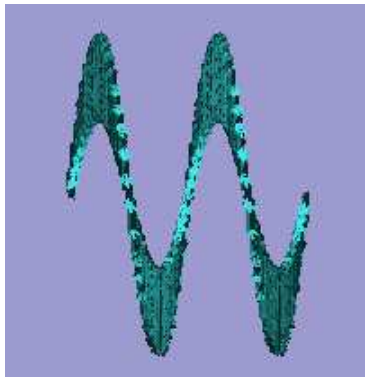


Figura 4.5: Histograma del helicoide.

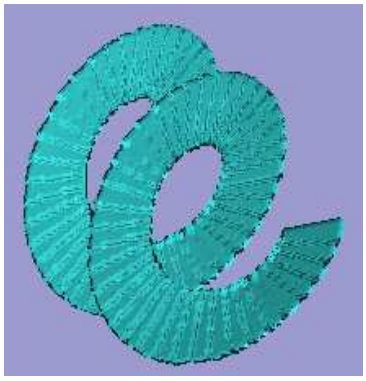
Finalmente, en La Fig. 4.6 y Fig. 4.7 observamos la visualización de las densidades que conforman el helicoide por medio de aplanados círculares. La primer figura corresponde al helicoide mientras que la segunda corresponde al cilindro. Las imágenes que ilustran las densidades usando: puntos, triángulos y cuadriláteros se encuentran en el apéndice D, Fig. D.2 y Fig. D.3.



Vista frontal del helicoide utilizando la primitiva de círculos.

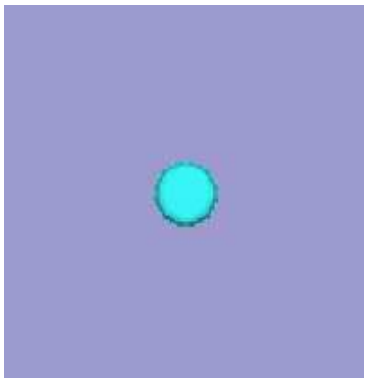


Vista lateral del helicoide utilizando la primitiva de círculos.

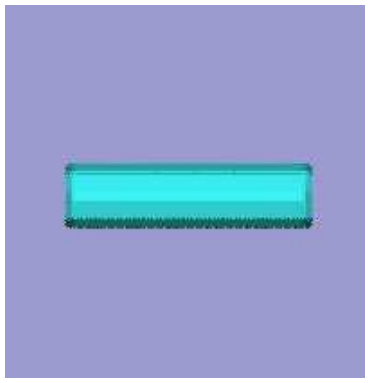


Vista general del helicoide utilizando la primitiva de círculos.

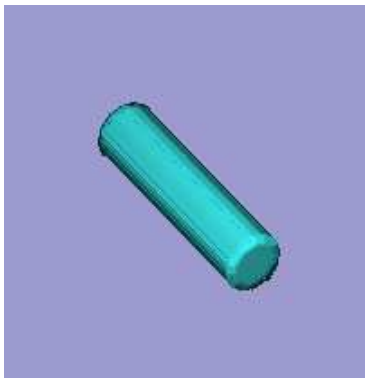
Figura 4.6: Visualización por medio de aplanados de un helicode, el cuál corresponde a una densidad 1 del fantasma diseñado en esta prueba.



Vista frontal del cilindro utilizando la primitiva de círculos.



Vista lateral del cilindro utilizando la primitiva de círculos.



Vista general del cilindro utilizando la primitiva de círculos.

Figura 4.7: Visualización por medio de aplanados de un cilindro, el cuál corresponde a una densidad 2 del fantasma diseñado en esta prueba.

4.3. Visualización de la reconstrucción de un helicoide

El objetivo de esta prueba es utilizar, con un objeto artificial, la mayoría de las partes del software diseñado, por esa razón se reconstruyó el fantasma (*helicoide*) de la Fig. 4.4 por medio del algoritmo ART, a partir de sus proyecciones obtenidas cada 5°. Es conveniente mencionar que los códigos que fueron programados para el centrado de proyecciones no son requeridos en esta prueba ya que se trabaja con datos artificiales.

Al igual que en la prueba anterior (sección 4.2), la fase de segmentación entrega tres volúmenes; sin embargo en ésta, existen huecos en la superficie de los objetos y algunos voxels de la densidad 2 son incluidos en el volumen que contiene la densidad 1, debido a que en el proceso de reconstrucción se pierde definición del umbral que identifica cada una de las densidades. La Fig. 4.8 ilustra el histograma de las densidades, en él se puede observar que el cambio entre densidades no es notorio, al perderse los picos que definen a

éstas.

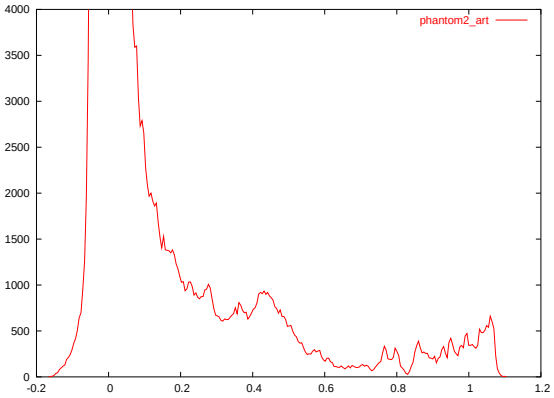
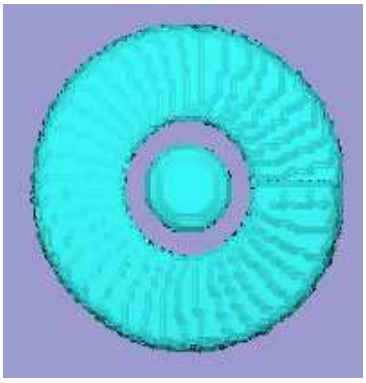


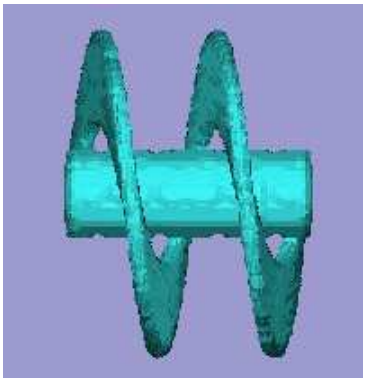
Figura 4.8: Histograma de la reconstrucción hecha con ART del fantasma helicoide.

Para eliminar los huecos en la superficie de los objetos aplicamos (antes de estraer los puntos en la superficie) la operación morfológica *Closing*. Ésta, utiliza un elemento estructural de dimensión $3 \times 3 \times 3$, se implanta por medio del programa *openingClosing*. El resultado de esta operación se muestra en la Fig. D.3 del apéndice D.

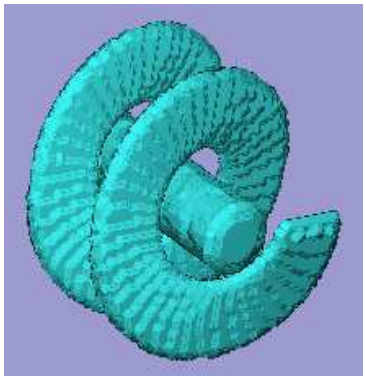
La visualización de las densidades reconstruidas por medio de aplanados círculares se ilustra en la Fig. 4.9 y Fig. 4.10. La primera corresponde al helicoide mientras que la segunda corresponde al cilindro. Ilustramos ambas densidades usando: puntos, triángulos y cuadriláteros en la Fig. D.4 y Fig. D.5 del apéndice D.



Vista frontal utilizando la primitiva de círculos.

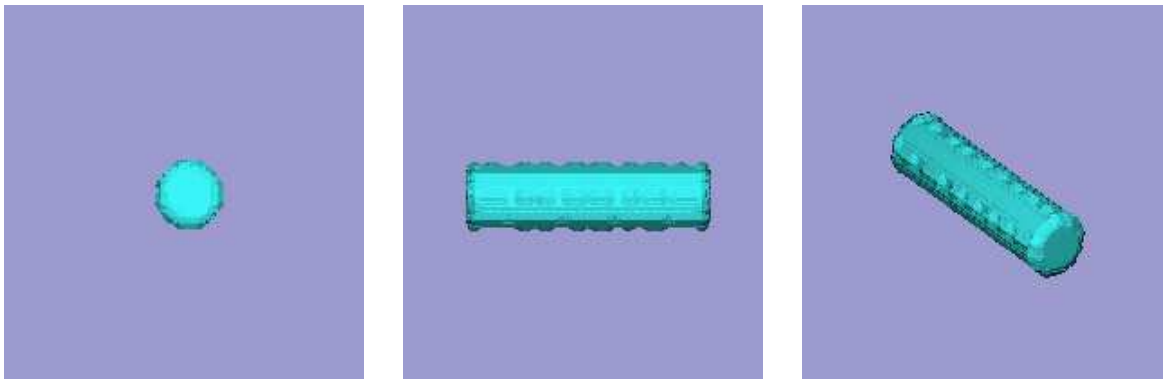


Vista lateral utilizando la primitiva de círculos.



Vista general utilizando la primitiva de círculos.

Figura 4.9: Visualización por medio de aplanados de la reconstrucción del helicoide (densidad 1).



Vista frontal utilizando la primitiva de círculos.

Vista lateral utilizando la primitiva de círculos.

Vista general utilizando la primitiva de círculos.

Figura 4.10: Visualización por medio de aplanados de la reconstrucción de un cilindro (densidad 2).

Capítulo 5

Conclusiones y trabajo a futuro

En este capítulo presentamos un resumen de la tesis, discutimos algunas ventajas y características de las técnicas utilizadas en esta tesis y concluimos listando algunas mejoras que se podrán realizar a futuro para este sistema.

5.1. Conclusiones

Para reconstruir volúmenes a partir de proyecciones obtenidas con un tomógrafo de rayos X y permitir la visualización de las reconstrucciones, dividimos en cinco partes el problema de reconstrucción:

- Adquisición de proyecciones del objeto escaneado
- Centrado de proyecciones del objeto escaneado
- Reconstrucción 3D del objeto escaneado
- Segmentación de las densidades del objeto reconstruido
- Visualización de las superficies de las densidades segmentadas

Sin embargo sólo diseñamos el software para cuatro de estas partes. A continuación damos las conclusiones que se obtuvieron lo largo de este trabajo para cada una de las partes desarrolladas.

5.1.1. Reconstrucción 3D

Para reconstruir un objeto tridimensional a partir de sus proyecciones implantamos el algoritmo de retroproyección y retroproyección filtrada, también utilizamos el algoritmo ART (Técnicas de Reconstrucción Algebráica) que pertenece a la colección XMIPP.

El algoritmo de retroproyección filtrada que implantamos utiliza dos filtros: rampa y Weiner. Éstos mejoran la reconstrucción que produce retroproyección; sin embargo el filtro rampa disminuye la amplitud dinámica del volumen reconstruido y aumenta las altas frecuencias (ruido). Aunque en presencia de ruido, la reconstrucción puede ser mejorada utilizando el filtro Weiner, la mejor reconstrucción la genera ART (como se puede observar en la figura 2.9, capitulo 2).

5.1.2. Centrado de proyecciones

Dividimos el problema de centrado de proyecciones en dos casos: (a) cuando el objeto escaneado es pequeño, (b) cuando el objeto escaneado es grande (ver 2.4, capítulo 2).

Para solucionar el problema de centrado de proyecciones donde el objeto es pequeño, definimos que el origen de coordenadas de un objeto es su centro de masa (centroide), el cual se puede estimar como el centro de la proyección debido a que las proyecciones están libres de ruido. El programa que realiza el centrado de proyecciones considerando objetos pequeños obtiene un error menor a un píxel para cada proyección (Tabla 2.1).

El centroide de la proyección es un buen punto de referencia cuando las proyecciones muestran la información completa del objeto escaneado, sin embargo cuando el objeto es grande las proyecciones no muestran la información completa del objeto, por lo que es necesario encontrar otra manera de solucionar el problema de centrado de proyecciones de objetos grandes.

Para el centrado de objetos grandes se mostró que cuando el incremento en el ángulo de rotación es pequeño, la proyección que tiene un desplazamiento igual a cero ($\phi = 90°$) puede ser tomada como referencia para centrar el resto de las proyecciones con respecto a su centro. Además se probó que es posible detectar el desplazamiento entre proyecciones usando correlación cruzada (capitulo 2, sección 2.4). Se encontró que el ángulo de rotación entre cada proyección no debe ser mayor a 5° para tener un error en el desplazamiento corrector (cantidad de pixeles que se debe de mover la proyección para que ésta se encuentre alineada con la proyección de referencia) menor a 1 píxel.

5.1.3. Segmentación

Debido a que la fase de reconstrucción arroja una matriz tridimensional, que contiene diferentes valores de grises (cada valor de la matriz representa una densidad), en este trabajo desarrollamos un algoritmo de segmentación que utiliza, como motor de agrupamiento, el método *K-means*. El objetivo de este algoritmo es el de segmentar volúmenes con la misma densidad. Los centros iniciales y el número de cúmulos (densidades) que necesita *K-means* son deducidos del histograma de las densidades de la reconstrucción 3D. El número de cúmulos es el número de picos en éste, cada centroide corresponde a una densidad cuyo valor se encuentra alrededor de cierto umbral definido por cada pico.

5.1.4. Visualización con aplanados

La fase de segmentación aisla las densidades que conforman el objeto escaneado, produciendo una serie de volúmenes binarios, cada volumen representa una densidad. Con el objetivo de visualizar la superficie de cada una de las densidades, implantamos un algoritmo que utiliza aplanados opacos para reconstruir la superficie. El algoritmo dibuja la superficie a partir de un conjunto de puntos muestra. El algoritmo hereda las ventajas generales de los métodos de visualización basados en puntos; sin embargo, existen algunas otras características particulares inherentes a la implantación de nuestro algoritmo. La estriba en que un punto muestra es tomado como el centro de cada voxel en la superficie.

El algoritmo que desarrollamos se conforma principalmente por los módulos:

- Paso 0: Extracción por medio de morfología matemática de los puntos de la superficie de cada volumen.

- Paso 1: Cálculo de la normal de cada punto de la superficie mediante el ajuste a un plano por mínimos cuadrados.
- Paso 2: Asociación de un aplanado a cada punto de la superficie orientado de acuerdo a su normal.

El módulo de extracción de superficies obtiene resultados acordes a las superficies del volumen original.

Los puntos son ajustados a un área de superficie del objeto a visualizar mediante el ajuste a un plano, requiriendo que para cada punto se obtengan los primeros vecinos, en una vecindad de 26 voxels. La dirección de la normal de la superficie en cada punto corresponde a la normal asociada al plano ajustado.

En esta fase del algoritmo de visualización otra característica particular de la implantación se hace presente, ya que los algoritmos de visualización basados en puntos obtienen el vecindario de éstos, calculando la *distancia euclidiana* que existe entre todos los puntos, porque suponen que no existe información disponible de conectividad entre ellos [23]. A diferencia de estos algoritmos, en esta tesis obtenemos el vecindario explorando los voxels alrededor de él, debido a que los puntos con los que contamos están contenidos en una matriz tridimensional.

Una vez que se ha resuelto la fase del cálculo de las normales, sección 3.3.4 capitulo 3,se tiene disponible la posición, la normal y el radio de cada aplanado (el radio se calculó como la distancia máxima entre dos puntos vecinos con el fin de evitar huecos en la visualización). Para representar un aplanado opaco utilizamos cuatro tipos de primitivas geométricas de OpenGL: puntos, triángulos, cuadrados y círculos (dodecágonos). La interfaz ofrece la visualización por medio de puntos como ayuda rápida para el usuario. La mejor geometría para ver un aplanado opaco corresponde al aplanado circular mostrado en la Figura 4.3.

Además de la comparación realizada entre las primitivas de OpenGL, comparamos la visualización de un fantasma generada mediante aplanados con: voxels simples y una malla deformable ajustada a la superficie. La visualización con voxels no da una buena visualización, ya que calcula la normal a cada cara del voxel. La visualización con la malla deformable es buena, pero conocer los parámetros de la malla y ajustarla no es un proceso sencillo. En comparación, la obtención con aplanados es mucho más sencilla y de mayor calidad. Los resultados de dicha comparación se pueden observar en la Figura 4.1.

5.2. Trabajo a futuro

En la búsqueda de encontrar una geometría adecuada para representar los aplanados se diseño y programó una interfaz en Qt [28] y OpenGL [29]. Aunque OpenGL nos evita desarrollar el algoritmo de dibujado de volumen desde abajo, las geometrías que ofrece no se adaptan a la curvatura principal de la superficie, es decir, el algoritmo de visualización puede ser mejorado si se implanta un algoritmo que dibuje directamente aplanados tanto elípticos como circulares en el espacio de pantalla. Esta implantación permitirá que los aplanados se adapten a la curvatura de la superficie, ya que los aplanados elípticos permiten dibujar directamente la proyección del espacio tridimensional al espacio de pantalla. Además se pueden combinar las técnicas de dibujado basadas en puntos y de filtrado de texturas, con el objetivo de contemplar las superficies de los objetos como una función de

pesos promediados, es decir el valor de un píxel sea la contribución de todos los aplanados difusos [8], tal como se muestra en la Figura 5.1.

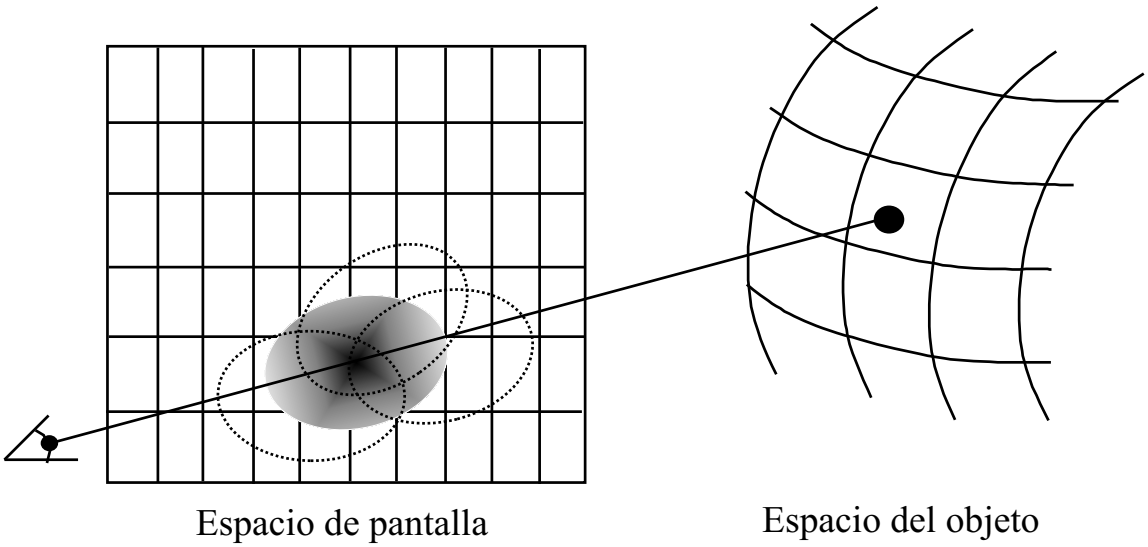


Figura 5.1: El valor de un píxel será la suma de los valores las contribuciones de todos los aplanados difusos.

También se puede desarrollar un algoritmo de segmentación que trabaje de modo no supervisado como lo puede ser un algoritmo genético [33], para evitar la necesidad de conocer previamente el número de densidades que existen en las reconstrucciones de los objetos, o en su defecto se puede combinar dos estrategías de segmentación supervisada que utilizen coordenadas espaciales y detección de bordes entre un objeto y otro.

Hemos probado que los algoritmos de esta tesis funcionan con modelos teóricos; sin embargo es conveniente probarlos con datos reales, es decir con proyecciones obtenidas directamente del tomógrafo, para identificar errores o posibles cambios en los programas y en el sistema de adquisición.

Apéndice A

Solución de un sistema de ecuaciones homogéneo sobredeterminado usando DVS

Lo que se demostrará en este apéndice viene dado en la fórmula de la sección 2.2.4, pág. 23, de [34], que dice: *La descomposición en valores singulares DVS* (SVD en sus siglas en inglés) de cualquier matriz A de tamaño $m \times n$ puede ser escrita como el producto de tres matrices

$$A = UDV^{\mathrm{T}}, \tag{A.1}$$

donde D es la matriz diagonal de tamaño $n \times n$ que contiene la raíz cuadradada de los eigenvalores de AA^{T}, U es la matriz de tamaño $m \times n$ cuyas columnas corresponden a los eigenvectores de AA^{T} y V^{T} es la matriz transpuesta de tamaño $n \times n$ cuyas columnas corresponden a los eigenvectores de $A^{\mathrm{T}}A$.

Debido a que en este trabajo utilizamos sistemas de ecuaciones lineales dónde tenemos más ecuaciones que incógnitas de la forma

$$A\mathbf{x} = 0, \tag{A.2}$$

donde A tiene un tamaño de $m \times n$, m es el número de ecuaciones, n es el número de incognitas y $m > n$. En este apéndice mostraremos como puede encontrarse la solución única usando DVS.

Si no consideramos la solución trivial $\mathbf{x} = 0$ para la Ec. A.2, la solución única, dado un factor de escala, puede encontrarse usando la descomposición a valores singulares. Esta solución es proporcional al eigenvector correspondiente al único eigenvalor igual a cero de $A^{\mathrm{T}}A$. A continuación se mostrará como sucede ésto.

Ya que la norma de la solución de un sistema de ecuaciones homogeneas es arbitraria, se encontrará la solución de norma unitaria en el sentido de mínimos cuadrados. Por lo tanto, se quiere minimizar

$$||A\mathbf{x}||^2 = (A\mathbf{x})^{\mathrm{T}}A\mathbf{x} = \mathbf{x}^{\mathrm{T}}A^{\mathrm{T}}A\mathbf{x}, \tag{A.3}$$

sujeto a la restricción

$$\mathbf{x}^{\mathrm{T}}\mathbf{x} = 1. \tag{A.4}$$

Pero si se introduce el multiplicador de Lagrange λ, resulta que esto es equivalente a minimizar el Langrangiano, es decir

$$\mathcal{L}(\mathbf{x}) = \mathbf{x}^{\mathrm{T}} A^{\mathrm{T}} A\mathbf{x} - \lambda(\mathbf{x}^{\mathrm{T}}\mathbf{x} - 1). \tag{A.5}$$

Si calculamos la derivada del lagrangiano con respecto a $\mathbf{x}$ [35], esto resulta ser [1]

$$\begin{aligned}
\frac{\mathcal{L}(\mathbf{x})}{d\mathbf{x}} &= \frac{d}{d\mathbf{x}}(\mathbf{x}^{\mathrm{T}} A^{\mathrm{T}} A\mathbf{x}) - \lambda \frac{d}{d\mathbf{x}}(\mathbf{x}^{\mathrm{T}}\mathbf{x}) \\
&= \mathbf{x}^{\mathrm{T}} A^{\mathrm{T}} A \frac{d\mathbf{x}}{d\mathbf{x}} + \left[\frac{d\mathbf{x}^{\mathrm{T}}}{d\mathbf{x}}\right] A^{\mathrm{T}} A\mathbf{x} - \lambda \mathbf{x}^{\mathrm{T}} \frac{d\mathbf{x}}{d\mathbf{x}} - \lambda \left[\frac{d\mathbf{x}^{\mathrm{T}}}{d\mathbf{x}}\right] \mathbf{x} \\
&= \mathbf{x}^{\mathrm{T}} A^{\mathrm{T}} A + A^{\mathrm{T}} A\mathbf{x} - \lambda \mathbf{x}^{\mathrm{T}} - \lambda \mathbf{x} \\
&= [\mathbf{x}^{\mathrm{T}} A^{\mathrm{T}} A]^{\mathrm{T}} + A^{\mathrm{T}} A\mathbf{x} - \lambda [\mathbf{x}^{\mathrm{T}}]^{\mathrm{T}} - \lambda \mathbf{x} \\
&= A^{\mathrm{T}} A\mathbf{x} + A^{\mathrm{T}} A\mathbf{x} - \lambda \mathbf{x} - \lambda \mathbf{x} \\
&= 2A^{\mathrm{T}} A\mathbf{x} - 2\lambda \mathbf{x}
\end{aligned}$$

Si igualamos a cero la derivada del lagrangiano con respecto a $\mathbf{x}$ la ecuación (A.5) obtenemos

$$A^{\mathrm{T}} A\mathbf{x} - \lambda \mathbf{x} = 0, \tag{A.7}$$

esta ecuación denota que λ es un eigenvalor de $A^{\mathrm{T}} A$ y que la solución $\mathbf{x} = \mathbf{e}_\lambda$ es el eigenvector correspondiente de $A^{\mathrm{T}} A$. Si se reemplaza $\mathbf{x}$ con $\mathbf{e}_\lambda$ y $A^{\mathrm{T}} A\mathbf{e}_\lambda$ con $\lambda \mathbf{e}_\lambda$ el lagrangiano es

$$\mathcal{L}(\mathbf{e}_\lambda) = \lambda, \tag{A.8}$$

por lo que el mínimo alcanzado es $\lambda = 0$, es decir el último eigenvalor de $A^{\mathrm{T}} A$ [36].

Ahora se mostrará que sólo hay que calcular la DVS de A y no de $A^{\mathrm{T}} A$, la cual es

$$\begin{aligned}
A^{\mathrm{T}} A &= (UDV^{\mathrm{T}})^{\mathrm{T}} \cdot UDV^{\mathrm{T}}, \\
&= VD^{\mathrm{T}} U^{\mathrm{T}} \cdot UDV^{\mathrm{T}}
\end{aligned}$$

pero tenemos que $U^{\mathrm{T}} \cdot U = I$, por lo tanto

$$A^{\mathrm{T}} A = VD^{\mathrm{T}} \cdot DV^{\mathrm{T}} \tag{A.9}$$

dado que V es una matriz ortogonal se tiene que $V^{\mathrm{T}} = V^{-1}$, y como D es una matriz diagonal $D^{\mathrm{T}} = D$, obtenemos

$$A^{\mathrm{T}} A = VD^2 V^{-1}, \tag{A.10}$$

que es la ecuación que define a los eigenvalores, también indica que las entradas de D^2 son los eigenvalores de $A^{\mathrm{T}} A$ y las *columnas* de V son los eigenvectores de $A^{\mathrm{T}} A$. Resumiendo,

[1] Este desarrollo hace uso de la siguiente fórmula que puede verse en [35], [34]

$$\frac{\partial \mathbf{x}\mathrm{T}\mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^{\mathrm{T}} \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a} \tag{A.6}$$

donde $\mathbf{a}$ es un vector.

los valores singulares de A son las raíces cuadradas de los eigenvalores de $A^{\mathrm{T}}A$ [37]. Si reescribimos la ecuación que representa los eigenvalores (ecuación (A.10)) como

$$V^{-1}A^{\mathrm{T}}A = V^{-1}VD^2V^{-1},$$
$$V^{-1}A^{\mathrm{T}}A = D^2V^{-1},$$

podemos ver que se cumple para un eigevector izquierdo (V^{-1}), pero dado que la matriz $A^{\mathrm{T}}A$ es una matriz simétrica, entonces los eigenvalores izquierdos y derechos son simplemente uno la traspuesta del otro [38].

Apéndice B

Algoritmo para la creación de un helicoide y un cilindro

Para crear el fantasma de la prueba sección 4.2, compuesto por un cilindro y un helicoide, se desarrolló un programa en C que recibe como argumento el nombre del volumen de salida y su dimensión. El programa está restringido para volúmenes con dimensión mayor o igual a 128.

Como se muestra en el procedimiento 8 el cilindro es generado a partir de la ecuación del círculo, evaluando en cada rebanada la función

$$f(x, y) = r^2 - (x^2 + y^2), \tag{B.1}$$

donde (x, y) son las coordenadas de cualquier pixel dentro de una rebanada y r representa la constante que determina el radio del círculo. Si $f(x, y) > 0$ el punto (x, y) está dentro del círculo, si $f(x, y) < 0$ entonces el punto está fuera, pero si $f(x, y) = 0$ entonces el punto está sobre el perímetro del círculo.

Por otro lado, para crear el helicoide partimos de las ecuaciones parámetricas [26]

$$\begin{aligned} x &= \cos v, \\ y &= \operatorname{sen} v, \\ z &= v, \end{aligned} \tag{B.2}$$

para $v \in [0, 2\pi)$.

El número de vueltas verticales está determinado por el intervalo $0 \leq v \leq 2n\pi$, donde n es el número de vueltas. En nuestra implantación (procedimiento 9) definimos un radio exterior y uno interior, tal como se muestra en la Fig. B.1 con el objetivo de determinar el ancho del helicoide a dibujar.

Procedimiento 8 Creación de un cilindro

Entrada: dimensión, altura, radio, densidad

Salida: matriz tridimensional que contiene el cilindro creado de acuerdo a los parámetros de entrada

1: vol = crear matriz tridimensional
2: $dim \leftarrow$ dimensión
3: $h \leftarrow$ altura
4: $c \leftarrow (2 * radio - 1.0)/2.0$ {calculamos el centro del cilindro}
5: $ct \leftarrow (dim - 2 * radio)/2$
6: **for** $(dim - h)/2 \geq i(dim + h)/2$ **do**
7: **for** $0 \geq j2 * radio$ **do**
8: Calcular $x = c - j$
9: **for** $0 \geq k2 * radio$ **do**
10: $x \leftarrow c - j$
11: $y \leftarrow c - k$
12: $f \leftarrow radio^2 - x^2 - y^2$ {Calculamos si el punto (x, y) está dentro, fuera o sobre el círculo}
13: **if** $f \geq 0.0$ **then**
14: $vol[i][(int)(j + ct + 0.5)][(int)(k + ct + 0.5)] =$ densidad
15: **end if**
16: **end for**
17: **end for**
18: **end for**
19: Regresar vol

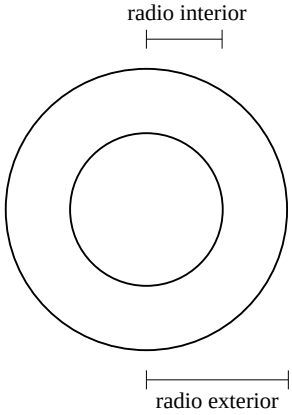


Figura B.1: Radio exterior e interior del helicoide diseñado.

Procedimiento 9 Creación de un helicoide

Entrada: dimensión, radio interior, radio exterior, altura, densidad

Salida: matriz tridimensional que contiene el helicoide creado de acuerdo a los parámetros de entrada

1: vol = crear matriz tridimensional
2: $dim \leftarrow$dimensión
3: $ri \leftarrow$radio interior
4: $re \leftarrow$radio exterior
5: $n \leftarrow 2$ {definimos el número de vueltas}
6: Inicializar $c \leftarrow (dim - 1)/2.0$
7: Inicializar $hi \leftarrow (dim - h)/2$
8: **for** $ri \geq r \leq re$ **do**
9: **for** $0 \geq i360 * n$ **do**
10: Calcular $x = (int)(r * \cos(i * \pi/180) + c + 0.5)$
11: Calcular $x = (int)(r * \text{sen}(i * \pi/180) + c + 0.5)$
12: Calcular $z = h * i/(360 * n) + hi$
13: $vol[z][y][x] \leftarrow$ densidad
14: $vol[z][y][x + 1] \leftarrow$ densidad
15: $vol[z][y][x - 1] \leftarrow$ densidad
16: $vol[z][y + 1][x] \leftarrow$ densidad
17: $vol[z][y - 1][x] \leftarrow$ densidad
18: $vol[z][y][x] \leftarrow$ densidad
19: $vol[z][y][x + 1] \leftarrow$ densidad
20: $vol[z - 1][y][x] \leftarrow$ densidad
21: **end for**
22: **end for**

Apéndice C

Lista de programas realizados

C.1. Reconstrucción

1. Retroproyección Filtrada. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   bpFiltered -i file.sel -o outputFile.vol -f filter

   - *file.sel*
     Archivo que contiene el nombre de las proyecciones del objeto a reconstruir.
   - *outputFile.vol*
     Nombre del volumen de salida. El formato del volumen es SPIDER, entre otras cosas cada voxel tiene un valor en punto flotante en precisión simple.
   - *filter*
     Número que indica el tipo de filtro a utilizar:
     - 0: Sin filtro
     - 1: Rampa
     - 2: Weiner

C.2. Centrado

C.2.1. Objetos pequeños

1. Desplazar una proyección. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   shft1Projection inputFile outputFile shift

   - *inputFile*
     Nombre de la proyección de entrada.
   - *outputFile*
     Nombre de la proyección de salida.
   - *shift*
     Número que indica el desplazamiento deseado.

2. Desplazar n proyecciones. La instrucción que ejecuta dicho programa, al igual que sus parámetros son descritos a continuación:

   shiftProjections file.sel shift

   - *file.sel*
     Archivo que contiene el nombre de las proyecciones que queremos desplazar.
   - *shift*
     Número que indica el desplazamiento deseado.

3. Centrar n proyecciones. La instrucción que ejecuta dicho programa, al igual que sus parámetros son descritos a continuación:

   centerProjections file.sel

   - *file.sel*
     Archivo que contiene el nombre de las proyecciones que queremos centrar.

   Nota: Los programas de desplazamiento duplican el tamaño de la proyección de entrada y la desplazan (sobre el eje de las x) $s * \cos\phi$, donde ϕ es el ángulo de la proyección y s es la cantidad de desplazamiento ingresada por el usuario.

C.2.2. Objetos grandes

1. Desplazar n proyecciones. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   shiftProjections2 file.sel shift

   - *file.sel*
     Archivo que contiene el nombre de las proyecciones que queremos desplazar.
   - *shift*
     Número que indica el desplazamiento deseado.

2. Correlación cruzada n proyecciones. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   crossCorrelationNImg file.sel

   - *file.sel*
     Archivo que contiene el nombre de las proyecciones que no estan centradas.

   Nota: Con el objetivo de simular proyecciones que no cuentan con toda la información, el programa de desplazamiento respeta el tamaño de la proyección de entrada y la desplaza (sobre el eje de las x) $s * \cos\phi$, donde ϕ es el ángulo de la proyección y s es la cantidad de desplazamiento ingresada por el usuario.

C.3. Segmentación

1. Programa que segmenta un volumen. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   kmeansCluster vol.vol file.cfg

   - *vol.vol*
     Nombre del volumen de entrada que deseamos segmentar. El volumen debe estar en formato SPIDER.
   - *file.cfg*
     Nombre del archivo donde se indica el número de cúmulos que deseamos obtener con sus respectivos centros.

2. Programa que calcula el histograma de un volumen. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   histogram vol.vol > salida.dat

   - *vol.vol*
     Nombre del volumen de entrada al que deseamos calcular el histograma. El volumen debe estar en formato SPIDER.
   - *salida.dat*
     Nombre del archivo al que redireccionamos la salida del programa. En este archivo se tendrá información que puede ser graficada con gnuplot, por medio de la script $hisVol.gnu$.

C.4. Visualización

1. Programa que calcula los puntos en la superficie de un volumen. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   perimeter volInput.raw volOutput.raw ncols nrows nslices

   - *volInput.raw*
     Nombre del volumen de entrada. El volumen debe ser binario.
   - *volOutput.raw*
     Nombre del volumen que contendrá los puntos en la superficie.
   - *ncols*
     Número de columnas del volumen.
   - *nrows*
     Número de filas del volumen.
   - *nslices*
     Número de rebanadas del volumen.

2. Programa que calcula las normales de los puntos en la superficie de un volumen. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   SVDnormal volInput.raw volOutput.raw ncols nrows nslices > salida.dat

   - *volInput.raw*
     Nombre del volumen de entrada. El volumen debe ser binario.
   - *volOutput.raw*
     Nombre del volumen que contendrá los puntos en la superficie.
   - *ncols*
     Número de columnas del volumen.
   - *nrows*
     Número de filas del volumen.
   - *nslices*
     Número de rebanadas del volumen.
   - *salida.dat*
     Nombre del archivo al que redireccionamos la salida del programa. En este archivo se tendrá la información de cada punto, sobre la superficie, con su respectiva normal.

3. Programa que normaliza el archivo que contiene puntos sobre la superficie y normales. Este programa puede ser ejecutado con valores adicionales que permiten hacer una normalización tomando encuenta los valores máximos y mínimos del volumen más externo. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   normalizeData fileInput.dat fileOutput.dat

   - *fileInput.dat*
     Nombre del archivo que contiene los puntos en la superficie con sus respectivas normales.
   - *fileOutput.dat*
     Nombre del archivo que se obtendrá de salida con los puntos en la superficie y sus normales. Sus valores serán entre -0.5 y 0.5.
   - *MAX*
     Valor máximo del volumen más externo. Este valor puede ser opcional.
   - *MIN*
     Valor mínimo del volumen más interno. Este valor puede ser opcional.

4. Interfaz de visualización que implanta aplanados. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   VisualTomography file.dat

- *file.dat*
  Nombre del archivo, normalizado, que contiene los puntos en la superficie con sus respectivas normales.

C.5. Normales 2D

1. Programa que calcula las normales de los puntos en el perímetro de una imagen 2D binaria. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   normales2D imagen.pgm > tmp.dat

   - *imagen.pgm*
     Nombre de la imagen de entrada. La imagen debe ser binaria.
   - *tmp.dat*
     Nombre del archivo al que redireccionamos la salida del programa.

2. Programa de visualización de normales. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   visualNormales2D tmp.dat > salida.dat

   - *tmp.dat*
     Nombre del archivo que fue generado por el programa $normales2D$.
   - *salida.dat*
     Nombre del archivo al que redireccionamos la salida del programa. En este archivo se tendrá información que puede ser graficada con gnuplot.

C.6. Otros programas

3. Programa que crea un helioide y un cilindro. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

   Helicoid nameVolOut.vol dim

   - *nameVolOut.vol*
     Nombre del volumen que contendrá el helidoide y el cilindro. El formato del volumen es SPIDER, entre otras cosas cada voxel tiene un valor en punto flotante en precisión simple.
   - *dim*
     Tamaño del volumen.

2. Programa que ejecuta dos operaciones morfológicas. Se puede ejecutar las operación de *Opening* o *Closing* según se requiera, indicando, opcionalmente, el tipo de elemento estructural que se quiere aplicar, este puede ser una cruz o una máscara

$3 \times 3 \times 3$. A continuación se describe la instrucción que ejecuta dicho programa, al igual que sus parámetros:

openingClosing volInput.raw volOutput.raw ncols nrows nslices flag cross

- *volInput.raw*
  Nombre del volumen de entrada. El volumen debe ser binario.
- *volOutput.raw*
  Nombre del volumen que contendrá el volumen después de la operación.
- *ncols*
  Número de columnas del volumen.
- *nrows*
  Número de filas del volumen.
- *nslices*
  Número de rebanadas del volumen.
- *flag*
  Tipo de operación:
  - 0: Opening
  - 1: Closing
- *cross*
  Valor opcional que indica el tipo de elemento estructural que se va aplicar:
  - 0: Cruz en tres dimensiones.
  - 1: Máscara $3 \times 3 \times 3$.

  Nota: si no se indica ningun valor, el programa se ejecutará por omisión usando la cruz como elemento estructural.

Apéndice D

Resultados de las pruebas realizadas en el capítulo

D.1. Visualización de una flor

En la Figura D.1 se visualiza el fantasma $flor128$ por medio de aplanados. El primer renglón corresponde a la visualización hecha por puntos. La visualización por triángulos, cuadrados y círculos corresponde al segundo, tercer y cuarto renglón.

D.2. Visualización de un helicoide

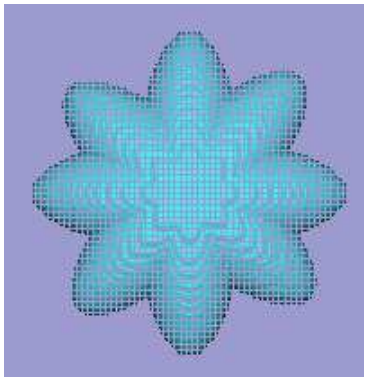
Las densidades segmentadas del helicoide se muestran por medio de rebanadas en la Fig. D.2 (decidimos no presentar el volumen que contiene los pixels del fondo ya que éste no es importante en el proceso de visualización).

En La Fig. D.2 y Fig. D.3 se puede observar por medio de aplanados la visualización de las densidades que conforman al helicoide. La primer figura corresponde al helicoide mientras que la segunda corresponde al cilindro. En ambos casos, el primer renglón muestra la visualización hecha por puntos. La visualización por triángulos, cuadrados y círculos se muestra en el segundo, tercer y cuarto renglón.

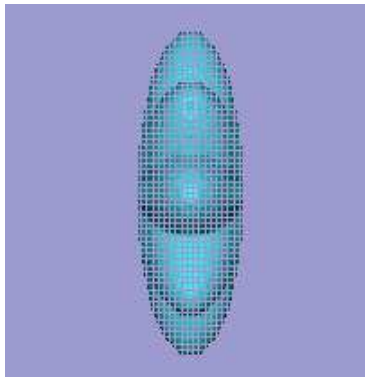
D.3. Visualización de la reconstrucción de un helicoide

La visualización de la reconstrucción de las densidades que conforman el helicoide se muestran por medio de aplanados en la Fig. D.4 (helicoide) y Fig. D.5 (cilindro). En ambos casos, el primer renglón muestra la visualización hecha por puntos. La visualización por triángulos, cuadrados y círculos se muestra en el segundo, tercer y cuarto renglón.

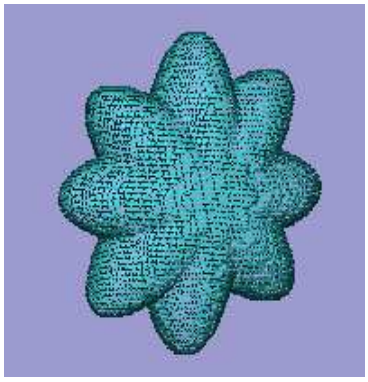
El resultado de las operaciones morfólogicas que aplica el programa *openingClosing* a la reconstrucción del helidoide se muestra en la Fig.D.3.



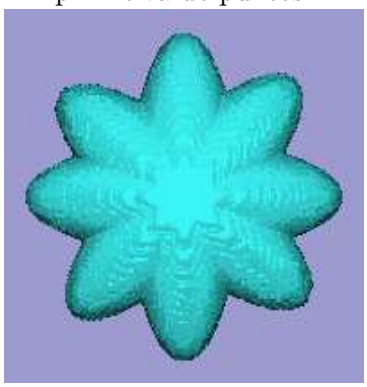
Vista frontal utilizando la primitiva de puntos.



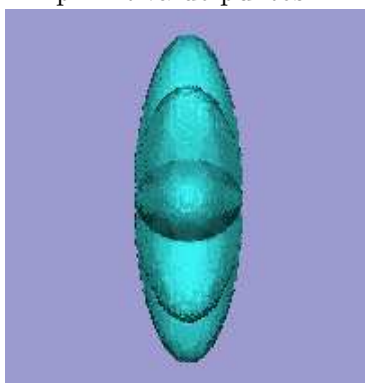
Vista lateral utilizando la primitiva de puntos.



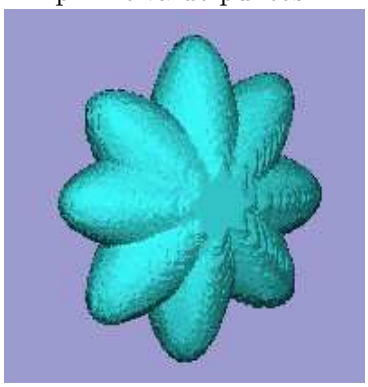
Vista general utilizando la primitiva de puntos.



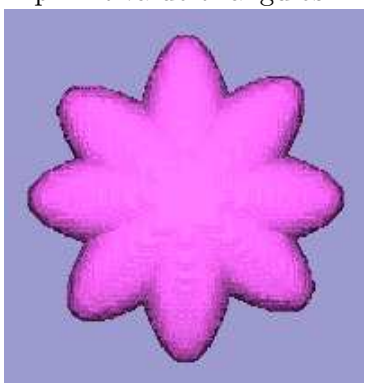
Vista frontal utilizando la primitiva de triángulos.



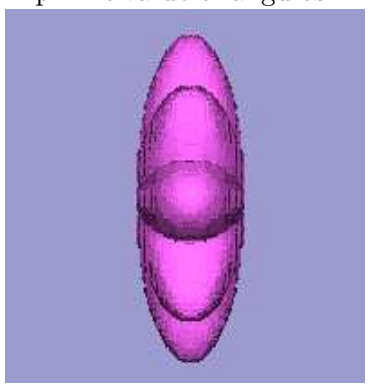
Vista lateral utilizando la primitiva de triángulos.



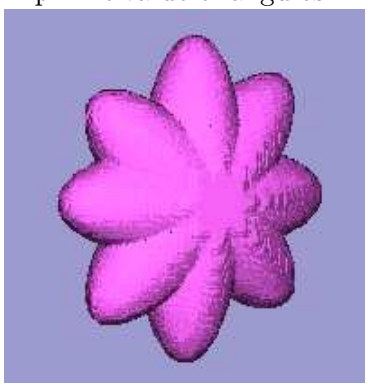
Vista general utilizando la primitiva de triángulos.



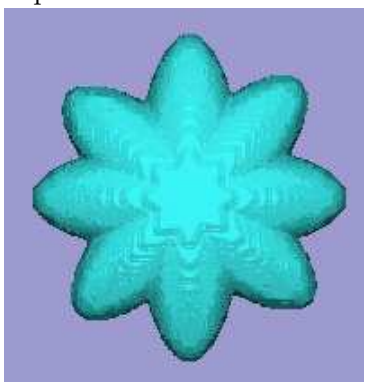
Vista frontal utilizando la primitiva de cuadrados.



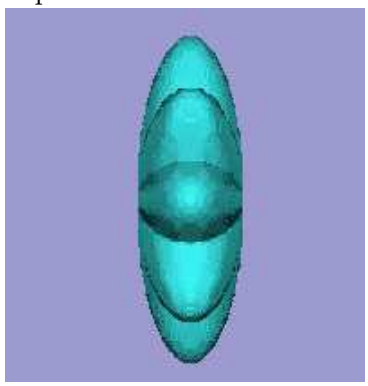
Vista lateral utilizando la primitiva de cuadrados.



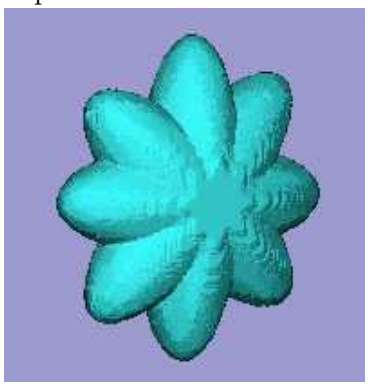
Vista general utilizando la primitiva de cuadrados.



Vista frontal utilizando la primitiva de círculos.

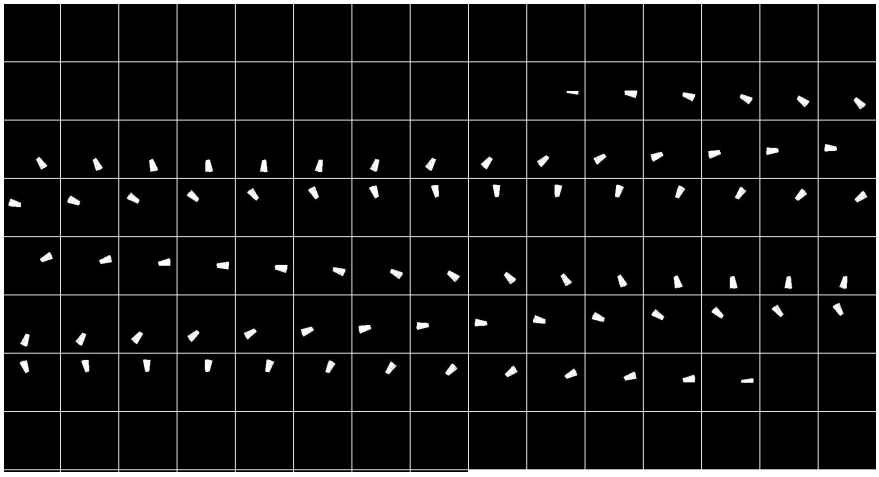


Vista lateral utilizando la primitiva de círculos.

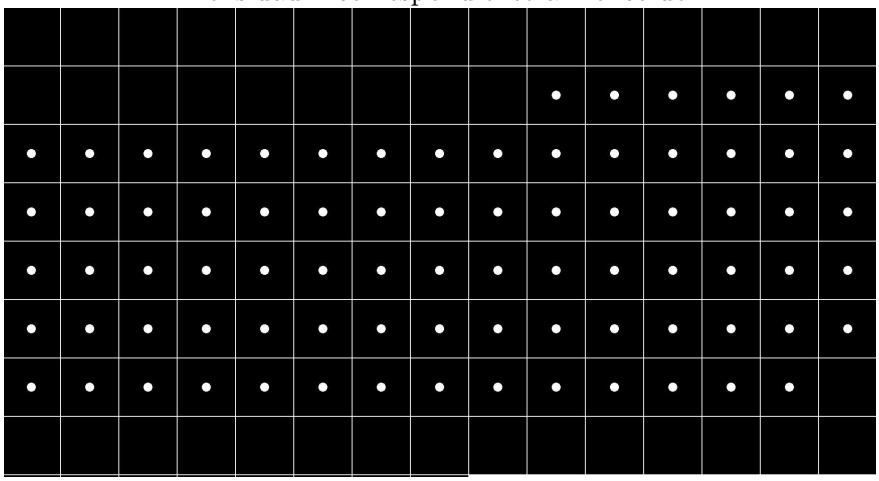


Vista general utilizando la primitiva de círculos.

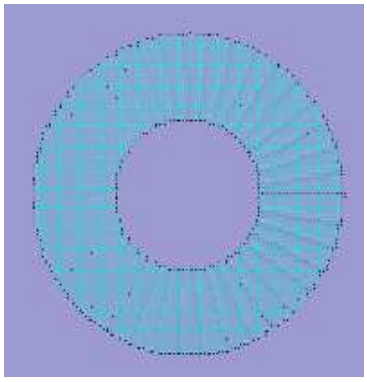
Figura D.1: Visualización por medio de aplanados de una flor.



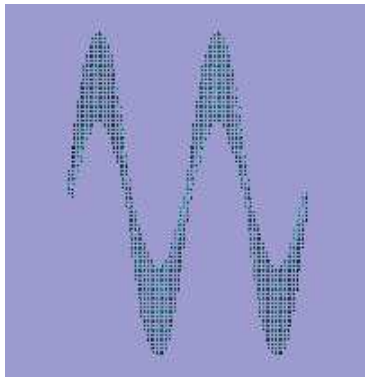
Densidad 1 correspondiente al helicoide.



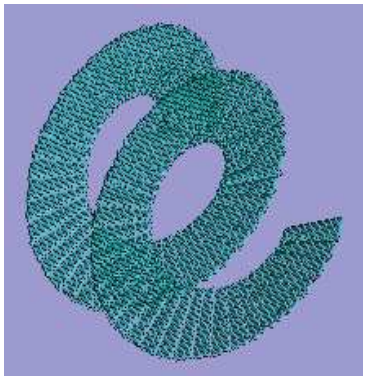
Densidad 2 correspondiente al cilindro.



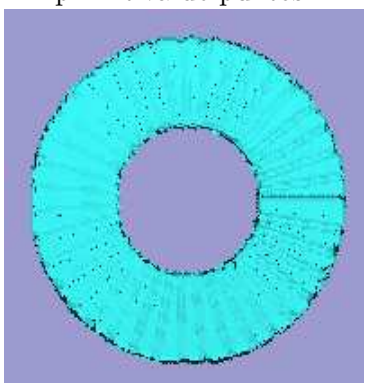
Vista frontal utilizando la primitiva de puntos.



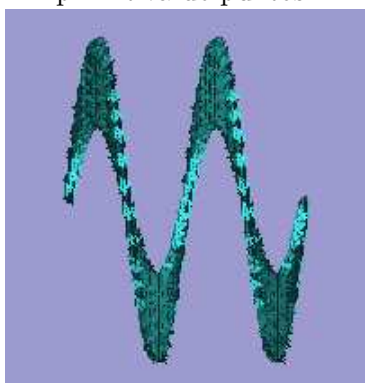
Vista lateral utilizando la primitiva de puntos.



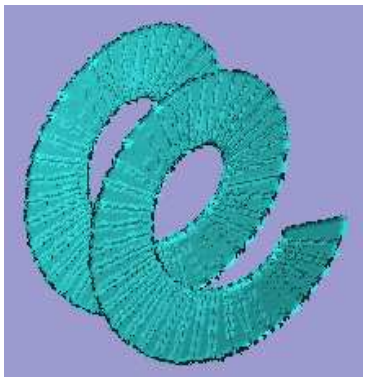
Vista general utilizando la primitiva de puntos.



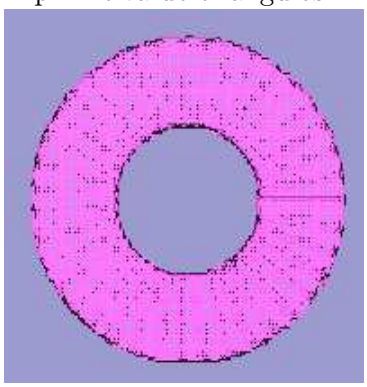
Vista frontal utilizando la primitiva de triángulos.



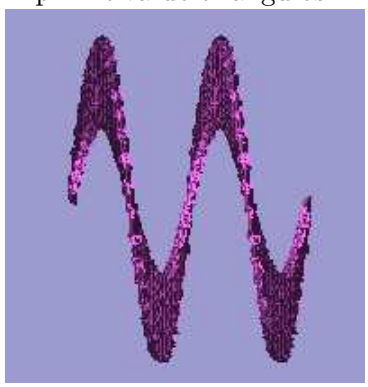
Vista lateral utilizando la primitiva de triángulos.



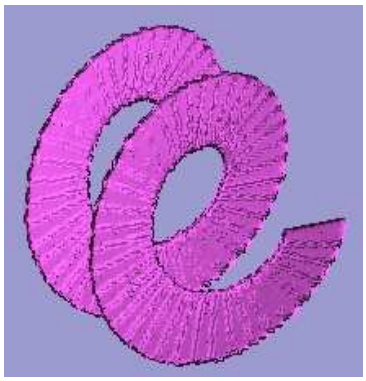
Vista general utilizando la primitiva de triángulos.



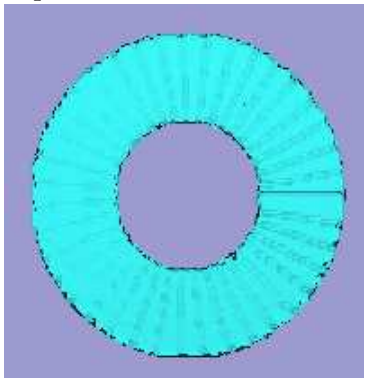
Vista frontal utilizando la primitiva de cuadrados.



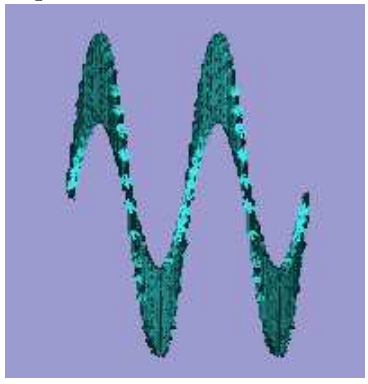
Vista lateral utilizando la primitiva de cuadrados.



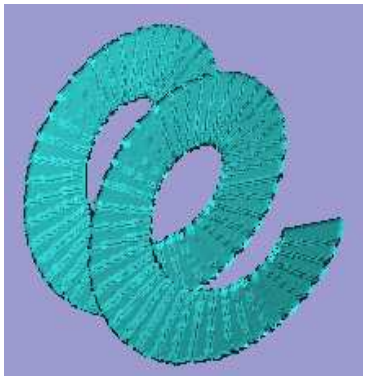
Vista general utilizando la primitiva de cuadrados.



Vista frontal utilizando la primitiva de círculos.

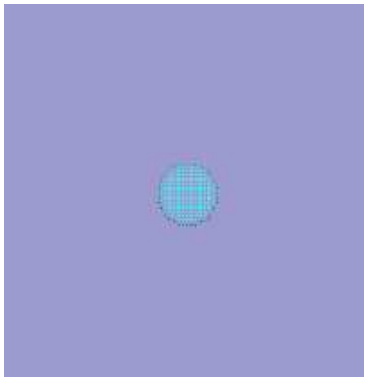


Vista lateral utilizando la primitiva de círculos.

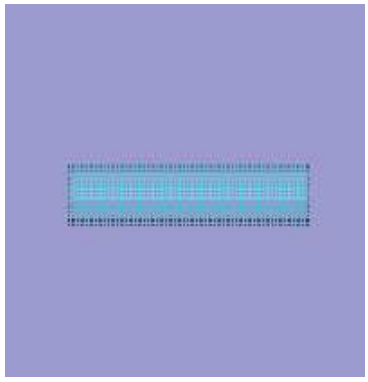


Vista general utilizando la primitiva de círculos.

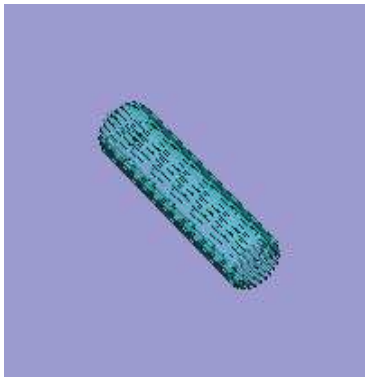
Figura D.2: Visualización por medio de aplanados de un helicode.



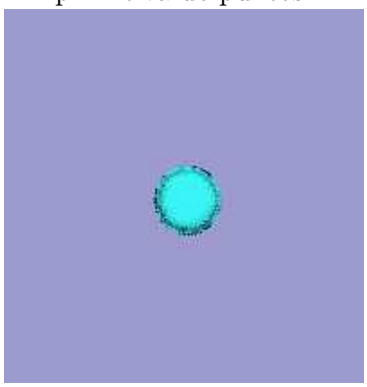
Vista frontal utilizando la primitiva de puntos.



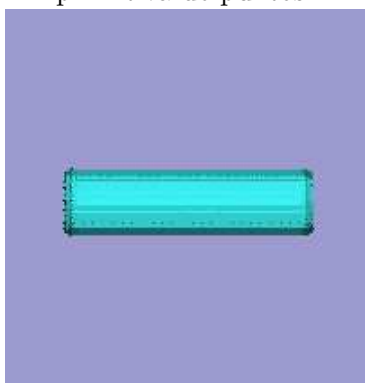
Vista lateral utilizando la primitiva de puntos.



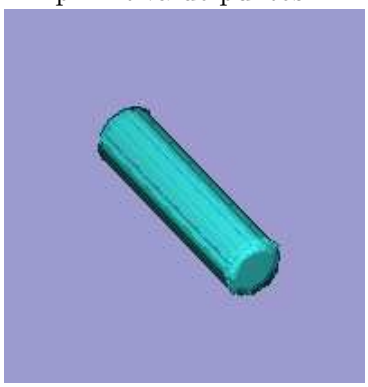
Vista general utilizando la primitiva de puntos.



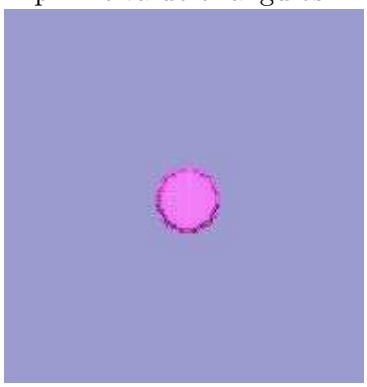
Vista frontal utilizando la primitiva de triángulos.



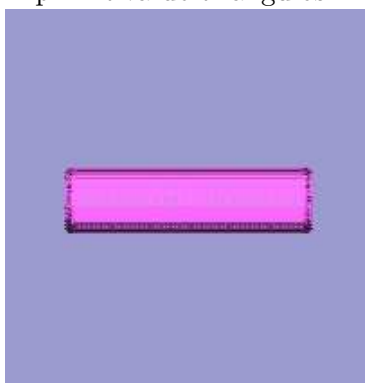
Vista lateral utilizando la primitiva de triángulos.



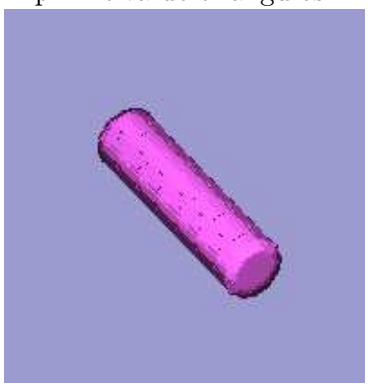
Vista general utilizando la primitiva de triángulos.



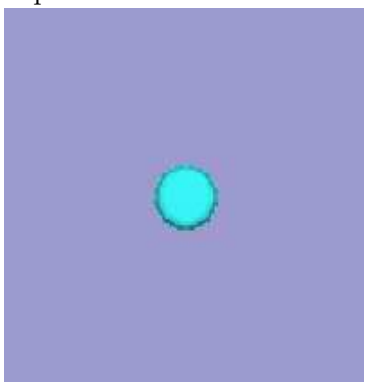
Vista frontal utilizando la primitiva de cuadrados.



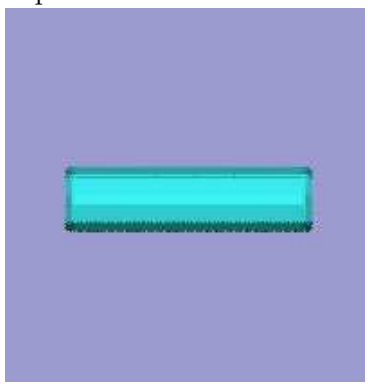
Vista lateral utilizando la primitiva de cuadrados.



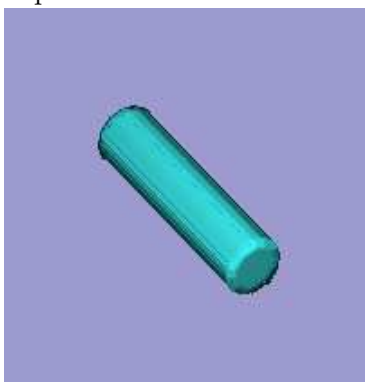
Vista general utilizando la primitiva de cuadrados.



Vista frontal utilizando la primitiva de círculos.

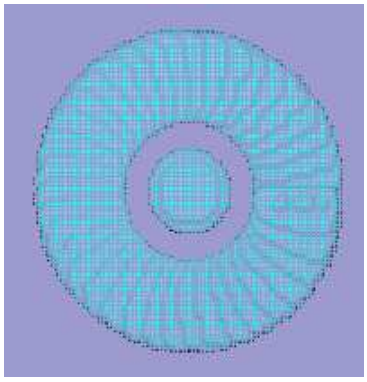


Vista lateral utilizando la primitiva de círculos.

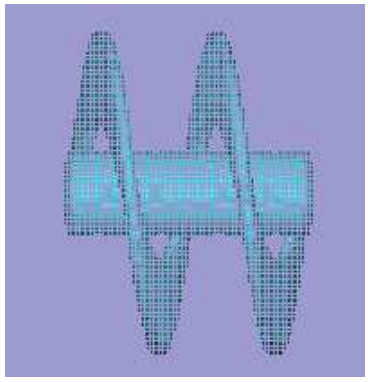


Vista general utilizando la primitiva de círculos.

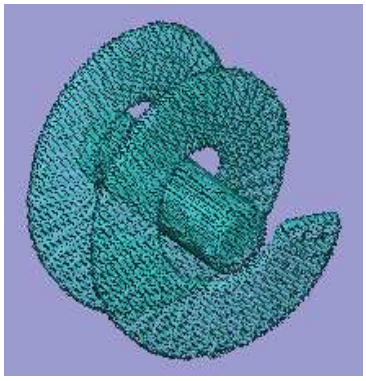
Figura D.3: Visualización por medio de aplanados de un cilindro.



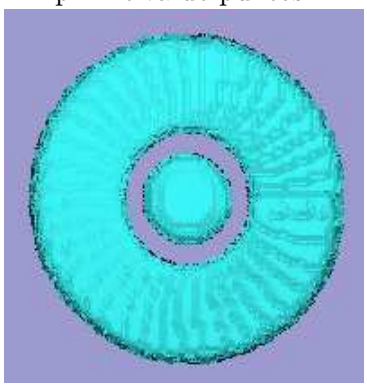
Vista frontal utilizando la primitiva de puntos.



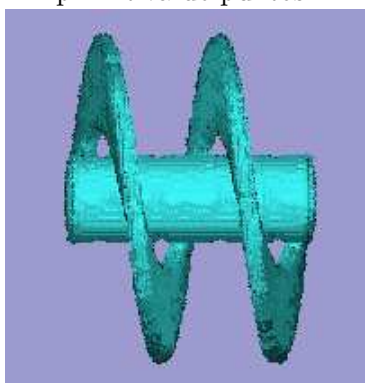
Vista lateral utilizando la primitiva de puntos.



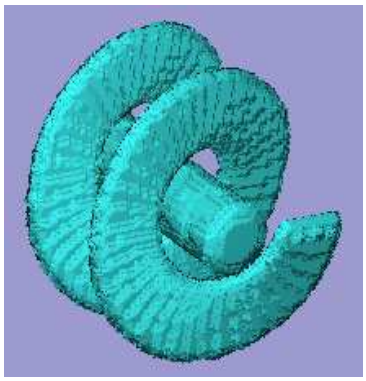
Vista general utilizando la primitiva de puntos.



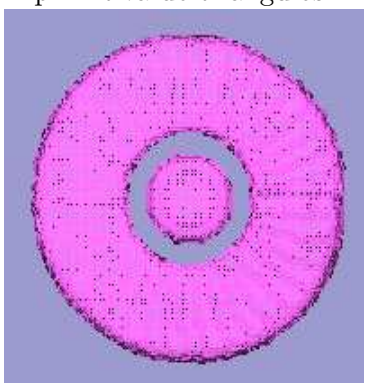
Vista frontal utilizando la primitiva de triángulos.



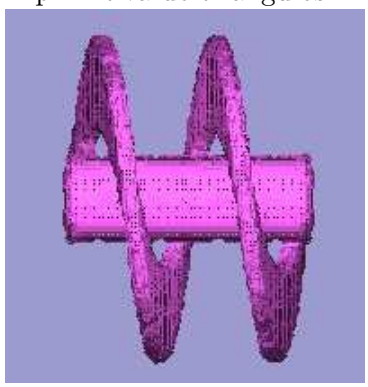
Vista lateral utilizando la primitiva de triángulos.



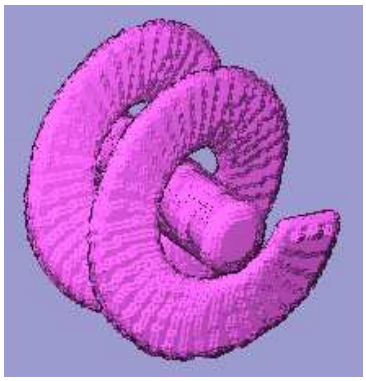
Vista general utilizando la primitiva de triángulos.



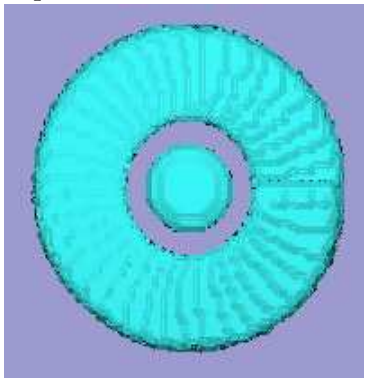
Vista frontal utilizando la primitiva de cuadrados.



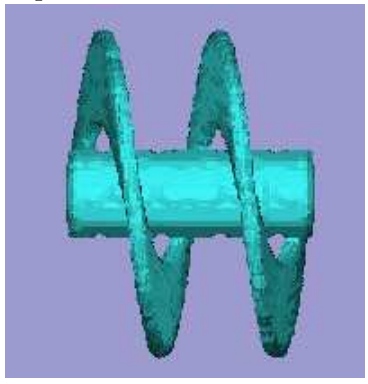
Vista lateral utilizando la primitiva de cuadrados.



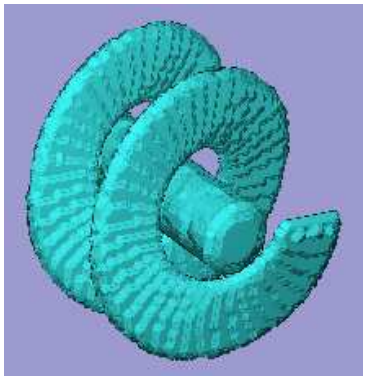
Vista general utilizando la primitiva de cuadrados.



Vista frontal utilizando la primitiva de círculos.

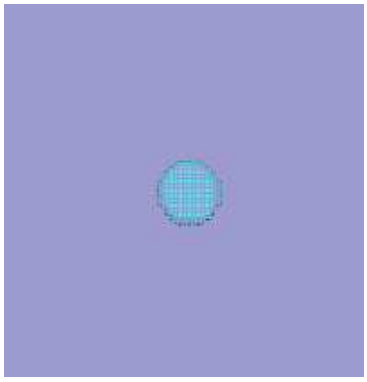


Vista lateral utilizando la primitiva de círculos.

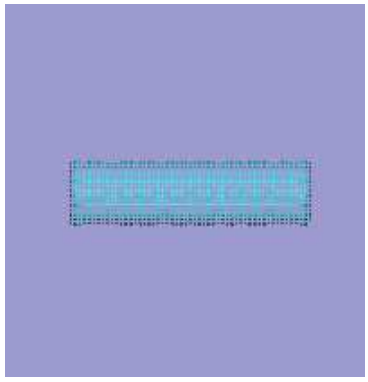


Vista general utilizando la primitiva de círculos.

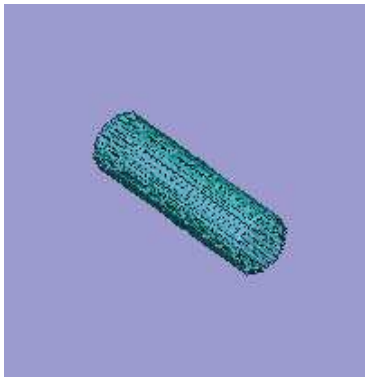
Figura D.4: Visualización por medio de aplanados de la reconstrucción un helicoide.



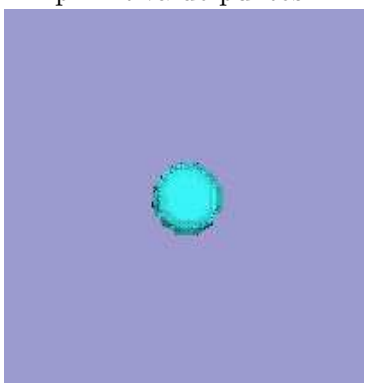
Vista frontal utilizando la primitiva de puntos.



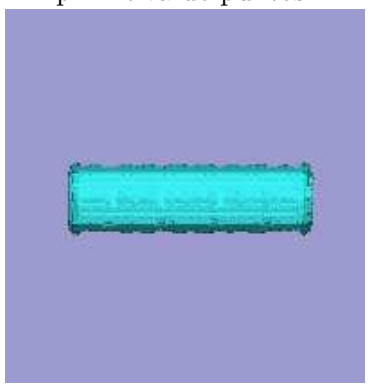
Vista lateral utilizando la primitiva de puntos.



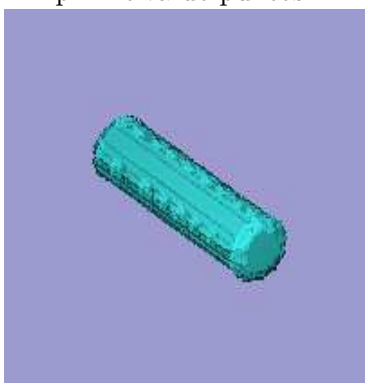
Vista general utilizando la primitiva de puntos.



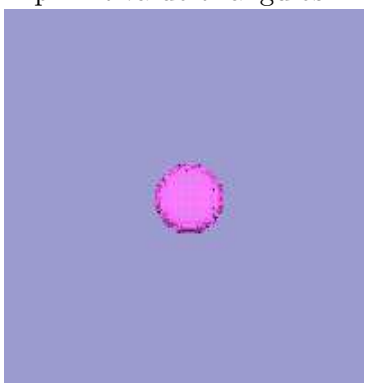
Vista frontal utilizando la primitiva de triángulos.



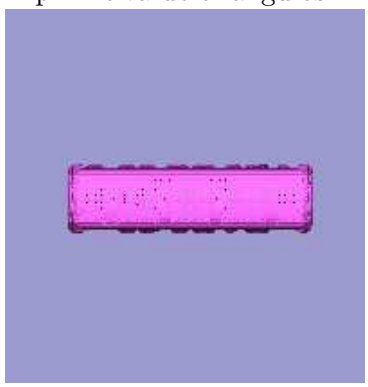
Vista lateral utilizando la primitiva de triángulos.



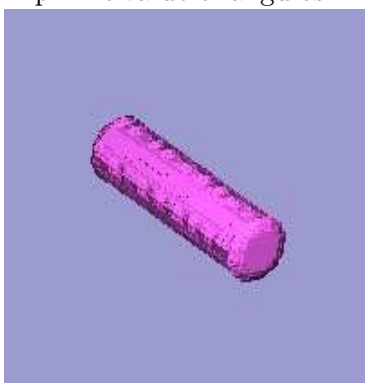
Vista general utilizando la primitiva de triángulos.



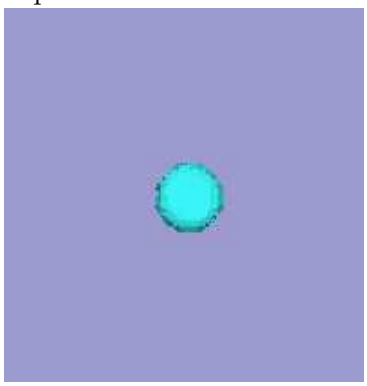
Vista frontal utilizando la primitiva de cuadrados.



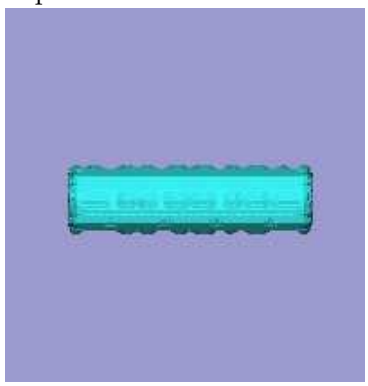
Vista lateral utilizando la primitiva de cuadrados.



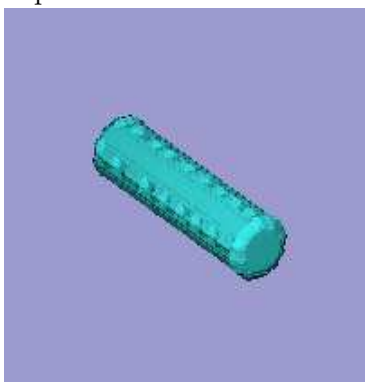
Vista general utilizando la primitiva de cuadrados.



Vista frontal utilizando la primitiva de círculos.

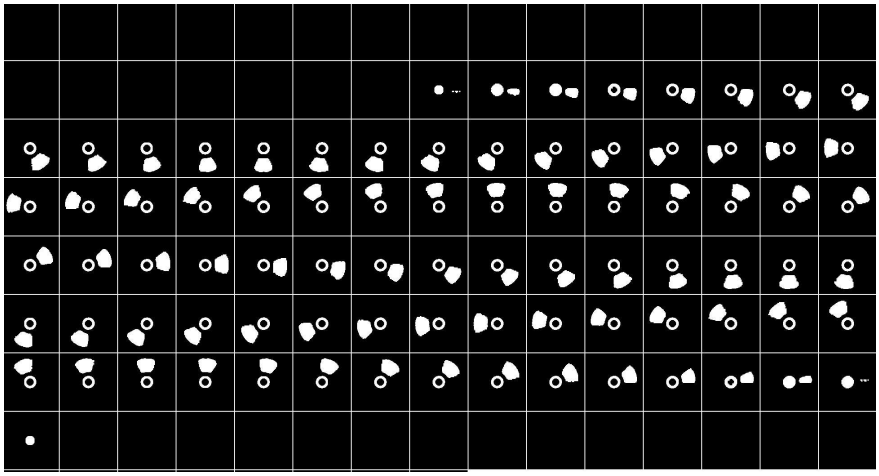


Vista lateral utilizando la primitiva de círculos.

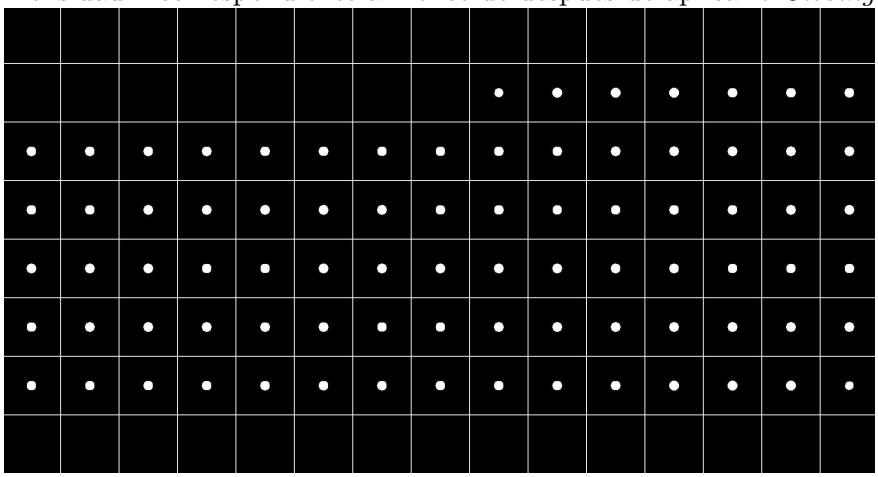


Vista general utilizando la primitiva de círculos.

Figura D.5: Visualización por medio de aplanados de la reconstrucción un cilindro.



Densidad 1 correspondiente al helicoide después de aplicarle *Closing*.



Densidad 2 correspondiente al cilindro después de aplicarle *Closing*.

Bibliografía

[1] Elke Van de Casteele. *Model-based approach for Beam Hardening Correction and Resolution Measurements in Microtomography.* PhD thesis, Universiteit Antwerpen, Antwerpen, 2004.

[2] Gabor T. Herman. *Image Reconstruction From Projections.* Academic Press, Inc, Orlando, Florida 32887, 1980.

[3] L.G. de la Fraga. *Refinamiento de la Estructura Tridimensional de Macromoléculas Biológicas Aisladas: Principios y Realización en Multiprocesadores.* PhD thesis, Centro Nacional de Biotecnología, Madrid, España, 1998.

[4] Mark Pauly, Leif P. Kobbelt, and Markus Gross. Point-Based Multiscale Surface Representation. *ACM Transactions on Graphics*, 25:177–193, 2006.

[5] Marching cubes. disponible en: http://www.exaflop.org/docs/marchcubes/ind.html.

[6] Jussi Rasanen. Surface Splatting: Theory, Extensions and Implementation. Master's thesis, Helsinki University of Technology, 2002.

[7] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA splatting. *j-IEEE-trans-vis-comput-graph*, 8(3):223–238, July/September 2002.

[8] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface Splatting. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 371–378. ACM Press / ACM SIGGRAPH, 2001.

[9] Delta de dirac. disponible en: http://mathworld.wolfram.com/deltafunction.html.

[10] John C. Russ. *The Image Processing Handbook*, chapter 9, 10. crc press & IEEE press, 3. edition, 1999.

[11] Ángulos de euler. disponible en: http://mathworld.wolfram.com/eulerangles.html.

[12] Edwin L. Dove. Notes on Computerized Tomography, 2001. www.engineering.uiowa.edu/ bme060/Lecture/CTNotes.pdf.

[13] Transformada de radon. disponible en: http://www.owlnet.rice.edu/ elec539/projects97/cult/node4.html.

[14] R. Marabini, I.M. Masegosa, M. C. San Martin, S. Marco, J.J. Fernandez, L.G. de la Fraga, C. Vaquerizo, and J. M. Carazo. Xmipp: An Image Processing Package for Electron Microscopy. *Journal of Structural Biology*, 116:237–240, 1996.

[15] Kenneth R. Castleman. *Digital Image Processing.* Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 1996.

[16] Laura Fritz. Interactive Diffusion-Based Volume Segmentation using Cg, a High-Level Shading Language. Master's thesis, Vienna University of Technology, Vienna, 2006.

[17] Dzung L. Pham, Chenyang Xu, and Jerry L. Prince. A survey of current methods in medical image segmentation. Technical report, The Jhons Hopkins University, 1999.

[18] Rui Xu and Donald Wunsch. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16:645–678, 2005.

[19] Daniel Fasulo. An analysis of recent work on clustering algorithms, 1999. http://www.cs.washington.edu/homes/dfasulo/clustering.ps.

[20] K-means. disponible en: http://people.revoledu.com/kardi/tutorial/kmean/index.html.

[21] The University of Tokyo, Institute of Medical Science, Human Genome Center. *The C Clustering Library for cDNA microarray data*, 2005. c2b2.columbia.edu.

[22] Matthias Zwicker. *Continuous Reconstruction, Rendering, and Editing of Point-Sampled Surfaces.* PhD thesis, Swiss Federal Institute of Technology, ETH, Zurich, 2003.

[23] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers and Graphics*, 28:801–814, 2004.

[24] Perl. disponible en: www.perl.org.

[25] Gnuplot. disponible en: http://www.gnuplot.info.

[26] Plane. disponible en: http://mathworld.wolfram.com/plane.html.

[27] Willian H.Press, Saul A. Teukolsky, William T. Vetterlin, and Brian P. Flanneryg. *Numerical Recipes in C: The Art of Scientific Computing*, chapter 2. Cambridge University Press, second edition edition, 1988-1992.

[28] Qt. disponible en: http://doc.trolltech.com/3.0/index.html.

[29] Opengl. disponible en: http://www.opengl.org/.

[30] Widget. disponible en: http://es.wikipedia.org/wiki/widget.

[31] Luis Gerardo de la Fraga and Feliú Sagols Troncoso. Scubes: A Program to Visualize Vox-Solids. In *VII Conferencia de Ingeniería Eléctrica.* CINVESTAV, 2001.

[32] Jorge Eduardo Ramírez Flores and Luis Gerardo de la Fraga. Basic three-dimensional objects constructed with simplex meshes. In *2004 1st International Conference on Electrical and Electronics Engineering. (ICEEE 2004)*, pages 166–171. Sep 8-10, Acapulco, Gro., 2004.

[33] Lin Yu Tseng and Shiueng Bien Yang. A genetic approach to the automatic clustering problem. *Pattern Recognition*, 34:415–424, 2001.

[34] Petersen & Pedersen. The Matrix Cookbook, February 17 2006. http://matrixcookbook.com/.

[35] Jon Dattorro. Convex Optimization and Euclidean Distance Geometry, 2005. http://www.stanford.edu/ dattorro/matrixcalc.pdf.

[36] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[37] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, New York, NY, USA, 2000.

[38] Eigenvector. disponible en: http://mathworld.wolfram.com/eigenvector.html.