



UNIVERSIDAD NACIONAL  
AUTÓNOMA DE  
MÉXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

“VISUALIZACIÓN SUAVIZADA DE SUPERFÍCIES  
OBTENIDAS POR RASTREO DE FRONTERAS  
APLICADO A VOLÚMENES DISCRETIZADOS”

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS  
(COMPUTACIÓN)

P R E S E N T A

JORGE ANTONIO GARCÍA GALICIA

DIRECTOR DE TESIS:

DR. EDGAR GARDUÑO ÁNGELES

México, D.F.

2011

*A todos aquellos que de alguna manera  
tienen algo que ver con las gráficas por computadora.*

# Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT), por apoyar esta investigación con la beca CVU-266004 para estudios de Maestría en la Universidad Nacional Autónoma de México.

La tesis fué realizada bajo la supervisión de Edgar Garduño, quien me invito a participar en este proyecto. Agradezco enormemente su apoyo, guía y paciencia.

Estoy agradecido con mis sinodales: Jorge Urrutia, Fernando Arámbula, Jorge Márquez y Ernesto Bribiesca, por las importantes recomendaciones, sugerencias y correcciones. Sin duda hicieron que este trabajo mejorara su calidad.

Agradezco también al Departamento de Ciencias de la Computación del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas por permitirme hacer uso de sus instalaciones y por todos los recursos materiales con los que me apoyaron.

Y por supuesto, gracias...

A mi mamá (q. e. p. d.), mi papá y mis hermanos, por siempre haber creído en mi y por haberme apoyado no solo en este trabajo, si no en todos los proyectos que he emprendido en la vida.

A Erika, por las platicas, el apoyo, los momentos compartidos, la solidaridad y por ser la persona tan maravillosa que es.

A Magali, Laura y Jose Luis, mis *roomates*, por haberme permitido compartir su vida y su amistad.

A mi grupo de trabajo: Etna, Fátima, Eduardo, Verena y Cinthya. Por la compañía, por haberme rebotado ideas y por siempre haber estado dispuestos a ayudarme.

A los *inges*: Sergio, Federico y Omar, por haber sido mis compañeros y amigos. Y por haberme apoyado en esos momentos tan difíciles que me tocó pasar. Agradezco también mis compañeros

de clase durante la maestría. En especial a Tzolkin, Carlos Alegría, Eliza, Uriel, Marlene, Nahela, Paty, Jaime Cabrera, Toño, Sebastian Bejos y Sonia. Por haberme permitido estudiar a su lado, por todo lo que les aprendí y por su amistad.

A mi amigos: José Raul, Sergio (maría), Jorge (primo), Bernardo, Onatta, Yesenia, Karina, Mariano y Chac. Primero por su amistad, luego por todas las veces que me escucharon y por último por haber tenido siempre tiempo de preguntarme: *¿Cómo va tu tesis?*

A Carmen Villar, por su amistad y por haberme mostrado por primera vez el maravilloso mundo de las gráficas por computadora.

A toda la gente que paga sus impuestos y que cumple con su trabajo en paz.

# Resumen

Este trabajo trata de como visualizar campos escalares que han pasado por un proceso de digitalización. En particular se usa una técnica de graficación por computadora conocida como visualización por superficies o *surface rendering*.

Se asume que se tienen conjuntos digitales de datos que provienen de haber muestreado de manera uniforme el espacio en tres dimensiones. Asumimos que este muestreo está hecho en una rejilla rectangular y por lo tanto tenemos una imagen digital en 3D o volumen. Hacemos dos suposiciones importantes sobre el volumen. Primero, que es una buena aproximación del campo escalar y segundo que no tenemos información de la manera como se realizó la digitalización.

Primeramente revisamos dos algoritmos de rastreo de superficies sobre volúmenes. El algoritmo de *Marching Cubes* que es el mas usado en la actualidad y el Algoritmo de Artzy cuya salida posee características deseables desde el punto de vista topológico. Ambos algoritmos producen una malla poligonal que aproxima una superficie del campo escalar original.

El problema en adelante es visualizar correctamente estas mallas. Se incluye una revisión de algunas técnicas de graficación por computadora para visualizar mallas. En particular nos enfocamos a la iluminación con el modelo de Phong y a las técnicas basadas en mapas; tales como el mapeo de texturas y el mapeo de relieve (*bump mapping*).

Se explica también la creación de superficies implícitas o modelo blobby usado comúnmente en graficación para hacer modelado orgánico. Se revisan algunas funciones base que se usan con este modelo. Se presentan las funciones Kaiser-Bessel generalizadas también llamadas *blobs*.

El objetivo de este trabajo consiste en encontrar una forma de mejorar la visualización

de la malla del Algoritmo de Artzy y hacerla equiparable con el algoritmo de *Marching Cubes* sin modificar la malla y usando solamente efectos de iluminación.

La principal aportación del trabajo es un algoritmo para encontrar una superficie implícita formada por *blobs* que envuelve la malla del Algoritmo de Artzy. Por medio de esta superficie calculamos vectores normales que luego ponemos en los vértices de la malla y usamos para iluminación.

Por último, se reportan resultados de algunos experimentos con esta técnica. Primera-mente se realizan experimentos en conjuntos de datos obtenidos por medio de técnicas de imagenología biomédica, estos resultados constituyen una prueba visual de que la técnica propuesta funciona. Por ultimo se hacen experimentos sobre campos escalares conocidos (*phantoms*) y se comparan las normales obtenidas por nuestro método con las normales analíticas de los *phantoms*. En la última sección del trabajo se reportan las conclusiones y se proponen algunas líneas de investigación para trabajo futuro.

# Índice general

<b>Introducción</b>	<b>1</b>
<b>1 Visualización de imágenes 3D</b>	<b>5</b>
1.1 Generación de mallas . . . . .	11
1.1.1 Adyacencias y caminos . . . . .	12
1.1.2 El algoritmo de <i>Marching Cubes</i> de Lorensen y Cline . . . . .	14
1.1.3 El Algoritmo de seguimiento de superficies de Artzy . . . . .	17
1.1.3.1 Condiciones del Algoritmo de Artzy . . . . .	17
1.1.3.2 Implementación del algoritmo . . . . .	19
1.1.3.3 Propiedades del Algoritmo de Artzy . . . . .	20
1.2 Visualización de mallas . . . . .	22
1.2.1 Iluminación . . . . .	22
1.2.1.1 Tipos de fuentes de luz . . . . .	23
1.2.2 Modelos de iluminación . . . . .	24
1.2.2.1 Modelo de Phong . . . . .	24
1.2.2.2 La reflexión ambiental . . . . .	25
1.2.2.3 La reflexión difusa . . . . .	25
1.2.2.4 La reflexión especular . . . . .	26

1.2.2.5	Modelo completo . . . . .	28
1.2.3	Modelos de sombreado . . . . .	30
1.2.3.1	Sombreado de Gouraud . . . . .	31
1.2.3.2	Sombreado de Phong . . . . .	32
1.2.4	Efectos a partir de mapas para aumentar el realismo de la escena	34
1.2.4.1	Mapeo de texturas . . . . .	34
1.2.4.2	Mapeo de relieves . . . . .	36
<b>2</b>	<b>Ponderado de normales</b>	<b>41</b>
2.1	Representación implícita de superficies . . . . .	41
2.1.1	Las funciones Kaiser-Bessel generalizadas ( <i>blobs</i> ) . . . . .	44
2.2	Contribución de la tesis . . . . .	48
2.2.1	Optimización de blobs para visualización . . . . .	49
2.2.2	Ponderado de normales por medio de blobs . . . . .	54
<b>3</b>	<b>Experimentos</b>	<b>59</b>
3.1	Experimentos visuales . . . . .	60
3.2	Experimentos con maniquíes ( <i>phantoms</i> ) . . . . .	60
3.2.1	Medida de error . . . . .	64
3.2.2	Criterio de muestro . . . . .	64
<b>Conclusiones</b>		<b>71</b>
<b>Bibliografía</b>		<b>77</b>

# Índice de figuras

1.1	Cuerpo humano dividido en planos sagital, axial y coronal (La imagen fue tomada de [48]) . . . . .	7
1.2	Un volumen o imagen 3D visualizada por planos sagitales de una cabeza obtenida por medio de MRI (obtenida de [46]) . . . . .	8
1.3	El mismo conjunto de datos de la Figura 1.2 visualizado por Volume Rendering . . . . .	9
1.4	Visualización por superficies de la imagen 3D de la Figura 1.2 con cuatro isovalores distintos . . . . .	10
1.5	Dos tipos de adyacencias típicas para vóxeles cúbicos . . . . .	13
1.6	Vecindades de un vóxel cúbico simple . . . . .	13
1.7	El algoritmo de MC utiliza un cubo formado con los centros de ocho vóxeles contiguos para recorrer el volumen . . . . .	15
1.8	Diferentes situaciones en el algoritmo de Marching Cubes . . . . .	15
1.9	Ejemplo de superficie obtenida por Marching Cubes . . . . .	16
1.10	Rutas validas en el Algoritmo de Artzy en la rejilla <i>sc</i> . . . . .	18
1.11	Cambio de caras en el Algoritmo de Artzy . . . . .	19
1.12	Ejemplo de superficie obtenida por el Algoritmo de Artzy . . . . .	21
1.13	Diferentes tipos de luz en una escena . . . . .	23
1.14	Reflexión difusa . . . . .	26

1.15	Reflexión especular . . . . .	27
1.16	Situación geométrica del modelo de Phong . . . . .	27
1.17	Componentes del modelo de iluminación de Phong . . . . .	29
1.18	Ejemplo de sombreado sobre un triangulo . . . . .	31
1.19	Comparación entre los modelos de sombreado de Gouraud y de Phong	34
1.20	Mapeo de texturas . . . . .	35
1.21	Ejemplo de mapeo de relieves . . . . .	37
1.22	Diagrama geométrico del mapeo de relieves . . . . .	38
2.1	Ilustración del efecto de <i>blobbiness</i> para diferentes radios $r$ y $\alpha$ de blobs de Blinn . . . . .	43
2.2	Comparación entre las diferentes funciones blobs . . . . .	44
2.3	Espectro del blob en escala logarítmica . . . . .	46
2.4	Ilustración de diferentes blobs con $a = 1$ y $m = 2$ variando $\alpha$ . . . . .	46
2.5	Comparación entre el blob y su gradiente . . . . .	48
2.6	Rejillas cúbicas . . . . .	50
2.7	Contorno de la superficie formada por dos blobs con los mismos parámetros en puntos vecinos de una rejilla y umbralizada a $t = 0.5$ . . . . .	53
2.8	Ejemplo de colocación de los blobs para visualización . . . . .	56
3.1	Comparación visual de superficies formadas a partir de la reconstrucción de una langosta . . . . .	61
3.2	Comparación visual de superficies de la reconstrucción de una langosta. Son las mismas mallas que las de la Figura 3.1 vistas desde otro punto de vista . . . . .	62
3.3	Comparación visual en el modelo de una cabeza humana . . . . .	63
3.4	Gráfica de la función $J_1$ en la frecuencia . . . . .	65

3.5 Esquema para la comparación de normales en la superficie de la esfera	66
3.6 Resultados visuales de la ponderación de normales en el caso de la esfera	68



# Índice de cuadros

2.1	Resultados de optimizar $a$ y su correspondiente $\alpha$ para las diferentes rejillas . . . . .	54
3.1	Propiedades del material con el que se visualizan las mallas . . . . .	59
3.2	Discretizaciones correspondientes para distintos ceros de la función $J_1(\xi)$	66
3.3	Resultados del experimento numérico sobre una esfera . . . . .	67
3.4	Resultados del experimento numérico sobre un cubo . . . . .	67

# Introducción

La percepción que tenemos los seres humanos del mundo a través de nuestros sentidos es inmediata. Además, dependemos de estos sentidos para realizar la mayoría de las actividades diarias. También hay que señalar que somos criaturas visuales, de nuestros cinco sentidos el que nos proporciona la mayor cantidad de la información que asimilamos sin lugar a dudas son nuestros ojos [24].

Desde hace más de tres décadas las computadoras se han vuelto más poderosas y hemos tenido la oportunidad de procesar con ellas una mayor cantidad de datos de manera rápida. Una de las áreas que se ha encontrado más beneficiada es la imagenología biomédica. Si bien es cierto que los rayos X eran desde hace mucho una herramienta probada para el diagnóstico médico (y diversas aplicaciones industriales), una imagen digital que pudiera almacenarse, procesarse y luego desplegarse en una computadora ha sido un parteaguas para este campo. Al mismo tiempo, muchas áreas de las ciencias de la computación (CC) han cobrado interés durante el mismo periodo. Particularmente, el procesamiento de imágenes digitales, las gráficas por computadora y la visualización científica, son ahora una opción viable para usarse rutinariamente.

El procesamiento de imágenes estudia como procesar imágenes digitales por medios computacionales [21]. Algunas veces este proceso implica transformar una imagen en otra imagen en donde se ha resaltado, o inhibido, cierta característica. Otras veces implica obtener otro tipo de información de una imagen, por ejemplo un histograma. Finalmente, algunas veces implica poder distinguir o interpretar datos de objetos a partir de las imágenes.

Las gráficas por computadora (GC) son la rama de las CC que estudia las técnicas para producir imágenes digitales a partir de descripciones matemáticas. Estas imágenes son visualizadas en dispositivos de despliegue en 2D, por ejemplo un monitor, por lo que

generalmente es necesario hacer una proyección de los objetos (en tres dimensiones) a un plano.

La visualización científica se encarga de construir información visual de conjuntos de datos científicos. Estos datos pueden tener orígenes muy diversos; inclusive pueden simular fenómenos de la naturaleza que no pueden ser percibidos por nuestra vista. Por ejemplo, ver en la pantalla el espacio de solución de una ecuación diferencial que modele la vibración de la cuerda de una guitarra es un problema de visualización científica que se podría equiparar con la idea de “visualizar la música”.

En éste trabajo se abarca un poco de estas tres áreas. Asumimos que tenemos un conjunto de datos distribuidos en un arreglo cúbico que representan un muestreo de un campo escalar tridimensional. Estos datos forman lo que típicamente se conoce como una imagen digital en 3D o volumen. Debido a la naturaleza de los datos nos interesa poder visualizarlos sin perder su información espacial.

Una forma de lograr éste objetivo es encontrar una superficie que defina la frontera de un objeto y luego visualizar la superficie como si ésta fuera un envolvente. Como en un principio se tiene la imagen en 3D es necesario utilizar algún criterio que defina la superficie (proceso típicamente conocido como segmentación). Dado la naturaleza del volumen debemos encontrar una manera de generar un conjunto de polígonos para aproximar la superficie de manera digital.

En este trabajo estudiaremos dos métodos para aproximar esas superficies: el algoritmo de *Marching Cubes* [34], que es el mas usado actualmente, y el Algoritmo de Artzy [2]. El Algoritmo de Artzy tiene varias ventajas sobre el primero, la mas importante es que garantiza que la superficie aproximada es topológicamente correcta, es decir que no contiene agujeros. Sin embargo, la desventaja mas importante de Artzy es que las mallas obtenidas parecen cuboides, como si fueran hechas por bloques del juego de LEGO.

Sin importar el método de obtención de la malla, se visualiza por medio de técnicas de graficación por computadora. Esto incluye utilizar *iluminación*, que en conjunto con otras técnicas, tiene la función de dar la apariencia de 3D a la proyección en 2D. Para poder utilizar la iluminación es necesario que proporcionemos un conjunto de vectores normales a la superficie en cada vértice de la malla. Como es mostrado en la técnica conocida como *bump mapping* [9], estos vectores pueden afectar significativamente la

apariencia final de la imagen.

En este trabajo proponemos una forma de obtener un conjunto de vectores normales en los vértices de la malla obtenida por el Algoritmo de Artzy, tales que al usarse en conjunto con la iluminación, el resultado final sea una apariencia mas suave del objeto. Por lo tanto se obtendría una malla que conserva las ventajas del Algoritmo de Artzy pero que sea visualmente equiparable con la malla producida por *Marching Cubes*. Para esto, pensamos encontrar un envolvente suave de la malla por medio de funciones base de transición suave de uno a cero; conocidas como Kaiser Bessel generalizadas.

La organización del trabajo es la siguiente. En el primer capítulo, se hace la fundamentación matemática necesaria de lo que entendemos como volumen y se explican que condiciones esperamos que cumplan los volúmenes de los que hablamos en el resto del trabajo. También se analizan los dos algoritmos de rastreo de superficies sobre volúmenes antes expuestos poniendo énfasis en las propiedades presentes en el Algoritmo de Artzy. También se revisa el modelo de iluminación de Phong y se habla de los modelos de sombreado de Phong y de Gouraud. Por último se revisan dos técnicas de GC que usan mapas para dar efectos: el mapeo de texturas y el mapeo de relieves (*bump mapping*).

En el segundo capítulo se introducen las superficies implícitas y algunas funciones base que comúnmente se usan para crear dichas superficies. Luego se introducen las funciones Kaiser Bessel generalizadas que servirán de funciones base en el resto del trabajo. También se exponen las contribuciones del trabajo. La primera es que se sigue el método de optimización para funciones base propuesto en [17] y se llegan a parámetros óptimos para tres casos. La segunda aportación es que se obtiene un algoritmo para crear la superficie implícita que envuelva la malla producida por el Algoritmo de Artzy y como se le asignan normales a los vértices de esa malla. Esta última es la principal aportación del trabajo.

El tercer capítulo esta dedicado a los experimentos realizados para probar la metodología expuesta. En primer lugar se muestran resultados puramente visuales sobre algunos conjuntos de datos provenientes de imagenología biomédica. Posteriormente, se explica la manera como se construyen dos *phantoms* (un cubo y una esfera) a diferentes resoluciones y se evalúan las diferencias entre las normales analíticas de los *phantoms* y las obtenidas por nuestro método.

Por último, en el capítulo final se presentan las conclusiones del trabajo y conjuntamente se proponen líneas de investigación futuras.

### **Objetivos del trabajo**

El principal objetivo de este trabajo es encontrar por medio de la modificación de normales una forma de hacer la superficie obtenida por el Algoritmo de Artzy visualmente más agradable. Haciendo énfasis en que no hacemos ninguna suposición sobre como fue la discretización del volumen.

# Capítulo 1

## Visualización de imágenes 3D

En éste trabajo estamos interesados en visualizar campos escalares discretizados. Un campo escalar es un mapeo  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  donde  $f$  típicamente representa el valor de una cantidad física en el espacio. En la actualidad muchos dispositivos, por ejemplo los tomógrafos, son capaces de producir muestreos de estos campos. Dada la naturaleza electrónica de los dispositivos con los que medimos el campo escalar, en realidad se tiene una aproximación de  $f$  que se obtiene, en general, muestreando de manera uniforme el espacio tridimensional.

Para representar un punto en  $\mathbb{R}^n$ , alternativamente  $\mathbb{Z}^n$ , usaremos la representación  $\mathbf{p} \in \mathbb{R}^n$ , alternativamente  $\mathbf{k} \in \mathbb{Z}^n$ , y su  $n$ -tupla  $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$  donde  $p_i \in \mathbb{R}$ , alternativamente  $\mathbf{k} = \{k_1, k_2, \dots, k_n\}$  donde  $k_i \in \mathbb{Z}$ . Debido a que usamos recursos finitos para llevar a cabo la discretización de  $f$  solo usaremos un subconjunto  $\Gamma \subset \mathbb{Z}^3$  definido de la siguiente manera:

$$\Gamma = \{\mathbf{k} | A_i \leq k_i \leq B_i \text{ para } 1 \leq i \leq 3 \text{ con } A_i < B_i \text{ y } A_i, B_i \in \mathbb{Z}\}. \quad (1.1)$$

Dado un punto  $\mathbf{k} \in \Gamma$  y un numero real positivo  $\Delta$ , definimos un vóxel cúbico como

$$\text{Vox}(\mathbf{k}) = \left\{ \mathbf{x} \in \mathbb{R}^3 | \Delta \left( k_i - \frac{1}{2} \right) \leq x_i < \Delta \left( k_i + \frac{1}{2} \right), \text{ para } 1 \leq i \leq 3 \right\}, \quad (1.2)$$

al número  $\Delta$  comúnmente se le conoce como distancia de muestreo[18]. El termino vóxel proviene del ingles *volume element* y por lo tanto pueden existir vóxeles no cúbicos. Al

al mismo tiempo un punto  $\mathbf{k} \in \mathbb{Z}^2$  tiene asociado un píxel, proveniente del inglés *picture element*. Debido a que en éste trabajo no utilizamos teselaciones no cúbicas, usaremos el término *vóxel* para referirnos a los *vóxeles* de (1.2).

Al subconjunto  $\Gamma$  de  $\mathbb{R}^3$  para el cual esta definido  $v$  se le llama *escena*. De manera convencional a cada uno de los límites de las dimensiones del dominio de  $v$  se le da un nombre. Típicamente al intervalo valido de  $k_1$  se le conoce como el *ancho* de la imagen, al intervalo de  $k_2$  como la *altura* de la imagen y al de  $k_3$  como la *profundidad* de la imagen.

Los *vóxeles* también pueden verse como la vecindad de Voronoi de un conjunto de puntos dispuestos como un arreglo rectangular que llena la escena [26], a este arreglo de puntos lo llamamos rejilla cúbica simple (*sc*). Aunque de momento solo se hace uso de rejillas *sc* estas no son las únicas formas de muestrear de manera uniforme el espacio, también existen otras rejillas como la *face-centered cubic grid* (*fcc*) o la *body-centered cubic grid* (*bcc*) que cubren el espacio de *vóxeles* con forma de rombo dodecaedros y de octaedros truncados respectivamente [26].

Dado un campo escalar  $f$  y un intervalo de muestreo  $\Delta$ , definimos una digitalización como

$$p_\Delta(\mathbf{k}) = \frac{1}{\Delta^3} \int_{\text{Vox}(\mathbf{k})} f(\mathbf{x}) d\mathbf{x}. \quad (1.3)$$

La digitalización definida en (1.3) da origen a una *imagen digital* en 3D (o *volumen*) que se define de la siguiente manera:

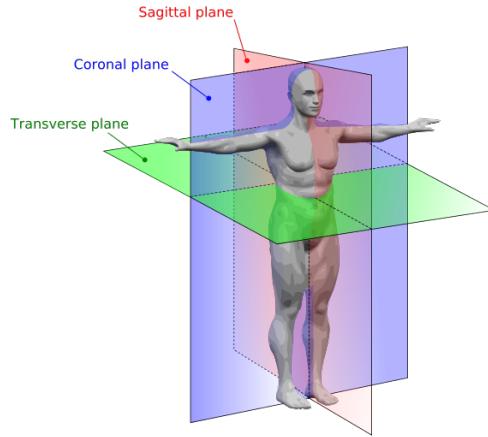
$$v(\mathbf{k}) = \begin{cases} p_\Delta(\mathbf{k}), & \text{si } \mathbf{k} \in \Gamma, \\ 0, & \text{en otro caso.} \end{cases} \quad (1.4)$$

Se espera que la imagen digital  $v$  sea una buena aproximación del campo escalar  $f$ . Para que esto último sea posible deben pasar dos cosas. Primero que la distancia  $|B_i - A_i|$  sean lo suficientemente grande como para abarcar todo el rango de interés del campo escalar  $f$  y segundo, que la distancia  $\Delta$  sea lo suficientemente pequeña para que los detalles importantes de la imagen no se pierdan debido a la digitalización expresada en (1.3) [18]. La principal limitante para lograr estas condiciones es que dependemos de la capacidad del dispositivo y de los algoritmos con los que se hace la digitalización. En

este trabajo, sin embargo, se asumirá que no se tiene control del dispositivo del que se obtienen las imágenes, pero que se han obtenido “buenas” aproximaciones.

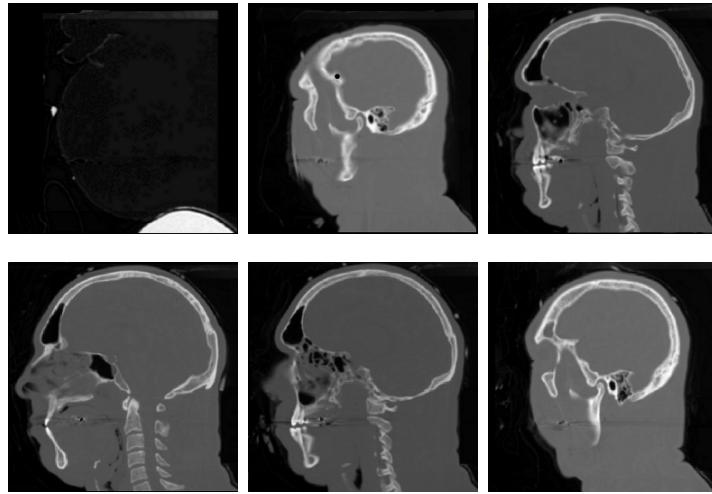
Al valor escalar  $p_{\Delta}(\mathbf{k})$  definido en (1.3) se le asocia una cantidad física que generalmente tiene que ver con la detección de la interacción de la materia que haya en el voxel  $\mathbf{k}$  con cierta radiación. Aunque los tomógrafos proporcionan un valor  $v(\mathbf{k}) = \lambda \in \mathbb{R}$ , algunas veces escalamos el rango de valores  $r_i \leq \lambda \leq r_f$  a un cierto intervalo  $[0, 2^i - 1]$  para algún entero positivo  $i$ .

En áreas como la medicina y la biología es común visualizar los volúmenes  $v$  por planos o cortes generados al dejar fijo algún valor  $k_i$  para alguna escena  $\Gamma$ . En medicina, si se trata de un volumen  $v$  que aproxima un ser humano visto de pie, es común llamar a estos cortes como *coronal* si divide al cuerpo humano en dos secciones adelante y atrás, *sagital* si divide al cuerpo humano en secciones izquierda y derecha y *axial* si lo divide en secciones arriba y abajo, ver Figura 1.1. Sin embargo, este tipo de visualización tiene la enorme desventaja de perder la estructura tridimensional de la imagen, ver la Figura 1.2.



**Figura 1.1:** Cuerpo humano dividido en planos sagital, axial y coronal (La imagen fue tomada de [48]).

Por esta razón se han desarrollado formas de hacer una visualización completa de las imágenes que preserve la estructura 3D. Hay dos métodos generales de atacar este problema. La primera forma es conocida como visualización directa o *volume rendering*. Este método fue originalmente propuesto por Drebin, Loren y Hanrahan en [13] y hace uso de una técnica de GC conocida como *alpha blending*. El método consiste en asociar a los voxels en la imagen  $v$  una cierta función de transferencia y con ella calcular para



**Figura 1.2:** Un volumen o imagen 3D visualizada por planos sagitales de una cabeza obtenida por medio de MRI (obtenida de [46]).

cada voxel  $\mathbf{k}$  un valor alfa (o de transparencia), de esta manera voxels con valores  $p_\Delta(\mathbf{k})$  diferentes adquieren diferentes grados de opacidad. Esto permite ver todo el volumen directamente como una colección de objetos translúcidos. Un ejemplo se muestra en la Figura 1.3.

El otro método de visualización se llama *visualización indirecta* o por superficies. Dado un campo escalar  $f(\mathbf{x})$ , donde  $f$  es una función escalar de  $\mathbb{R}^3$ , se define una superficie como sigue

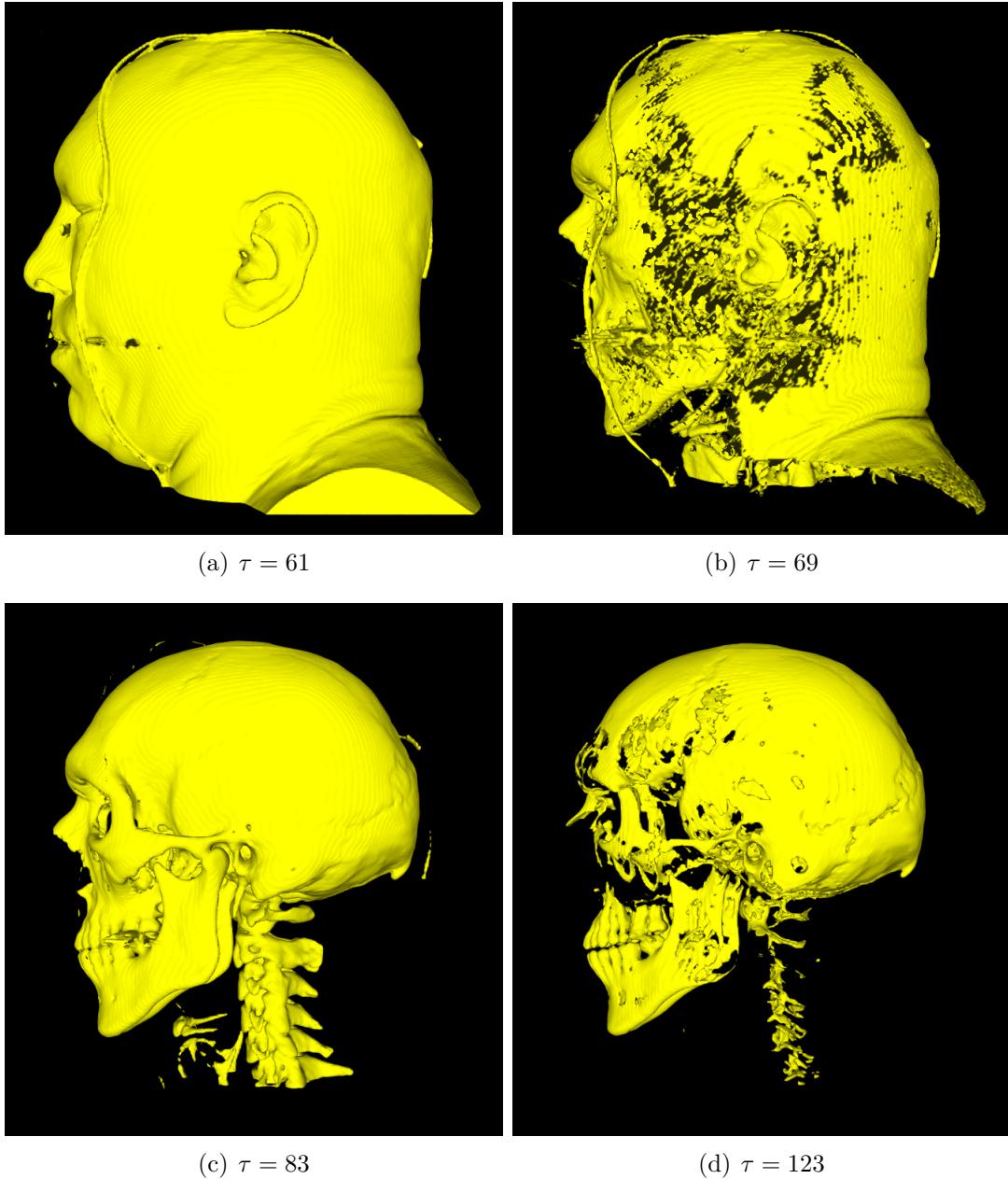
$$S_\tau = \{\mathbf{x} \mid f(\mathbf{x}) = \tau\}. \quad (1.5)$$

La superficie  $S_\tau$  es llamada *isosuperficie* o *frontera* definida por  $\tau$ ; el valor constante  $\tau$  es llamado *isovalor* o *umbral*. Debido a que tenemos la aproximación  $v$  de  $f$ , las técnicas por isosuperficie consisten en seleccionar un isovalor del conjunto de  $p_\Delta(\mathbf{k})$  (el rango de la imagen) y rastrear la isosuperficie definida por ese valor. Esto resulta en una especie de envolvente o cascara discreta que forma una malla poligonal. Para poder definir la malla es necesario tener los vértices de cada polígono y la conectividad (saber que par de vértices forman cada arista). Esta malla se puede visualizar por métodos tradicionales de GC. En la Figura 1.4 se muestra el mismo volumen de las Figuras 1.2 y 1.3 visualizado por cuatro isosuperficies correspondientes a diferentes isovalores en el rango  $[0, 255]$  y con el método de *Marching Cubes*, explicado mas adelante.

Hay varias razones por las que la visualización por superficies es más usada. Por ejemplo



**Figura 1.3:** El mismo conjunto de datos de la Figura 1.2 visualizado por Volume Rendering. En esta imagen se usa el método de suma de intensidades como función de transferencia.



**Figura 1.4:** Visualización por superficies de la imagen 3D de la Figura 1.2 con cuatro isovalores dentro del rango [0, 255] producida con el método de *Marching Cubes*.

la rapidez del cómputo y el hecho de que tanto el hardware como el software que usamos esta optimizado para visualizar estas superficies. Sin embargo, lo mas importante es que los humanos estamos acostumbrados a ver las cosas de esa manera, generalmente solo percibimos la capa exterior de la mayoría de los objetos.

## 1.1 Generación de mallas

Hay varios algoritmos que pueden rastrear la frontera y construir la malla que la aproxima. El algoritmo de *Marching Triangles*, propuesto por Hilton, Stoddart, Illingworth y Windeatt en [27], produce una malla a partir de una nube de puntos. En este algoritmo se empieza con un triangulo y se empiezan a añadir vértices que forman triángulos con las aristas que ya pertenecen a la malla, cuidando de cumplir que la triangulación que forma la malla sea siempre una triangulación de Delaunay. Este método tiene la enorme ventaja de producir mallas estables y de buena calidad. La desventaja de este algoritmo es que la malla resultante no suele adaptarse bien a las secciones de la superficie donde hay muchos detalles finos. En [51] se propone una mejora, capaz de producir mallas adaptables y además es capaz de encontrar los diferentes componentes conexos del volumen.

Otro algoritmo que fue propuesto por Keppel [31] reconstruye la superficie a través de secciones transversales del volumen. En un primer paso en cada una de las secciones transversales extraen un contorno por medio de algún algoritmo de detección de bordes. En un segundo paso los contornos de secciones adyacentes se empiezan a unir formando polígonos. El paso crucial de este algoritmo es conectar de manera correcta los diferentes contornos. Sin embargo, debido a que los contornos que deseamos reconstruir en medicina son muy complejos no se logra hacer la conexión de manera aceptable. Por esto último se han escrito algunas mejoras (ver por ejemplo [11]), pero debido a que la mayoría se basan en heurísticas estos métodos son poco usados en la actualidad [23].

En éste trabajo estamos interesados en dos métodos: el método de *Marching Cubes* (MC)[34] que es el estándar mas usado actualmente y el Algoritmo de Artzy[2].

### 1.1.1 Adyacencias y caminos

Antes de explicar como funcionan estos algoritmos necesitamos definir los conceptos de adyacencia y camino entre véxeles

El concepto de *adyacencia* entre dos véxeles es una relación binaria simétrica sobre  $\mathbb{Z}^3$ . Se dice que dos véxeles son vecinos cuando son adyacentes. Existen varias adyacencias, aquí solo vamos a utilizar dos de las más comunes.

Se dice que dos véxeles  $\mathbf{c}$  y  $\mathbf{d}$  en la rejilla  $sc$  son *adyacentes cara a cara* o  $\omega$ -adyacentes, si difieren en sólo una de sus coordenadas, ver Figura 1.5(a). Es decir:

$$(\mathbf{c}, \mathbf{d}) \in \omega \leftrightarrow |\mathbf{c} - \mathbf{d}| = 1, \quad (1.6)$$

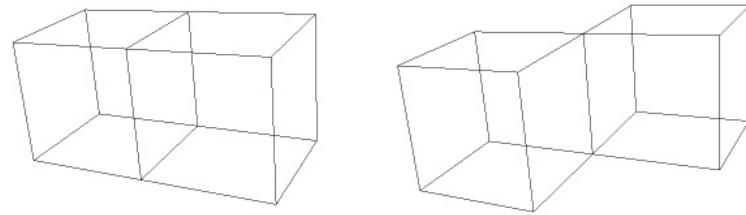
en donde  $|\mathbf{p}|$  denota la norma euclídea del vector  $\mathbf{p}$ . Un véxel en la rejilla puede ser adyacente bajo  $\omega$  con otros seis véxeles de la rejilla, es por eso que a esta adyacencia se le conoce también como la 6-vecindad del véxel, ver la Figura 1.6(a).

Hay otra adyacencia llamada *adyacencia arista a arista* o  $\delta$ -adyacencia. En ésta adyacencia los véxeles son vecinos si comparten una cara o comparten una arista, ver Figura 1.5(b). Esta adyacencia se define de la siguiente manera:

$$(\mathbf{c}, \mathbf{d}) \in \delta \leftrightarrow 1 \leq |\mathbf{c} - \mathbf{d}| \leq \sqrt{2}. \quad (1.7)$$

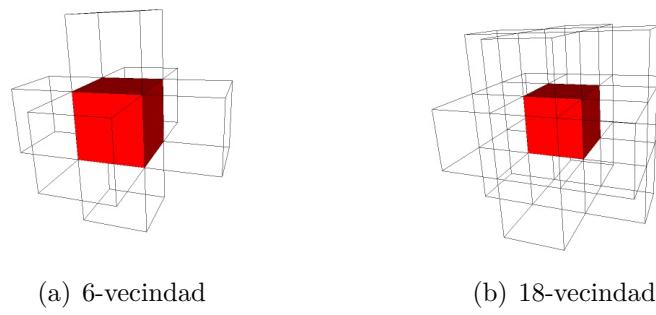
Esta adyacencia también se le llama la 18-vecindad porque el véxel tiene 18 vecinos, ver Figura 1.6(b). Se hace énfasis en que estas adyacencias son inclusivas. La  $\delta$ -adyacencia incluye a la  $\omega$ -adyacencia.

Un  $\pi$ -camino de  $\mathbf{c}$  a  $\mathbf{d}$  se define como la secuencia de véxeles  $\langle \mathbf{c}^0, \mathbf{c}^1, \dots, \mathbf{c}^n \rangle$  tales que  $\mathbf{c}^0 = \mathbf{c}$ ,  $\mathbf{c}^n = \mathbf{d}$  y para todo  $1 \leq i \leq n$   $\mathbf{c}^{k-1}$  es  $\pi$ -adyacente a  $\mathbf{c}_k$ . Un conjunto  $A$  de véxeles es  $\pi$ -conexo si entre cada par de véxeles de  $A$  hay un  $\pi$ -camino. Por lo tanto se pueden tener elementos  $\omega$ -conexos a través de un  $\omega$ -camino y similarmente para la  $\delta$ -adyacencia.



(a) Vóxeles  $\omega$  y  $\delta$  adyacentes      (b) Vóxeles  $\omega$  adyacentes.

**Figura 1.5:** Dos tipos de adyacencias típicas para vóxeles cúbicos.



(a) 6-vecindad      (b) 18-vecindad

**Figura 1.6:** Diferentes vecindades de un vóxel cubico simple.

### 1.1.2 El algoritmo de *Marching Cubes* de Lorensen y Cline

Este algoritmo fue propuesto inicialmente por Lorensen y Cline en [34]. Al ser tan popular se ha escrito bastante acerca de sus propiedades, de sus puntos débiles y de sus posibles mejoras. Un resumen de todos estos trabajos puede verse en [38].

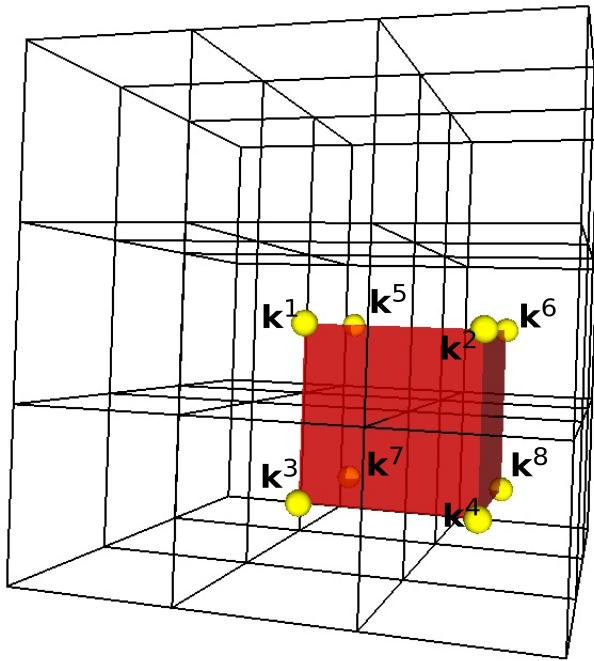
La entrada del algoritmo es un volumen  $v$  junto con un umbral  $\tau$ . La salida es un conjunto de triángulos  $\mathcal{M}_\tau$  que aproxima la superficie  $S_\tau$  de (1.5). El algoritmo consiste en hacer recorrer un cubo por el volumen, en cada parada se van generando los triángulos que forman  $\mathcal{M}_\tau$ , de ahí su nombre.

El algoritmo utiliza un cubo formado por los centros de ocho vértices contiguos de tal forma que para cada vértice  $\mathbf{k} \in \Gamma$ , el cubo tiene por vértices los centros de vértices  $C(\mathbf{k}) = \{\mathbf{k}^i | k^i \in \mathbb{Z}, \mathbf{k}^1 = \mathbf{k} \text{ y } 1 \leq i \leq 8\}$  en donde:  $\mathbf{k}^2 = (k_1^1 + 1, k_2^1, k_3^1)$ ,  $\mathbf{k}^3 = (k_1^1, k_2^1 - 1, k_3^1)$ ,  $\mathbf{k}^4 = (k_1^1 + 1, k_2^1 - 1, k_3^1)$ ,  $\mathbf{k}^5 = (k_1^1, k_2^1, k_3^1 + 1)$ ,  $\mathbf{k}^6 = (k_1^1 + 1, k_2^1, k_3^1 + 1)$ ,  $\mathbf{k}^7 = (k_1^1, k_2^1 - 1, k_3^1 + 1)$  y  $\mathbf{k}^8 = (k_1^1 + 1, k_2^1 - 1, k_3^1 + 1)$ , ver la Figura 1.7.

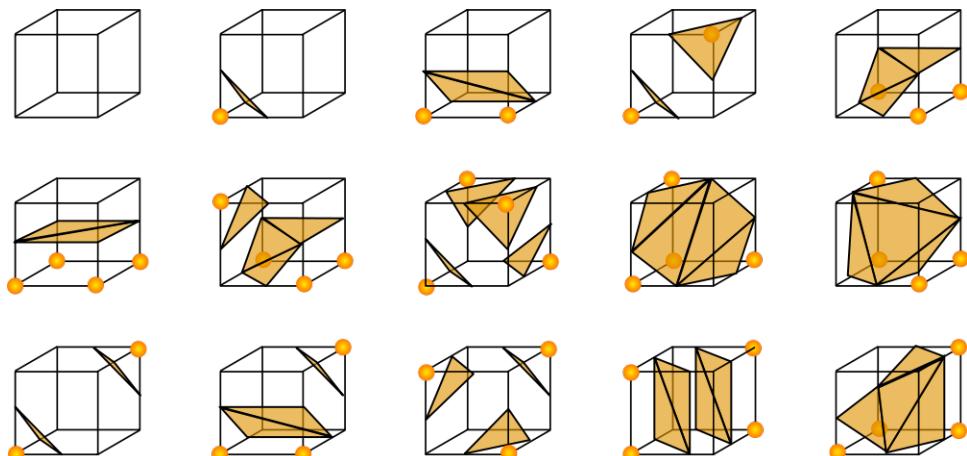
Para cada voxel  $\mathbf{k} \in \Gamma$  se revisan los valores de cada  $\mathbf{a} \in C(\mathbf{k})$  que conforman el cubo, si el valor del voxel  $v(\mathbf{a})$  es mayor o igual que el isovalor  $\tau$ , marca al voxel como *interior*, si el valor  $v(\mathbf{a})$  es menor que el isovalor lo marca como *exterior*. Como en cada parada el cubo toca ocho vértices y cada uno de puede ser dentro o fuera, hay  $2^8$  posibles casos. Si todos los vértices son interiores, el cubo está dentro de la superficie y por lo tanto no hay nada que hacer. Si el valor de todos los vértices es exterior, el cubo está fuera de la superficie y tampoco hay que hacer.

Los casos interesantes son cuando hay tanto vértices externos como vértices internos, porque eso significa que estamos en un cruce de la isosuperficie. Cuando estamos en uno de estos casos hay que generar triángulos que aproximen la superficie. En el algoritmo original, la configuración de los vértices de los triángulos a generar en cada caso se guardan en una tabla que es construida manualmente.

El MC estándar considera que algunos casos son equivalentes de acuerdo a rotaciones y complementos. Un caso es complemento de otro si todos los vértices del primero tienen el marcado contrario en el segundo. Un caso es una rotación de otro si hay un conjunto de rotaciones que transforman al primer caso en el segundo. Por lo que se pueden clasificar las 256 configuraciones posibles en solo 15 casos. En la Figura 1.8 se ven todos los casos a los que se reducen las configuraciones una vez que se explotan los complementos y las rotaciones.



**Figura 1.7:** El algoritmo de MC utiliza un cubo (ilustrado en rojo) formado por los centros de ocho vértices contiguos ilustrados por las esferas  $\mathbf{k}^i$  con  $1 \leq i \leq 8$ . Este cubo se va recorriendo a lo largo de todo el volumen  $v$ .



**Figura 1.8:** Los diferentes casos de acuerdo a las configuraciones en MC. Los puntos amarillos se consideran interiores y donde no hay marca se consideran exteriores.

Para cada par  $(\mathbf{a}, \mathbf{b})$  donde  $\mathbf{a}, \mathbf{b} \in C(\mathbf{k})$  tales que  $v(\mathbf{a}) > \tau$  y  $v(\mathbf{b}) \leq \tau$  y  $(\mathbf{a}, \mathbf{b}) \in \omega$  se considera que el cruce de la superficie se encuentra en algún punto entre  $\mathbf{a}$  y  $\mathbf{b}$ . El punto de cruce  $\mathbf{p}$  se puede hallar por interpolación lineal (aunque hay otros métodos [38]) de la siguiente forma:

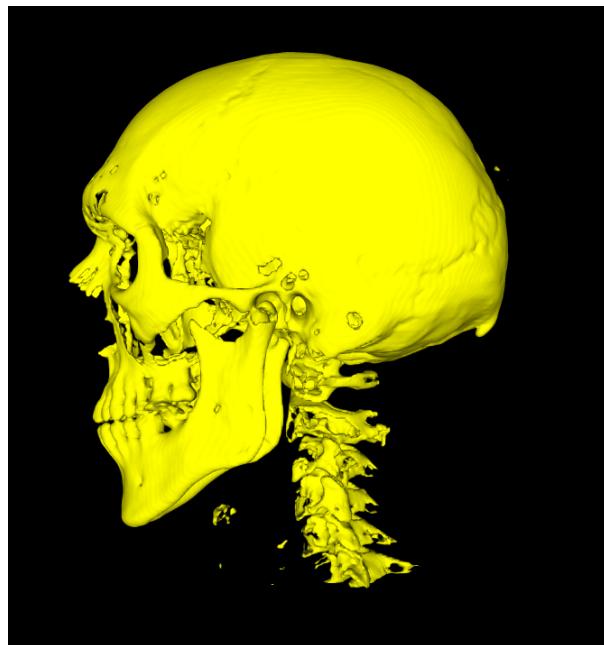
$$\mathbf{p} = \mathbf{a} + \rho(\mathbf{b} - \mathbf{a}), \quad (1.8)$$

en donde

$$\rho = \frac{\tau - v(\mathbf{a})}{v(\mathbf{b}) - v(\mathbf{a})}. \quad (1.9)$$

Una vez que se conocen todos los puntos de intersección en las aristas del cubo, se construyen los triángulos de las superficies de acuerdo al caso en el que se encuentre el cubo, ver Figura 1.8.

La colección de todas las caras triangulares  $\mathcal{M}_\tau$ , después de que el cubo ha recorrido todo el volumen, es la aproximación a la superficie  $S_\tau$ . Este algoritmo tiene la ventaja de que el hardware y el software para hacer gráficas por computadora es muy eficiente al dibujar triángulos. Además de que se puede saber fácilmente la normal a la superficie en cualquier punto de un triángulo (por ejemplo, calculando el producto vectorial de los segmentos que forman dos de sus lados).



**Figura 1.9:** Superficie del mismo conjunto de datos de la Figura 1.2 obtenida por Marching Cubes, para un isovalor  $\tau = 100$ .

### 1.1.3 El Algoritmo de seguimiento de superficies de Artzy

Este fue uno de los primeros algoritmos que permite encontrar una aproximación a (1.5). Fue propuesto por Ehud Artzy y por Gabor T. Herman en [2]. En su libro [26] de posterior publicación, Herman usa el nombre de *Algoritmo de Artzy* para referirse a este método.

Para usar este algoritmo necesitamos también un volumen  $v$  junto con un umbral  $\tau$ . Su salida es también un conjunto de caras  $\mathcal{A}_\tau$  que aproxima  $S_\tau$ , en este caso rectangulares y ortogonales a los ejes que definen  $v$ . A diferencia de MC este algoritmo necesita una entrada mas, una cara del conjunto  $\mathcal{A}_\tau$  que sirve de posición de inicio (a veces llamada semilla).

#### 1.1.3.1 Condiciones del Algoritmo de Artzy

El Algoritmo de Artzy empieza suponiendo que los véxeles del volumen están de alguna manera clasificados en dos conjuntos: interior y exterior (conocido como un volumen binarizado). El criterio con el que se hace esta clasificación puede ser arbitrario. Por esto, el Algoritmo de Artzy puede usarse en conjunto con cualquier método de segmentación.

Dado que el propósito de este trabajo no es la segmentación (operación implícita en MC) y se desea ocupar el Algoritmo de Artzy como comparación con el algoritmo de MC, consideraremos todos los véxeles  $\mathbf{k} \in \tau$  tales que  $v(\mathbf{k}) < \tau$  como exteriores y los véxeles para los cuales  $v(\mathbf{k}) \geq \tau$  como interiores, para un cierto isovalor  $\tau$ . Este método de segmentación es conocido como umbralización.

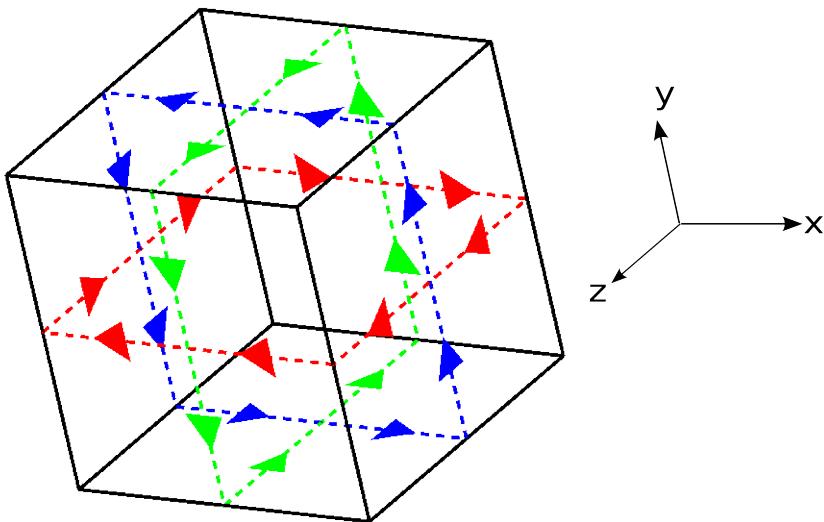
Definimos tambien un elementos de la frontera o *bel*, el término *bel* viene del ingles *boundary element*, como una cara común entre dos véxeles  $\omega$ -adyacentes tales que uno de ellos es interior y el otro es exterior. Es fácil ver que para nuestros véxeles los *bels* serán caras rectangulares.

El Algoritmo de Artzy encuentra un conjunto  $\mathcal{A}_\tau$  de caras de véxeles tales que cada cara en  $\mathcal{A}_\tau$  es un *bel*. El conjunto  $\mathcal{A}_\tau$  es la aproximación poligonal de la isosuperficie (1.5). En términos mas concretos si  $O$  es un conjunto de véxeles interiores y  $Q$  es un conjunto de véxeles exteriores, definimos la frontera  $\partial$  entre  $O$  y  $Q$  como:

$$\partial(O, Q) = \{(\mathbf{c}, \mathbf{d}) | \mathbf{c} \in O, \mathbf{d} \in Q, \text{ y } (\mathbf{c}, \mathbf{d}) \in \omega\}. \quad (1.10)$$

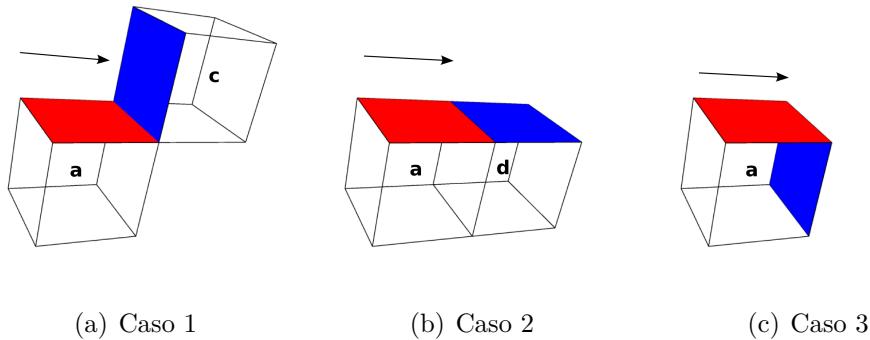
El Algoritmo de Artzy necesita un punto de inicio o semilla  $\beta^0$ . El punto de inicio es uno de los  $bels \in \partial(O, Q)$  y de ahí procede de manera iterativa hasta encontrar todos los *bels* que aproximan la frontera  $\partial(O, Q)$ .

Para hacer este rastreo el algoritmo simplemente visita cada cara que sea un *bel* de acuerdo a las rutas validas que se muestran en la Figura 1.10. La idea es que cada que se llega a una cara hay dos maneras de continuar de acuerdo a la orientación de la cara. Como las caras pertenecen a vóxeles cúbicos solo hay seis posibles orientaciones.



**Figura 1.10:** En el Algoritmo de Artzy para cada cara hay dos formas validas de llegar y dos formas validas de salir.

Cada vez que paramos en un *bel*  $\beta$  se toman las dos posibles direcciones válidas de salida y por cada una se busca una cara  $\beta^i \in \partial$ . Hay tres posibles casos ilustrados en la Figura 1.11. Supongamos que  $\beta$  separa dos vóxeles  $\mathbf{a} \in O$  y  $\mathbf{b} \in Q$  y sean  $\mathbf{c}$  y  $\mathbf{d}$  los dos vóxeles tales que  $(\mathbf{b}, \mathbf{c}) \in \omega$ ,  $(\mathbf{a}, \mathbf{c}) \in \delta$ ,  $(\mathbf{b}, \mathbf{d}) \in \delta$ ,  $(\mathbf{a}, \mathbf{d}) \in \omega$  y que estan en la dirección valida a partir de  $\beta$ . Si  $\mathbf{c} \in O$  estamos en el caso ilustrado en la Figura 1.11(a), si por el contrario  $\mathbf{c} \in Q$  debemos revisar en  $\mathbf{d}$ . Si  $\mathbf{d} \in O$  estamos en el caso de la Figura 1.11(b) y si  $\mathbf{d} \in Q$  en el caso de la Figura 1.11(c). En cualquiera de los casos anteriores siempre encontramos una cara  $\beta^i \in \partial$ .



**Figura 1.11:** Si se está en la cara roja  $\beta$  con la dirección válida apuntada por la flecha, hay tres casos posibles para encontrar la cara azul  $\beta^i$ . Por simplicidad en la figura solo se dibujan los sólidos en  $O$ .

### 1.1.3.2 Implementación del algoritmo

En [20] se realizó una implementación eficiente del Algoritmo de Artzy por medio de tres estructuras de datos:

- Una lista auxiliar  $L$  de caras  $\beta$  que nos permita buscar en ella de manera eficiente. En la implementación se sugiere un árbol binario de búsqueda.
- Una cola  $Q$  para tener un estado de espera donde guardar las rutas de salida válidas que aún no se han recorrido. La lista  $Q$  es una manera de simular el paralelismo que se define matemáticamente en [26]. Debido a que se requiere que se exploren un número potencialmente grande de rutas al mismo tiempo este tipo de paralelismo no puede ser alcanzado en la implementación y debe simularse con una estructura de datos.
- La lista de *bels*  $\mathcal{A}_\tau$  que aproxima la superficie  $S_\tau$ .

El algoritmo se lista a continuación. Se asume que el volumen  $v$ , ha sido clasificado por algún criterio de segmentación para producir el volumen binarizado  $B(v)$  y que de alguna manera se encontró un cara  $\beta^0$  como semilla. En nuestra implementación se recorre  $v$  voxel por voxel hasta que se encuentra la primera cara  $\beta^0$  que se encuentra en la frontera definida en (1.10).

---

**Algoritmo 1** Algoritmo de Artzy

---

**Entradas:** Un *bel* inicial  $\beta^0$  y el volumen binarizado  $B(v)$ .

**Salidas:** Una lista  $\mathcal{A}_\tau$  con todas las caras cuadradas que aproximan la superficie.

```

1: Colocar el bel  $\beta^0$  en la salida  $\mathcal{A}_\tau$ , agregar  $\beta^0$  a  $Q$  y poner dos copias de  $\beta^0$  en  $L$ .
2: while  $Q \neq \emptyset$  do
3:   Sacar una cara  $\beta$  de  $Q$ 
4:   Encontrar las caras  $\beta^1$  y  $\beta^2$  en  $B(v)$  a las que se puede ir desde  $\beta$ , siguiendo las
   dos direcciones de la Figura 1.10, de acuerdo a alguno de los casos mostrados en
   la Figura 1.11.
5:   if  $\beta^1 \in L$  then
6:     Quitar a  $\beta^1$  de  $L$ 
7:   else
8:     Agregar a  $\beta^1$  en  $\mathcal{A}_\tau$ , en  $L$  y en  $Q$ 
9:   end if
10:  if  $\beta^2 \in L$  then
11:    Quitar a  $\beta^2$  de  $L$ 
12:  else
13:    Agregar a  $\beta^2$  en  $\mathcal{A}_\tau$ , en  $L$  y en  $Q$ 
14:  end if
15: end while

```

---

### 1.1.3.3 Propiedades del Algoritmo de Artzy

Se demuestra en [26] que el Algoritmo de Artzy es topológicamente correcto. Esto quiere decir de manera intuitiva que la frontera que produce es cerrada, conexa y que divide al espacio en dos. Es un análogo digital al Teorema de la Curva de Jordán [26].

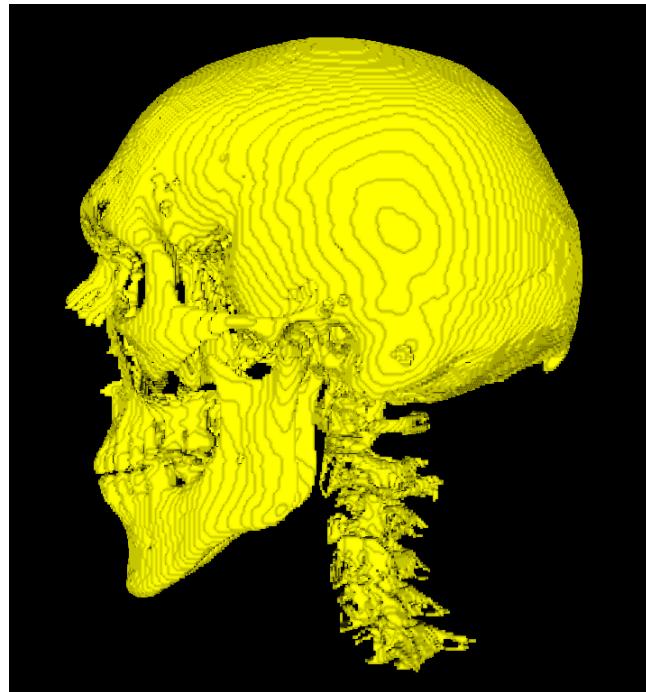
Supongamos que el volumen  $B(v)$  esta binarizado, supongamos también que conocemos una cara  $\beta_0$  tal que está entre dos véxoles  $\mathbf{g} \in O$ ,  $\mathbf{h} \in Q$  y  $(\mathbf{g}, \mathbf{h}) \in \omega$ . Sea también  $O$  el componente  $\delta$ -conexo que contiene a  $\mathbf{g}$  y  $Q$  el componente  $\omega$ -conexo que contiene  $\mathbf{h}$ . El Algoritmo de Artzy asegura que después de un número finito de pasos tendremos a  $\partial(O, Q) = \mathcal{A}_\tau$ .

Aun más, asegura que esta frontera  $\partial(O, Q)$  divide al espacio  $\mathbb{Z}^3$  en dos subconjuntos propios  $I$  y  $E$ , tales que:

1.  $O \subset I$  y  $Q \subset E$ .
2.  $\partial(O, Q) = \partial(I, E)$ .
3.  $I \cup E = \mathbb{Z}^3$  y  $I \cap E = \emptyset$ .
4.  $I$  es un subconjunto  $\delta$ -conexo de  $\mathbb{Z}^3$  y  $E$  es un subconjunto  $\omega$ -conexo de  $\mathbb{Z}^3$ .
5. Todo  $\omega$ -camino que vaya de un elemento de  $I$  a un elemento de  $E$  cruza  $\partial(O, Q)$ .

La propiedad de mas importancia para nosotros es la última, pues nos asegura que a diferencia de MC, en el Algoritmo de Artzy nuestra aproximación poligonal a la isosuperficie  $\mathcal{A}_\tau$  *nunca* tiene agujeros.

La Figura 1.12 presenta una superficie obtenida por el Algoritmo de Artzy, para el mismo conjunto de datos y con el mismo isovalor que el de la Figura 1.9.



**Figura 1.12:** Superficie obtenida por el Algoritmo de Artzy sobre el mismo conjunto de datos que en la Figura 1.9 para un isovalor  $\tau = 100$ .

## 1.2 Visualización de mallas

Como mencionamos antes, una vez obtenida la aproximación a  $S_\tau$  se utilizan técnicas de GC para producir una imagen 2D a partir del modelo, tratando de preservar la apariencia tridimensional.

El proceso por el que pasa la malla hasta producir la imagen final se conoce como *render pipeline*. Durante este proceso se hacen operaciones que afectaran la apariencia de la imagen final. La técnicas que ayudan a conservar la apariencia tridimensional se les llama pistas de profundidad. Algunas de estas técnicas son las siguientes:

- La *proyección en perspectiva*. La proyección es la operación que proyecta el modelo 3D a un plano. Generalmente puede ser ortogonal o en perspectiva. En la proyección ortogonal los objetos no alteran su tamaño en función de la distancia al espectador. En contraste, en la proyección en perspectiva los objetos cambian su tamaño de acuerdo a la distancia a la que estén del espectador lo que da una apariencia mas cercana a la realidad.
- La *eliminación de superficies ocultas*, también conocida en GC como *z-buffering* o *depth buffering* se encarga de eliminar las partes de objetos que se encuentran detrás de otros objetos en relación con el espectador. El efecto final consiste en que partes de algunos objetos de la escena no son visibles desde algunos puntos de vista.
- La *iluminación* y el *sombreado* son técnicas de GC que afectan en gran medida la imagen final. Consisten en modelar la interacción de la luz con los materiales de los que están hechos los objetos y por lo tanto dar una apariencia mas realista a la imagen final.
- El *mapeo de texturas* y el *mapeo de relieves* son técnicas que consisten en recalcular la apariencia de los píxeles finales en la imagen 2D con información guardada en un mapa y pueden afectar en gran medida la imagen final.

### 1.2.1 Iluminación

En graficación por computadora hay dos formas para poner color a una escena, una es simplemente asignándoles un color a los píxeles que están dentro de los objetos que es

equivalente a decir que coloreamos los objetos. La otra consiste en calcular el color de cada píxel con base a las condiciones de iluminación en la escena.

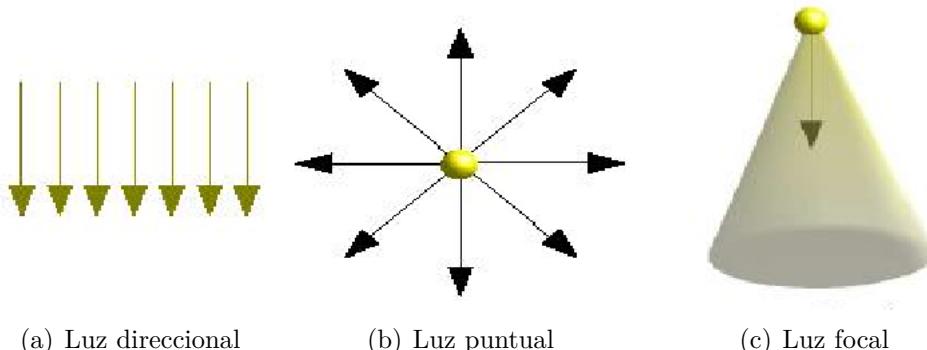
La segunda de estas técnicas se conoce como iluminación y es visiblemente más agradable y realista que el coloreado. Para poder usar la iluminación es necesario definir fuentes de luz en la escena, luego definir las propiedades de reflectividad de los objetos, y también un vector normal unitario  $\vec{n}$  a la superficie de los objetos.

La información de  $\vec{n}$  se usa para dos cosas. La primera de ellas es determinar cual es la cara frontal del polígono, pues algunos dispositivos de despliegue en un intento por hacer el proceso mas rápido no dibujan la cara de atrás de los polígonos (esto se llama *culling*). El segundo uso de  $\vec{n}$  es para hacer cálculos de contribución de las luces en la escena a la iluminación del objeto.

El proceso de iluminación se lleva a cabo en dos partes, una donde se calcula la influencia de las luces en cada vértice del modelo de acuerdo con el modelo de iluminación (*lighting model*) y otra donde se calcula el color de los píxeles que corresponden a los puntos en la superficie de cada polígono con base a un modelo de sombreado (*shading model*).

#### 1.2.1.1 Tipos de fuentes de luz

Generalmente hay tres tipos de fuentes luz posibles que afectan la escena: las luces puntuales, las luces focales y las luces direccionales. Los diferentes tipos de fuente de luz están representados en la Figura 1.13.



**Figura 1.13:** Tipos de fuentes de luz en una escena. Esta imagen fue tomada de [47].

**Luz direccional:** En este caso se asume que la fuente de luz está en el punto al infinito, por lo que a pesar de conocer la dirección en donde esta la fuente, ésta se encuentra tan lejos de la escena que los rayos de luz llegan de manera paralela.

**Luz puntual:** Se asume que la fuente de luz esta en un punto cercano e irradia sus rayos sobre la escena de manera igual en todas direcciones.

**Luz focal:** Éste es un caso particular del anterior, también sabemos la posición de la fuente de luz, pero ésta irradia sus rayos en una dirección preferente formando un cono de luz.

## 1.2.2 Modelos de iluminación

El comportamiento físico de la luz es un fenómeno muy difícil de modelar. Por lo tanto, en las gráficas por computadora se usan modelos que aproximan este comportamiento de manera que los cálculos sean posibles en un tiempo razonable. Por esta razón se dice que los modelos de iluminación son *inspirados en la física*, pero no son *físicamente correctos* [9].

### 1.2.2.1 Modelo de Phong

El modelo de iluminación de Phong es el mas usado actualmente, fue propuesto por Bui Tuong Phong en su tesis de doctorado [41]. Debido a su simpleza y los buenos resultados, se ha implementado eficientemente tanto en *hardware* como en *software*.

El modelo de Phong es en esencia un modelo de como se refleja la luz sobre los objetos en la escena. En este modelo se asume que el reflejo de la luz puede verse como la superposición de tres componentes independientes: la componente ambiental, la componente difusa y la componente especular. A su vez los objetos están hechos de cierto material. Cada material tiene una cierta sensibilidad a cada componente de la luz. De esta manera, cada material tiene coeficientes de reflexión ambiental, difuso y especular distintos. El color final de un píxel en el objeto en la escena es función del material y de las propiedades de la luces que interactúen con él.

Adicionalmente se asume que el modelo de percepción humana del color donde se percibe por medio de la superposición de tres longitudes de onda, llamados canales y que los

cálculos de iluminación se pueden hacer en cada uno de los canales de manera independiente. Estos canales comúnmente representan luz en longitud de onda cercana al rojo, al verde y al azul. Por eso, la intensidad de la luz es representada como  $\mathbf{l} = (l_r, l_g, l_b)$  donde  $l_r$ ,  $l_g$  y  $l_b$  representan la longitud de onda cercana al rojo, al verde y al azul, respectivamente.

### 1.2.2.2 La reflexión ambiental

La reflexión ambiental es el color del objeto debido a los rayos de luz que provenían de una fuente, pero que han rebotado tantas veces en objetos de la escena que es imposible saber ni su dirección ni su origen. Se puede pensar que están flotando en la escena y llegan al objeto desde todas direcciones. Aun cuando en esta reflexión la luz parece estar en todas partes, desaparece si se apaga la fuente de donde proviene.

La reflexión ambiental  $\mathbf{r}_a$  se calcula en función de dos cosas. La luz que tiene un cierto componente ambiental  $\mathbf{l}_a$  y el objeto que reacciona con los rayos de luz de acuerdo a un coeficiente  $0 \leq \rho_a \leq 1$ . En el modelo de Phong se calcula como:

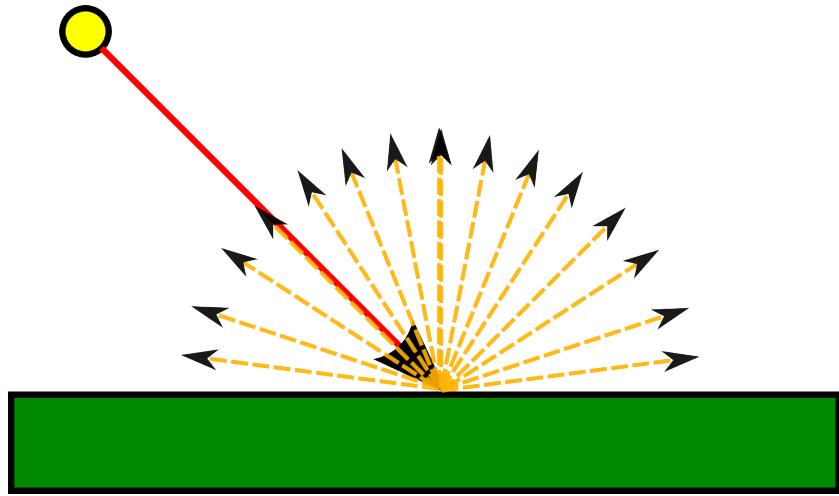
$$\mathbf{r}_a = \rho_a \mathbf{l}_a. \quad (1.11)$$

### 1.2.2.3 La reflexión difusa

En esta reflexión los rayos de luz llegan al objeto provenientes de una fuente y al llegar son reflejados por la superficie del objeto en todas las direcciones posibles con la misma intensidad en cada dirección. Esta reflexión intenta modelar el comportamiento de la luz sobre superficies opacas. Véase la Figura 1.14 en donde la reflexión difusa se representa con flechas color naranja.

La reflexión difusa es función de cuatro cosas. Primero las propiedades de la fuente de luz  $\mathbf{l}_d$ . Luego la posición relativa de la fuente de luz con respecto al punto donde estemos calculando la iluminación. Tercero, el coeficiente de reflexión difusa  $0 \leq \rho_d \leq 1$  específico al material del objeto. Por último el vector normal  $\vec{\mathbf{n}}$  a la superficie del objeto en el punto donde estemos calculando la iluminación.

El modelo de Phong modela la reflexión difusa de la siguiente manera:



**Figura 1.14:** El rayo de luz rojo incide en la superficie y es reflejado en todas las direcciones posibles.

$$\mathbf{r}_d = \rho_d \mathbf{l}_d (\mathbf{c} \cdot \vec{\mathbf{n}}), \quad (1.12)$$

en donde  $\mathbf{c} = \mathbf{l} - \mathbf{p}$  es un vector que va del punto donde estamos calculando la iluminación a la fuente de luz, ver la Figura 1.16.

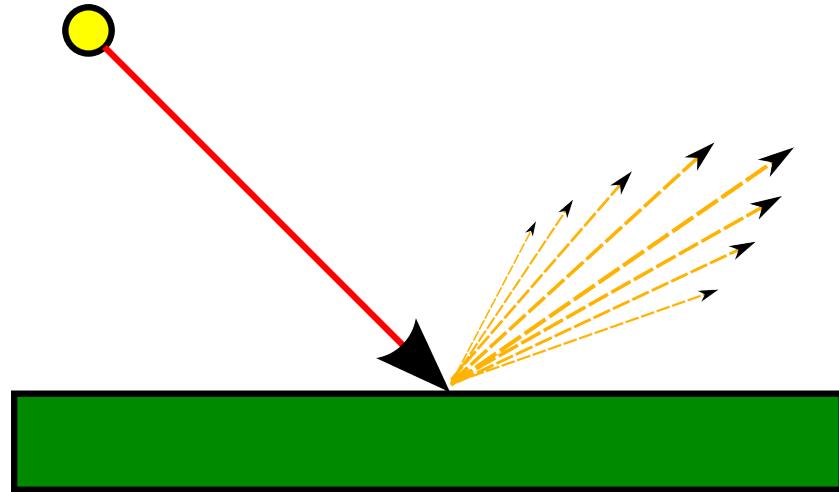
#### 1.2.2.4 La reflexión especular

Aquí los rayos de luz llegan al objeto y son reflejados por la superficie de éste en una dirección preferente, ver Figura 1.15. Esta reflexión intenta modelar la interacción de la luz con superficies muy brillantes. El espectador sólo percibe esta reflexión si está en algún lugar a donde lleguen los rayos reflejados y mientras más se acerca al reflejo perfecto más intensa es para él esta reflexión.

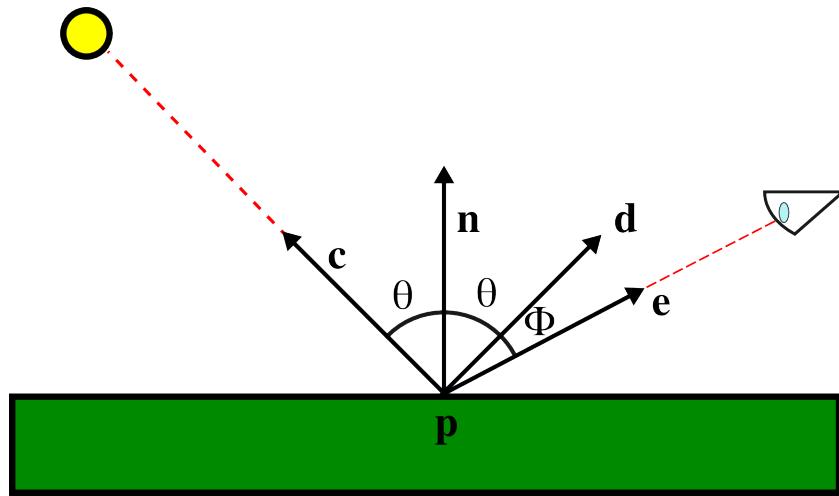
La reflexión especular en este modelo se calcula con la siguiente fórmula

$$\mathbf{r}_s = \rho_s \mathbf{l}_s (\mathbf{d} \cdot \mathbf{e})^\gamma, \quad (1.13)$$

en donde  $\mathbf{e}$  es un vector del punto que estamos calculando a la posición del observador. El vector  $\mathbf{d}$  va del punto donde estamos calculando a donde se haría el reflejo perfecto del rayo proveniente de la fuente de luz. El coeficiente  $0 \leq \rho_s \leq 1$  es una propiedad del material del objeto, al igual que el parámetro  $\gamma \geq 0$  que dice que tan brillante es el objeto. Todo puede verse en la Figura 1.16.



**Figura 1.15:** El rayo de luz rojo incide en la superficie y es reflejado en una dirección preferente.



**Figura 1.16:** La situación geométrica del modelo de Phong. El círculo amarillo es la fuente de luz y el ojo el observador.

El vector  $\mathbf{d}$  es aquel vector coplanar con  $\vec{\mathbf{n}}$  y con  $\mathbf{c}$  que hace el mismo ángulo  $\theta$  con  $\vec{\mathbf{n}}$  que  $\mathbf{c}$  y puede calcularse  $\mathbf{d} = 2(\mathbf{c} \cdot \vec{\mathbf{n}})\vec{\mathbf{n}} - \mathbf{c}$ . El ángulo  $\Phi$  entre los vectores  $\mathbf{d}$  y  $\mathbf{e}$  nos dice que tan cerca estamos del reflejo perfecto. Mientras el parámetro  $\gamma$  sea mas grande el abanico del reflejo especular es más angosto (Figura 1.15), valores típicos de  $\gamma$  van en el rango [50, 80].

### 1.2.2.5 Modelo completo

En el modelo de Phong se asume que los componentes de cada reflexión son independientes, por lo tanto el color final  $\mathbf{l}$  de cada píxel se puede obtener sumando las reflexiones de cada componente

$$\mathbf{l} = \mathbf{r}_a + \mathbf{r}_d + \mathbf{r}_s. \quad (1.14)$$

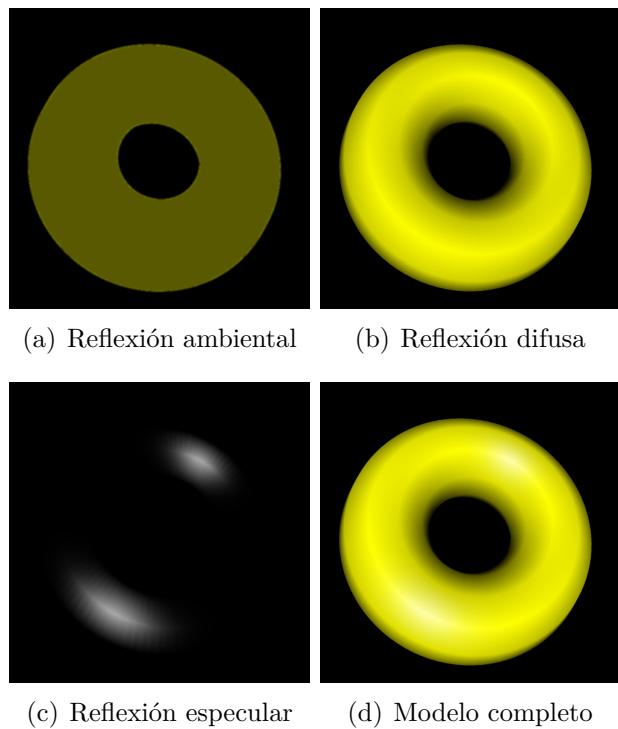
En la Figura 1.17 se pueden ver los efectos y contribuciones de este modelo. Donde primero se presentan los efectos de cada reflexión por separado y luego se presentan todos los componente juntos.

Hasta ahora los cálculos se han hecho bajo la suposición de que hay solo *una* luz afectando en la escena en caso de haber mas de una luz se suman las contribución de cada luz sobre el punto donde se este calculando la iluminación. También es común tener una componente ambiental  $\mathbf{r}_a^g = \rho_a \mathbf{l}^g$  que representa una contribución en el componente ambiental de la escena misma, es decir que no se debe a ninguna luz. Por lo que en una escena donde hay  $L$  luces el color del píxel es

$$\mathbf{l} = \sum_{i=1}^L (\mathbf{r}_a^i + \mathbf{r}_d^i + \mathbf{r}_s^i) + \mathbf{r}_a^g \quad (1.15)$$

Hay que tener cuidado al usar la ecuación anterior. Antes de sumar la contribución de cada luz debemos ver que ésta de verdad ilumine al punto donde estamos calculando la iluminación. Debemos verificar que  $\vec{\mathbf{n}} \cdot \mathbf{c}^i > 0$  en caso contrario simplemente saltamos esa contribución sin hacer ningún calculo. Físicamente si  $\vec{\mathbf{n}} \cdot \mathbf{c}^i \leq 0$  significa que la luz y el punto a iluminar están en lados opuestos de la superficie, como puede deducirse de la Figura 1.16.

Se hace énfasis de que en éste modelo las reflexiones que mas contribuyen al efecto final



**Figura 1.17:** Componentes del modelo de iluminación de Phong.

de la escena son la especular y la difusa (ver la Figura 1.17), las cuales son funciones del vector normal  $\vec{n}$ .

### 1.2.3 Modelos de sombreado

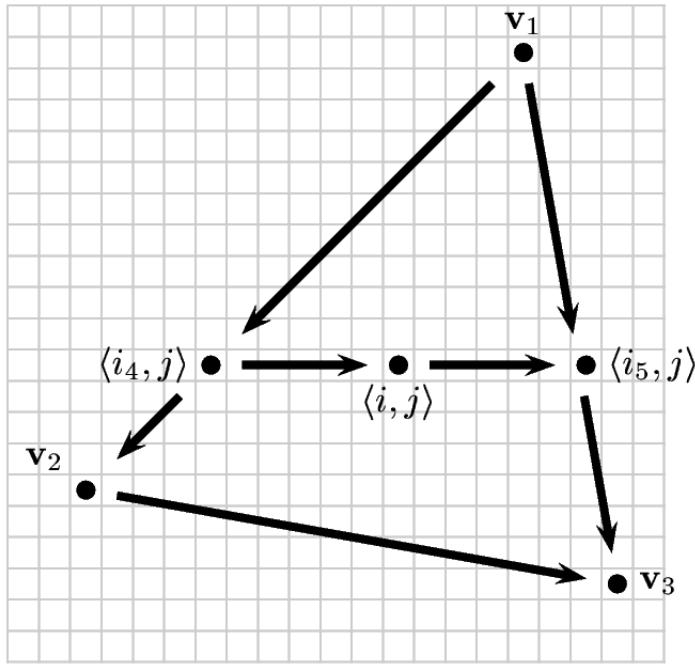
El modelo de iluminación se encarga de calcular el color de los vértices de la malla. Sin embargo, para hacer el despliegue esto no es suficiente, pues generalmente queremos desplegar los polígonos que forman la malla y necesitamos conocer el color en todos los puntos interiores del polígono no solo en su vértice. A la acción de calcular el color que deben tener los puntos interiores de un polígono en función de los colores de los vértices se le llama *sombreado* (en inglés el término es *shading*).

El sombreado se lleva a cabo en la última etapa del render pipeline llamada *rasterización* [43]. Esta etapa, a diferencia del resto del render pipeline es dependiente del *hardware* de despliegue. Para hacer la rasterización es necesario transformar todo el modelo de coordenadas en números reales a una discretización en coordenadas enteras que se corresponden con los píxeles del dispositivo de despliegue[4]. Debido a que el interior del polígono tiene puntos infinitos no podemos hacer el cálculo si no hasta saber cuáles de estos puntos se verán en la imagen final. Es decir, necesitamos saber cuáles de los puntos del polígono corresponderán a píxeles en el dispositivo de salida y solo se hacen los cálculos en estos puntos.

Los algoritmos de raster solo son capaces de actuar sobre triángulos debido a que necesitan asegurarse que el polígono a colorear es plano y convexo. En etapas anteriores del render pipeline los polígonos que forman el modelo son triangulados. De esta manera, aunque el modelo estuviera originalmente compuesto de polígonos complejos, al rasterizador solo llegan triángulos.

Para hacer su trabajo el rasterizador sabe las coordenadas y el color de los vértices de cada triángulo. Hace una correspondencia de cada vértice al píxel más cercano de manera que tiene la situación representada en la Figura 1.18.

El algoritmo de *raster* más común es conocido como *line scan*. El primer paso del algoritmo consiste en calcular qué píxeles forman las aristas del triángulo. Para calcular los píxeles que forman cada una de las aristas se hace uso del algoritmo de Bresenham [8]. Después el algoritmo calcula el color de los píxeles de cada arista usando el color de



**Figura 1.18:** El algoritmo de *line scan* calcula el color de los píxeles interiores del triángulo. Esta imagen fue tomada de [9].

los vértices. Por último el algoritmo empieza a barrer de manera horizontal todos los píxeles interiores del triángulo (de aquí su nombre) en cada píxel calcula el color entre los colores de los vértices que estén al inicio y al final del segmento horizontal sobre el que esta barriendo. En la Figura 1.18 se muestra el cálculo del color del píxel  $\langle i, j \rangle$  en función de los colores de los píxeles  $\langle i_4, j \rangle$  y  $\langle i_5, j \rangle$ .

### 1.2.3.1 Sombreado de Gouraud

Para que el algoritmo de *raster* funcione, necesita calcular los colores de los píxeles de un segmento de recta en función de los colores en los puntos extremos del segmento que lo delimitan. La manera como se hace este cálculo es dependiente del modelo de sombreado. El modelo de sombreado más usado es el de Gouraud, que fue propuesto en [22].

El sombreado de Gouraud es muy usado por su gran facilidad, actualmente está implementado tanto en *software* como en *hardware* en las tarjetas gráficas. Básicamente consiste en hacer una interpolación lineal de los colores de los píxeles en las aristas a partir del color de los vértices del triángulo. Luego, hacer interpolación lineal para

calcular el color de los píxeles interiores del triangulo a partir de los colores recién calculados en las aristas. Esto se conoce mas generalmente como *interpolación bilineal*.

Dicho de otra manera si en un segmento de recta los límites  $\mathbf{x}_0$  y  $\mathbf{x}_1$  tienen colores  $\langle r_0, g_0, b_0 \rangle$  y  $\langle r_1, g_1, b_1 \rangle$  y algún otro píxel está posicionado sobre el segmento de recta a una fracción  $\rho$  del camino que va de  $\mathbf{x}_0$  a  $\mathbf{x}_1$ , su color debería ser

$$(1 - \rho)\langle r_0, g_0, b_0 \rangle + \rho\langle r_1, g_1, b_1 \rangle. \quad (1.16)$$

Como ejemplo, haciendo referencia de nuevo a la Figura 1.18 primero interpolamos linealmente los colores del segmento que va de  $\mathbf{v}_1$  a  $\mathbf{v}_2$  (en algún momento de este paso calculamos el color en  $\langle i_4, j \rangle$ ), luego calculamos de la misma manera los colores del segmento de  $\mathbf{v}_1$  a  $\mathbf{v}_3$  (entre éstos al color de  $\langle i_5, j \rangle$ ) y finalmente de  $\mathbf{v}_2$  a  $\mathbf{v}_3$ . Luego procedemos a calcular los colores en el interior del triángulo, entre estos calculamos el color de  $\langle i, j \rangle$ .

Una de las desventajas importantes del modelo de Gouraud es que si un efecto luminoso, por ejemplo una reflexión especular, debería de afectar los píxeles interiores de un triangulo, pero por la situación geométrica no afecta ninguno de los vértices del mismo triangulo, entonces el sombreado de Gouraud pierde ese efecto luminoso.

### 1.2.3.2 Sombreado de Phong

En [41] no solo se propone un modelo de iluminación si no también se propone un modelo de sombreado para usarse en conjunto con el modelo de iluminación. Aunque este modelo fue diseñado para usarse en conjunto con el modelo de iluminación de Phong y proporciona mejores resultados que el modelo de Gouraud, en muchas aplicaciones se prefiere no usarlo debido a que requiere hacer cálculos considerablemente mas complicados.

En éste modelo se hace uso de la interpolación bilineal exactamente como la antes descrita, pero en vez de interpolar los colores, se interpolan las normales. Como las normales para la iluminación deben ser unitarias si existen dos píxeles  $\mathbf{x}_0$  y  $\mathbf{x}_1$  donde las normales son  $\mathbf{n}_0$  y  $\mathbf{n}_1$  respectivamente, en un píxel que está a fracción  $\rho$  de  $\mathbf{x}_0$  en el segmento de  $\mathbf{x}_0$  a  $\mathbf{x}_1$ , la normal interpolada es:

$$\frac{(1 - \rho)\mathbf{n}_0 + \rho\mathbf{n}_1}{\|(1 - \rho)\mathbf{n}_0 + \rho\mathbf{n}_1\|}. \quad (1.17)$$

Una vez que se conoce la normal, el color del píxel se calcula usando de nuevo el modelo de iluminación descrito en (1.15). Por esta razón, es necesario que información acerca de la posición y color de las luces, además del material del objeto se conserve en el pipeline hasta la etapa de rasterizado.

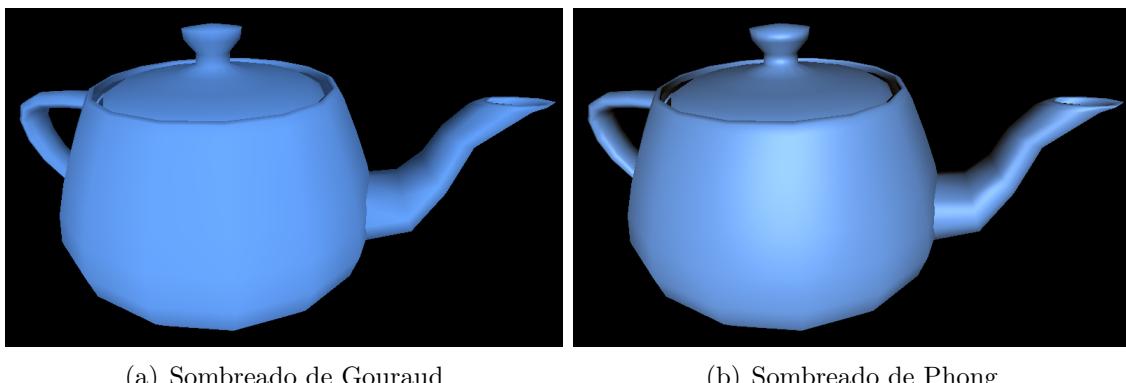
En éste modelo se corrige la desventaja de Gouraud explicada anteriormente. En contraste, una de las desventajas de usar el modelo de Phong son la mayor cantidad de cálculos y la necesidad de guardar mas información durante todo el render pipeline. Sin embargo, actualmente con las nuevas tarjetas de video esto no es un factor determinante.

Otra de las desventajas de interpolar linealmente las normales de una malla es la tasa de cambio de las normales. En polígonos que estén orientados frente al espectador las normales cambiaran mas lentamente que en áreas donde las normales estén apuntando hacia los lados del espectador.

Una manera de compensar este problema es usar las siguientes fórmulas para calcular las normales:

$$\begin{aligned} n_1(\rho) &= (1 - \rho)n_{1,0} + \rho n_{1,0}, \\ n_2(\rho) &= (1 - \rho)n_{2,0} + \rho n_{2,0}, \\ n_3(\rho) &= \sqrt{1 - n_{1,\rho}^2 - n_{2,\rho}^2}. \end{aligned} \quad (1.18)$$

Una comparación entre sombrear con el modelo de Phong y con el modelo de Gouraud se puede ver en la Figura 1.19.



**Figura 1.19:** El mismo modelo visto en condiciones geométricas iguales y con el mismo modelo de iluminación pero con dos modelos de sombreado distintos.

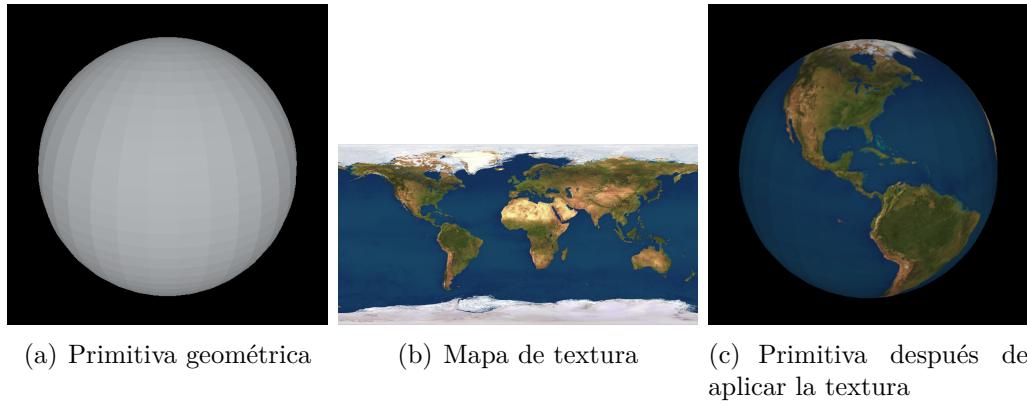
## 1.2.4 Efectos a partir de mapas para aumentar el realismo de la escena

### 1.2.4.1 Mapeo de texturas

Otra técnica de GC que sirve para aumentar realismo a la escena es el mapeo de texturas. La manera mas simple de pensar en el mapeo de texturas es imaginar una calcomanía que se pega a las primitivas geométricas. Por ejemplo podría tratarse de una esfera a la que quisiéramos aplicar un calcomanía de un mapa de la tierra para pensar en un modelo del planeta tierra. El poder representar la superficie de una esfera en el plano ha sido un problema al que se han enfrentado los cartógrafos desde hace muchos años. El mapeo de texturas es el mismo problema pero visto al contrario, queremos poder encontrar una correspondencia entre puntos en una superficie a un plano que representa la textura. De ahí el nombre de mapeo de texturas.

El mapeo de texturas ha avanzado mucho desde los inicio de las GC, por ejemplo ahora es común que las texturas no solo guarden información del color. En su forma más general la textura sirve como una *lookup table*, en la que se guarda alguna propiedad que se asigna a algún píxel de la primitiva por medio de una función. Las propiedades mas comunes que se guardan en la textura son el color, el brillo, los coeficientes de reflexión y las normales. Uno de los ejemplos mas claros es el *bump mapping* que será explicado en la siguiente sección.

El principal reto del mapeo de texturas es definir una función que haga el mapeo. Generalmente un mapa de textura toma las coordenadas  $u$  y  $v$  en el intervalo  $[0, 1]$ . Por lo tanto el reto es encontrar una manera de representar la primitiva geométrica como una superficie en coordenadas paramétricas, y luego encontrar un mapeo de los parámetros a coordenadas en el intervalo antes citado.



**Figura 1.20:** Ejemplo de una primitiva con y sin mapeo de texturas.

Por ejemplo en la Figura 1.20 se hace el mapeo de texturas sobre una esfera. La esfera primero se representó en coordenadas paramétricas de la forma  $\mathbf{p}(\theta, \phi) = \langle r \sin \theta \cos \phi, r \sin \theta \sin \phi, r \cos \theta \rangle$  en donde  $r$  es el radio de la esfera,  $\theta \in [0, 2\pi]$  y  $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$  y se ocuparon las siguientes formas para mapear la textura al intervalo  $[0, 1]$

$$\begin{aligned} s &= \frac{\theta}{2\pi}, \\ t &= \frac{\phi}{\pi} + \frac{1}{2}. \end{aligned} \tag{1.19}$$

En general, podemos tener problemas cuando no hay una correspondencia uno a uno entre los píxeles de la pantalla y los píxeles del mapa, en este caso pueden pasar dos cosas. Primero que la resolución de los píxeles del mapa sea mucho menor que la resolución de los píxeles de la pantalla, esto se traducirá en que un solo píxel del mapa puede corresponder a varios píxeles de la pantalla. Por lo tanto, veamos que en la pantalla hay regiones grandes de forma aproximadamente rectangular de un sólo color que están correspondiendo a un sólo píxel del mapa.

Segundo, si la resolución del mapa es mucho mayor que la de la pantalla. En un principio

esto parecería algo bueno, porque significa que el mapa tiene una resolución mayor que la que necesitamos. Sin embargo, esto también significa que un solo píxel de la pantalla corresponde a varios píxeles en el mapa. Esto implica que sólo algunos de los píxeles del mapa se están usando para producir la imagen final. Lo anterior puede derivar en que aparezcan algunos patrones no deseados en la imagen final. Una primera aproximación para resolver este problema es tratar de utilizar todos los píxeles al momento de hacer el cálculo del valor que debería proporcionarnos el mapa. Por ejemplo con una interpolación lineal o con un promedio entre alguna vecindad del píxel seleccionado.

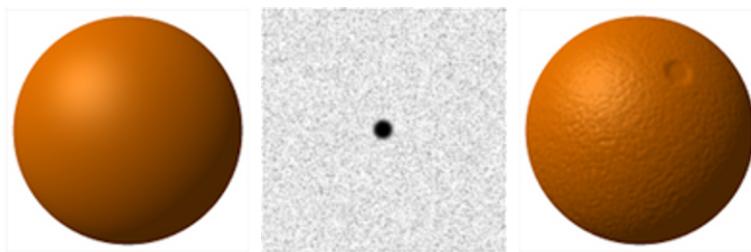
Quizás una de las soluciones más aceptadas es la propuesta por Williams en [49], donde propone la técnica conocida como *mipmapping* que consiste en construir a partir de un mapa original varios mapas de menor resolución hasta llegar a un mapa de un píxel de resolución. Para hacer el mapeo final entonces se calculan: el mapa que está más cerca pero es mayor en resolución y el mapa más cercano pero que es menor en resolución. Luego, en cada mapa se calcula un valor haciendo una interpolación lineal entre los píxeles más cercanos, lo que produce dos valores. Por último, el valor final se calcula con una interpolación lineal entre las resoluciones de los mapas y la escena. Williams mostró una manera eficiente de hacer este cálculo tanto en tiempo como en memoria. Para guardar los mapas de textura de varias resoluciones solo requiere de 33% más espacio que el mapa original. Este algoritmo está actualmente implementado en la mayoría de los procesadores de video (GPU).

Con la llegada de las GPUs programables y de los shaders, se potenció mucho el uso de texturas. Las GPUs permiten romper el pipeline gráfico y poder hacer el mapeo de texturas en varios lugares del pipeline y por lo tanto utilizarlo para varios efectos.

#### 1.2.4.2 Mapeo de relieves

Una forma de utilizar el mapeo de texturas para guardar información que afecta el modelo más allá de su color es el mapeo de relieves (el término en inglés es *bump mapping* y no existe una buena traducción del término al español, lo más cercano podría ser mapeo de relieves, o mapeo de bordes). En esta técnica propuesta por Blinn en [6], la idea es guardar un mapa de alturas con cuya información se perturban los vectores normales a la superficie del modelo. Una vez que se hacen los cálculos de iluminación esto da una apariencia rugosa al modelo.

Imaginemos que sobre el modelo se pega una calcomanía con relieve (o bordes) de tal forma que la superficie antes suave del modelo ahora tienen ciertas perturbaciones. Como hacer eso implicaría modificar la posición de los vértices de la superficie y sería muy costoso, se trata de simular el proceso. Con ayuda de la calcomanía, que ahora la llamamos mapa de alturas, podemos saber a dónde habría que mover los vértices del modelo para dar la apariencia deseada. En vez de mover los vértices calculamos el vector normal que tendrían los vértices perturbados y se lo asignamos a los vértices *sin modificar* su posición. En conjunto con el modelo de iluminación podemos dar la apariencia de que los vértices fueron perturbados sin alterar la geometría del modelo. Un ejemplo puede verse en la Figura 1.21.



(a) Modelo original (b) Mapa de alturas (c) Mapeo de relieves

**Figura 1.21:** Ejemplo de mapeo de relieves. Los bordes no están realmente ahí, son artefactos producidos por la iluminación en la escena.

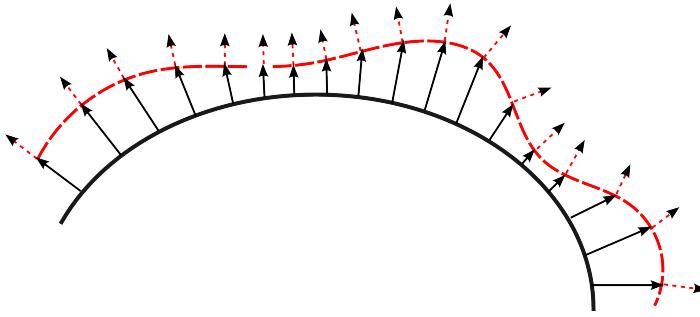
Supongamos que tenemos una superficie que puede ser definida de manera paramétrica  $\mathbf{p}(\mu, \nu)$ . Supongamos también que las derivadas parciales  $\frac{\partial \mathbf{p}}{\partial \mu} = \mathbf{p}_\mu$  y  $\frac{\partial \mathbf{p}}{\partial \nu} = \mathbf{p}_\nu$  existen y no son cero en ninguna parte de la superficie.

Podemos encontrar un vector normal unitario a la superficie por medio de la siguiente expresión:

$$\vec{\mathbf{n}}(\mu, \nu) = \frac{\mathbf{p}_\mu \times \mathbf{p}_\nu}{\|\mathbf{p}_\mu \times \mathbf{p}_\nu\|}. \quad (1.20)$$

El mapa de bordes puede imaginarse como un mapa de alturas de valores escalares  $h(\mu, \nu)$  que representa un desplazamiento de la superficie en dirección de la normal. Ver la Figura 1.22 en donde la superficie, representada por la linea negra continua, se desplaza en dirección del vector normal una cierta altura y se approxima la superficie perturbada (en lineas rojas punteadas).

La fórmula de los puntos en la superficie perturbada  $\mathbf{p}^*$  a partir de la de la superficie



**Figura 1.22:** La superficie real negra, es perturbada por medio del mapa de alturas  $h$ , para simular bordes resultando en la superficie roja punteada.

real  $\mathbf{p}$  es

$$\mathbf{p}^* = \mathbf{p} + h\vec{\mathbf{n}}. \quad (1.21)$$

En la Figura 1.22 las flechas de color negro representan  $h\vec{\mathbf{n}}$ , por lo tanto el mapa de bordes  $h$  se traduce en la longitud de estos vectores  $\vec{\mathbf{n}}$ . Por eso decimos que el mapa de bordes es una especie de *mapa de alturas*. Ahora deseamos encontrar las normales a la superficie perturbada, en la Figura 1.22 son representadas por las flechas de color rojo y asignar a la superficie real esas normales. Las normales a la superficie perturbada  $\mathbf{p}^*$  pueden calcularse encontrando primero las derivadas parciales

$$\begin{aligned} \frac{\partial \mathbf{p}^*}{\partial \mu} &= \frac{\partial \mathbf{p}}{\partial \mu} + \frac{\partial h}{\partial \mu}\vec{\mathbf{n}} + h\frac{\partial \vec{\mathbf{n}}}{\partial \mu}, \\ \frac{\partial \mathbf{p}^*}{\partial \nu} &= \frac{\partial \mathbf{p}}{\partial \nu} + \frac{\partial h}{\partial \nu}\vec{\mathbf{n}} + h\frac{\partial \vec{\mathbf{n}}}{\partial \nu}. \end{aligned} \quad (1.22)$$

En las expresiones anteriores podemos asumir que los últimos términos valen cero. La justificación de esta simplificación es que en general si la superficie original es suave las normales no pueden cambiar mucho a lo largo de ella. Además como sólo permitimos pequeños bordes en la superficie también esperamos que la perturbación  $h$  sea pequeña. Sin embargo, como queremos que los bordes se noten no podemos asumir que las derivadas  $h_\mu$  y  $h_\nu$  son pequeñas

$$\begin{aligned} \frac{\partial \mathbf{p}^*}{\partial \mu} &\approx \frac{\partial \mathbf{p}}{\partial \mu} + \frac{\partial h}{\partial \mu}\vec{\mathbf{n}}, \\ \frac{\partial \mathbf{p}^*}{\partial \nu} &\approx \frac{\partial \mathbf{p}}{\partial \nu} + \frac{\partial h}{\partial \nu}\vec{\mathbf{n}}. \end{aligned} \quad (1.23)$$

Ahora el vector normal  $\mathbf{m}$  que estamos buscando es simplemente el producto cruz de las derivadas parciales

$$\begin{aligned}\mathbf{m} &\approx \left( \frac{\partial \mathbf{p}}{\partial \mu} + \frac{\partial d}{\partial \mu} \vec{\mathbf{n}} \right) \times \left( \frac{\partial \mathbf{p}}{\partial v} + \frac{\partial d}{\partial v} \vec{\mathbf{n}} \right) \\ &= \left( \frac{\partial \mathbf{p}}{\partial \mu} \times \frac{\partial \mathbf{p}}{\partial v} \right) + \left( \frac{\partial d}{\partial \mu} \vec{\mathbf{n}} \times \frac{\partial \mathbf{p}}{\partial v} \right) - \left( \frac{\partial d}{\partial v} \vec{\mathbf{n}} \times \frac{\partial \mathbf{p}}{\partial \mu} \right).\end{aligned}\quad (1.24)$$

Recordemos que los vectores para iluminación deben ser unitarios, así que en realidad a cada punto de la superficie le asignamos la normal  $\vec{\eta} = \frac{\mathbf{m}}{\|\mathbf{m}\|}$  este vector es representado por las flechas rojas en la Figura 1.22. También podemos ver de (1.24) que  $\vec{\eta}$  no depende explícitamente del mapa de alturas  $d$  si no más bien de sus derivadas parciales.

Una aproximación de  $h_\mu$  y  $h_\nu$  se construye por diferencias finitas en la imagen que representa el mapa. Es también común que en vez de guardar el mapa de alturas se guarden un par de imágenes obtenidas por diferencias finitas sobre la imagen original que directamente representan  $h_\mu$  y  $h_\nu$ .

También se debe señalar que  $\vec{\eta}$  no está definida en lugares en donde  $p_\mu$  o  $p_v$  valen cero. Por esta razón se debe tener cuidado al asignar normales en esas regiones. Una manera de resolver el problema es interpolar linealmente las normales en una cierta vecindad de estos puntos críticos.



# Capítulo 2

## Ponderado de normales

Se desea encontrar una forma de asignar las normales a la superficie que se usarán para hacer la iluminación. La manera propuesta sera crear primero una superficie implícita que envuelva la aproximación poligonal  $\mathcal{A}_\tau$ . La superficie implícita esta formada por funciones base, por lo tanto cada una de estas contribuye a la superficie y contribuye también a la forma como se van a modificar las normales. En este sentido en este trabajo entenderemos como *ponderación* al proceso de ir sumando vectores que modifican la dirección de un vector normal. Dado que para hacer cálculos de iluminación los vectores deben ser unitarios, después de la ponderación hacemos un postproceso que nos asegure solo usar vectores unitarios. Por esta razón se hace énfasis en que las funciones base cambian *la dirección* del vector normal.

### 2.1 Representación implícita de superficies

Tradicionalmente en las GC hay dos formas de modelar superficies en el espacio  $\mathbb{R}^3$  [5]. Una de ellas es tener la superficie definida de manera paramétrica  $s(\mu, \nu)$ . Por lo tanto es posible crear una aproximación por medio de polígonos calculando los vértices mediante incrementos constantes en los parámetros.

Otra forma, es la llamada representación implícita. Que consiste en encontrar las soluciones de la siguiente ecuación:

$$f(\mathbf{x}) = 0, \quad (2.1)$$

donde  $\mathbf{x} \in \mathbb{R}^3$ . Aunque ya se habían estudiado las superficies producidas por funciones de segundo grado, Blinn propuso una manera de generalizar estas superficies en [5]. La idea de Blinn tenía el objetivo de representar estructuras moleculares donde los átomos se combinan con otros con transiciones graduales. Esta representación ha sido adoptada para realizar modelado de objetos orgánicos y funciones topológicamente muy complicadas. En el trabajo original una molécula es representada aproximadamente por medio de la combinación lineal de funciones suaves. En dicha representación, cada  $j$ -esimo átomo de la molécula es representado por una función suave  $b_j$  ponderada por un coeficiente  $c_j$  y centrada en el punto  $\mathbf{p}_j$  de la siguiente forma

$$g(\mathbf{x}) \approx \sum_{j=1}^J c_j b_j(\mathbf{x} - \mathbf{p}_j). \quad (2.2)$$

Blinn propuso utilizar la siguiente función  $b$  (radialmente simétrica):

$$b(\alpha; r) = e^{-\alpha r}, \quad (2.3)$$

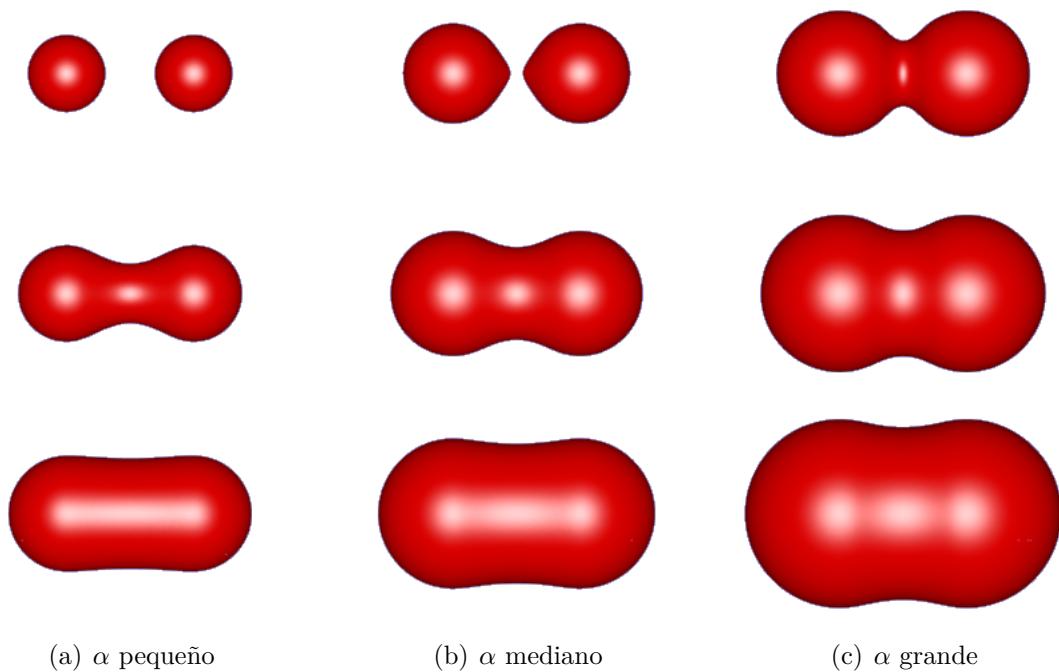
donde  $r$  es la distancia de  $\mathbf{x}$  al punto  $\mathbf{p}_i$  donde está centrada la función y  $\alpha$  representa un parámetro de forma de la función (el ancho).

Para llevar a cabo la visualización de (2.2), se define un umbral o isovalor  $t$ , por lo que se obtiene una superficie  $S_t$  ver (1.5), que se aproxima por poligonalización o por *raycasting*. Claramente el resultado de la aproximación (2.2) depende de la elección de los conjuntos  $\{\mathbf{p}_j\}$  y  $\{c_j\}$  y de la elección de los parámetros de (2.3), ver Figura 2.1.

En la literatura de GC se conoce a la combinación lineal (2.2) como *modelo blobby* u orgánico y a las diversas funciones base  $b$  como *metaballs* o blobs. En [39] se propone la función base

$$b(\kappa, a; r) = \begin{cases} \kappa \left(1 - \frac{3r^2}{a^2}\right), & \text{si } 0 \leq r \leq \frac{a}{3}, \\ \frac{3\kappa}{2} \left(1 - \frac{r}{a}\right)^2, & \text{si } \frac{a}{3} \leq r \leq a, \\ 0, & \text{en otro caso,} \end{cases} \quad (2.4)$$

donde  $a$  representa la extensión o soporte de la función y  $\kappa$  determina la densidad máxima de la función. En [39] los autores se refieren a (2.4) como *metaball*, dando origen al término genérico *metaballs*. En [50] se usa una función alternativa conocida



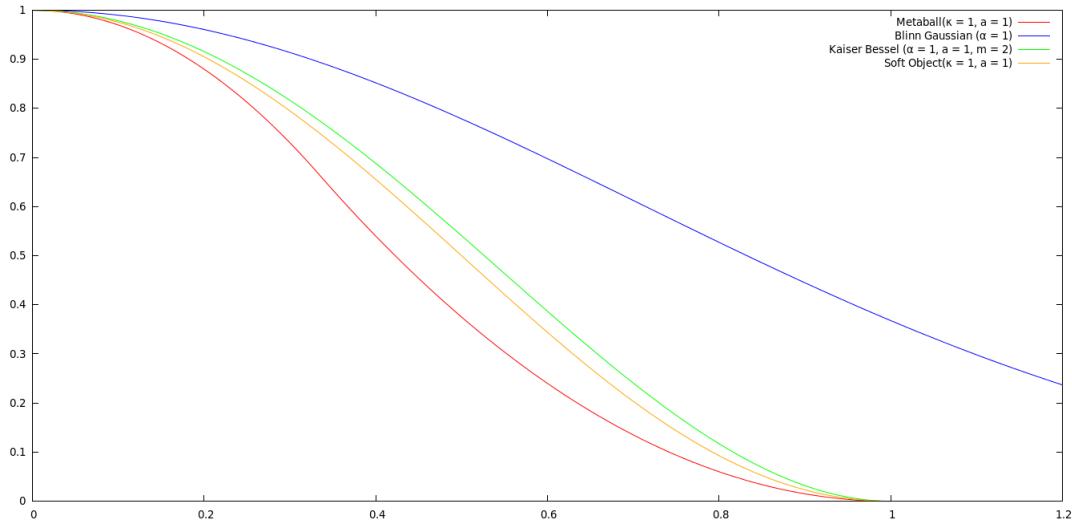
**Figura 2.1:** Ilustración del efecto de *blobbiness* para diferentes radios  $r$  y  $\alpha$  de blobs de Blinn. El umbral  $t$  y la posición de los blobs se mantiene constante. El radio  $r$  se incrementa de arriba a abajo y la  $\alpha$  de izquierda a derecha.

como *soft object*, definida como:

$$b(\kappa, a; r) = \begin{cases} \kappa \left( 1 - \frac{4r^6}{9a^6} + \frac{17r^4}{9a^4} - \frac{22r^2}{9a^2} \right), & \text{si } 0 \leq r \leq a, \\ 0, & \text{en otro caso,} \end{cases} \quad (2.5)$$

donde los parámetros  $a$  y  $\kappa$  representan los mismos conceptos que en (2.4).

En la Figura 2.2 se muestra una comparación entre varias funciones blobs. El blob de (2.3) etiquetado como *Blinn Gaussian*, el blob de (2.4) etiquetado como *Metaball*, el blob de (2.5) etiquetado como *Soft Object* y el blob *Kaiser-Bessel* que se analiza a detalle en la siguiente sección.



**Figura 2.2:** Comparación entre los diferentes blobs.

### 2.1.1 Las funciones Kaiser-Bessel generalizadas (*blobs*)

La aproximación (2.2) también sirve para modelar funciones de densidad en el área de reconstrucción a partir de proyecciones (un problema inverso cuyo ejemplo más típico son los instrumentos médicos que realizan tomografías [25]). En particular son usados en aquellos métodos basados en expansiones en series de funciones base. En este campo también se han usado varias funciones base tales como véxeles. Pero una función base que ha resultado en buenas aproximaciones son las funciones Kaiser-Bessel generalizadas

[33], definidas como:

$$b(m, \alpha, a; r) = \begin{cases} \frac{I_m\left(\alpha\sqrt{1 - \left(\frac{r}{a}\right)^2}\right)}{I_m(\alpha)} \left(\sqrt{1 - \left(\frac{r}{a}\right)^2}\right)^m, & \text{si } 0 \leq r \leq a, \\ 0, & \text{en otro caso,} \end{cases} \quad (2.6)$$

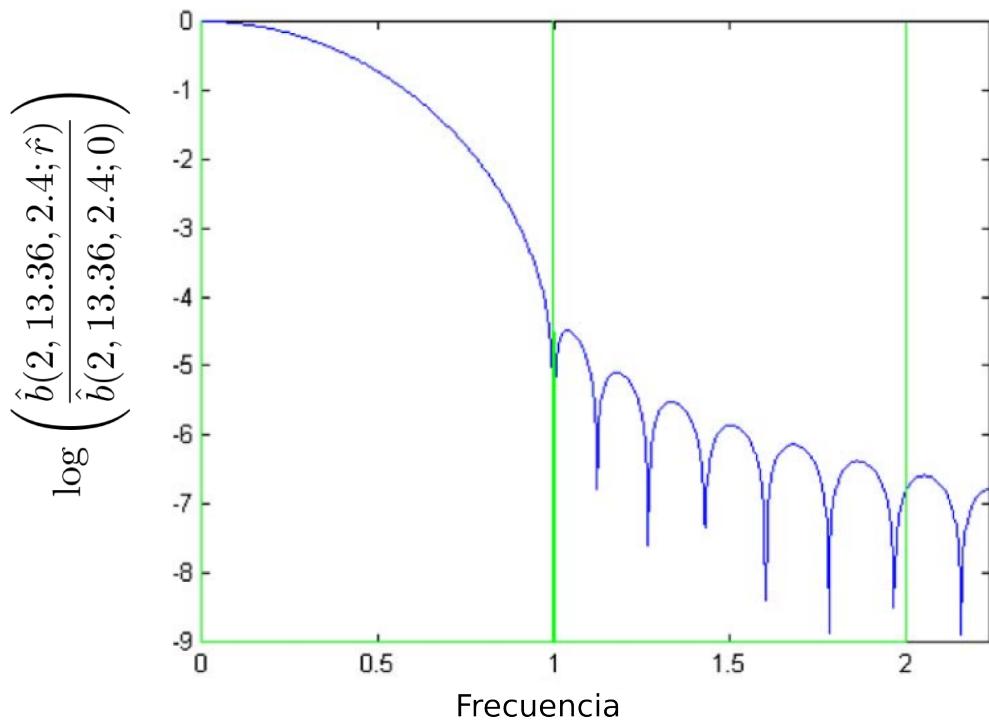
en donde  $r$  es la distancia radial al centro del blob,  $I_m$  es la función Bessel modificada de orden  $m$ ,  $a$  determina el soporte y  $\alpha$  es el parámetro que controla la forma de la función.

La elección de las funciones Kaiser-Bessel generalizadas se debe a sus propiedades adecuadas para el área de procesamiento de imágenes, tales como la continuidad en el soporte finito y su rápida tasa de desvanecimiento en su espectro (algo que se considera como ancho de banda limitado en la práctica), ver Figura 2.3. De hecho, estas funciones son muy conocidas en el área de procesamiento de señales donde son utilizadas típicamente como ventanas (*windows* en inglés). De ahora en adelante al usar el término *blob* nos referiremos a la función base de (2.6).

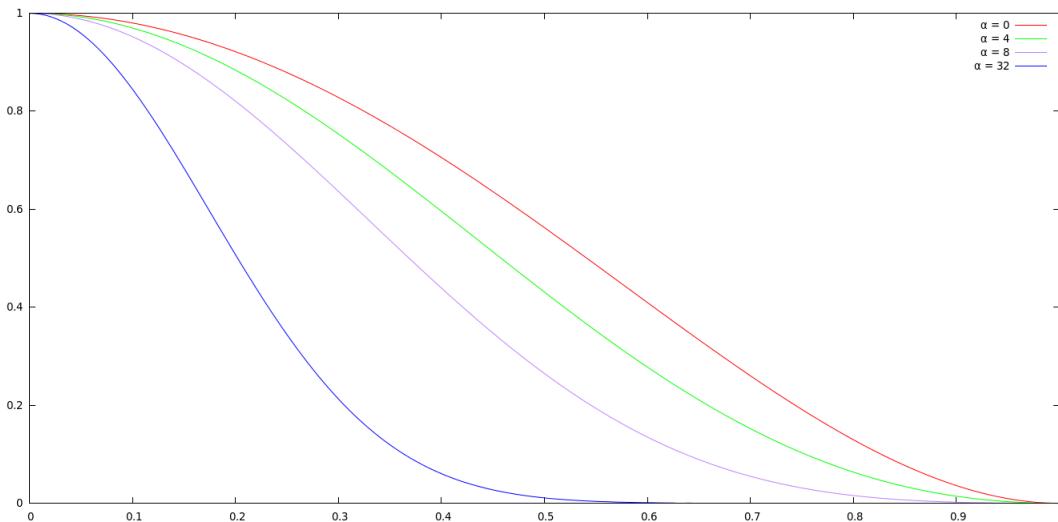
El orden  $m$  del blob es un entero positivo que determina el número de derivadas que tiene la función en su punto  $a$ . Como es bien sabido este parámetro está relacionado con la tasa de cambio de la función y en visualización es comúnmente asociado con la suavidad de la función. Para el orden dos, el blob es ya una función suave que tiene derivada en todo el soporte, además nadie ha demostrado que elecciones de  $m$  mayores sean más efectivas. Por lo que al igual que en [17], aquí fijaremos el valor  $m = 2$ .

Otro parámetro que afecta la forma del blob es el parámetro  $\alpha$ , mientras  $\alpha$  sea más pequeño el blob decrece más lentamente. La Figura 2.4 muestra la variación del blob con respecto a  $\alpha$ , dejando como constantes  $a = 1$  y  $m = 2$ .

En este trabajo utilizaremos la aproximación (2.2), con las funciones base definidas por (2.6) para poner normales sobre la malla  $\mathcal{A}_\tau$  generada por el algoritmo de Artzy y visualizar usando el modelo de (1.15). Como hemos mencionado antes, para hacer iluminación es necesario conocer el gradiente a la superficie y consecuentemente la normal [9].



**Figura 2.3:** Aquí se gráfica la magnitud de la transformada de Fourier del blob (espectro) en escala logarítmica contra la distancia radial  $\hat{r}$  al centro del blob (frecuencia).



**Figura 2.4:** Ilustración de diferentes blobs con  $a = 1$  y  $m = 2$  variando  $\alpha$  en el blob de (2.6), mientras  $\alpha$  aumenta el blob decae más rápido.

Para la aproximación (2.2) el gradiente se obtiene como

$$\nabla g(\mathbf{x}) = \nabla \sum_{j=1}^j c_j b_j(\mathbf{x} - \mathbf{p}_j) = \sum_{j=1}^j c_i \nabla b_j(\mathbf{x} - \mathbf{p}_j). \quad (2.7)$$

Esto quiere decir que el gradiente de la superficie (y por ende un vector normal a la superficie) se puede obtener sumando las contribuciones de los gradientes de cada blob que forma la superficie.

En [33] Lewitt ha proporcionado una forma analítica para obtener el gradiente de (2.6) como sigue:

$$\frac{\partial b}{\partial r} = \begin{cases} \left[ \frac{-1/\alpha}{\alpha^{m-2} I_m(\alpha)} \right] (r/a) z^{m-1} I_{m-1}(z), & \text{si } 0 \leq r \leq a, \\ 0, & \text{en otro caso,} \end{cases} \quad (2.8)$$

en donde  $z = \sqrt{\alpha(1 - (r/a)^2)}$ .

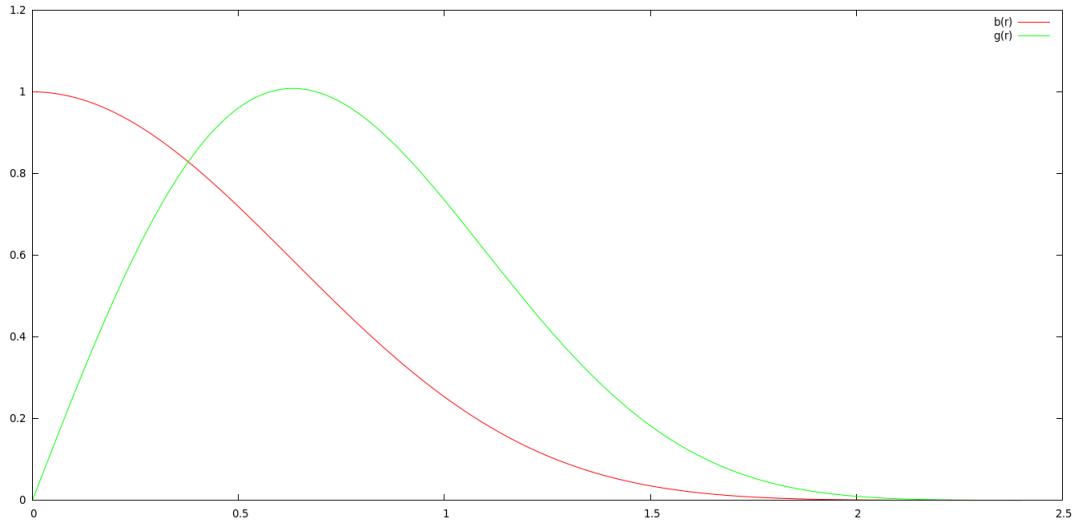
Para una función radialmente simétrica el gradiente puede calcularse fácilmente como

$$\begin{aligned} \nabla b(\mathbf{x} - \mathbf{p}_j) &= \nabla b(r(\mathbf{x} - \mathbf{p}_j)) \\ &= \frac{\partial b}{\partial r} \frac{\partial}{\partial \mathbf{x}} r(\mathbf{x} - \mathbf{p}_j), \end{aligned} \quad (2.9)$$

donde  $r$  es la función de distancia euclídea  $r(\mathbf{x} - \mathbf{p}_j) = \sqrt{\sum_{i=1}^3 (x_i - p_{i,j})^2}$ . Es fácil ver que  $\frac{\partial}{\partial \mathbf{x}} r(\mathbf{x} - \mathbf{p}_j) = \frac{\mathbf{x} - \mathbf{p}_j}{|\mathbf{x} - \mathbf{p}_j|}$ , por lo tanto podemos concluir que:

$$\nabla g(\mathbf{x}) = \sum_{j=1}^j c_j \frac{\partial b}{\partial r} \left( \frac{\mathbf{x} - \mathbf{p}_j}{|\mathbf{x} - \mathbf{p}_j|} \right). \quad (2.10)$$

Esto quiere decir que el gradiente es un vector que va del centro  $\mathbf{p}_j$  del blob hasta el punto  $\mathbf{x}$  cuya magnitud es  $\frac{\partial b}{\partial r}$ . Finalmente, se muestra en la Figura 2.5 la gráfica del blob  $b(r)$  y de su gradiente  $\frac{\partial b}{\partial r}$ .



**Figura 2.5:** La gráfica muestra el valor del blob con parámetros  $b(a = 2.4, \alpha = 13.36, m = 2)$  y la magnitud de su gradiente calculado con (2.8).

## 2.2 Contribución de la tesis

El objetivo de el trabajo es poder mejorar la calidad visual de la malla de Arzy  $\mathcal{A}_\tau$  por medio de técnicas de GC.

Para este fin se planea modificar los vectores normales a la superficie y usar el modelo de iluminaciónde Phong 1.15. La manera como se hace esta modificación es creando una superficie implícita que envuelve la malla  $\mathcal{A}_\tau$  y calculando las normales en dicha superficie implícita.

La primera aportación del trabajo consistio en encontrar los parametros de los blobs para visualización en tres configuraciones del conjunto  $\{\mathbf{p}_j\}$ . Aunque se usa el método descrito en [17] en dicho trabajo solo se obtienen parámetros para una configuración.

La aportación mas importante es la manera como se construye la malla con la que se hace la ponderación de las normales. Este proceso es descrito en la sección 2.2.2.

Finalmente, también se hacen experimentos para probar la tecnica propuesta en la esta sección. Dichos experimentos son explicados en el capítulo 3 y tambien constituyen una aportación.

### 2.2.1 Optimización de blobs para visualización

Claramente la elección de los parámetros  $\alpha$ ,  $a$ ,  $m$  y la distribución de las funciones bases  $\{\mathbf{p}_j\}$  afectará los resultados de la aproximación (2.2). Debido a que en GC se conoce con toda certeza la función a modelar es posible seleccionar un conjunto  $\{b_j\}$  con diferentes funciones base o una función base con diferentes parámetros correspondientes. Por ejemplo, en regiones muy homogéneas se pueden utilizar funciones mas anchas y en regiones con mas detalle usar funciones mas finas. En contraste, en el caso de la reconstrucción no se conoce, en general, la naturaleza de la función  $v$  a aproximar, por lo tanto se usa una función base cuyos parámetros tienen que ser determinados bajo un criterio que tome en cuenta la distribución del conjunto  $\{\mathbf{p}_j\}$ .

Existen varias posibles distribuciones del conjunto  $\{\mathbf{p}_j\}$ , llamadas rejillas. En este trabajo nos interesan las tres siguientes:

1. La rejilla cubica simple (*sc*) definida por

$$G_\Delta = \{\Delta \mathbf{x} | \mathbf{x} \in \mathbb{Z}^3\}. \quad (2.11)$$

2. La rejilla *bcc* que viene del termino *body centered cubic grid*, que se define como

$$B_\Delta = \{\Delta \mathbf{x} | \mathbf{x} \in \mathbb{Z}^3 \text{ y con } x_1 \equiv x_2 \equiv x_3 \pmod{2}\}. \quad (2.12)$$

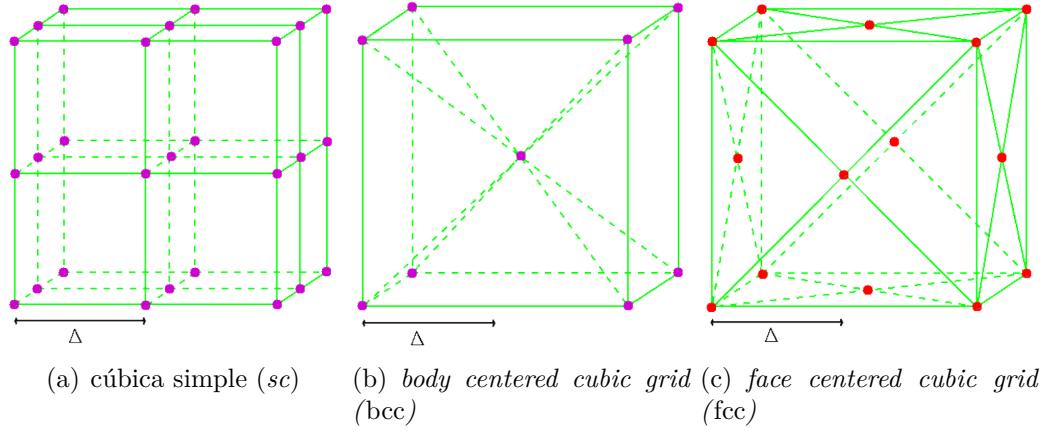
3. La rejilla *fcc* o *face centered cubic grid* definida como

$$F_\Delta = \{\Delta \mathbf{x} | \mathbf{x} \in \mathbb{Z}^3 \text{ y con } x_1 + x_2 + x_3 \equiv 0 \pmod{2}\}. \quad (2.13)$$

Para todas estas rejillas  $\Delta$  es la distancia de muestreo. La Figura 2.6 muestra una porción de estas rejillas; como las rejillas son regulares, el resto del espacio se llena repitiendo las porciones mostradas en la figura de la forma mas natural.

Definimos también las funciones  $\text{III}_{G_\Delta}$ ,  $\text{III}_{B_\Delta}$  y  $\text{III}_{F_\Delta}$  generadas a partir de poner pulsos unitarios en las posiciones de la rejilla correspondiente.

En [17] se propone el siguiente criterio para elegir una distribución basado en el procesamiento de señales. Considerando que se desea aproximar una función constante por medio de (2.2) con coeficientes  $c_j = 1$ , entonces el volumen puede expresarse como:



**Figura 2.6:** Diferentes rejillas cúbicas.

$$v(\mathbf{x}) \approx \text{III}_G \circledast b(\mathbf{x}), \quad (2.14)$$

donde el operador  $\circledast$  representa la convolución y  $\text{III}_G$  un tren de pulsos unitarios colocados sobre cierta rejilla  $G$ .

Para continuar con el análisis requerimos de definir la transformada de Fourier para una función  $g(\mathbf{x})$  como

$$\mathcal{F}(g) = \hat{g}(\boldsymbol{\xi}) = \int_{\mathbb{R}^n} g(\mathbf{x}) e^{-i2\pi \langle \mathbf{x}, \boldsymbol{\xi} \rangle} d\mathbf{x}, \quad (2.15)$$

donde  $\langle \mathbf{x}, \boldsymbol{\xi} \rangle$  representa el producto interno entre los vectores  $\mathbf{x}$  y  $\boldsymbol{\xi}$ . Se hace notar que seguimos la convención de usar  $\hat{g}$  en vez de  $\mathcal{F}(g)$  para denotar la transformada de Fourier de  $g$ . Se adopta también la convención de usar  $\boldsymbol{\xi}$  como un vector en el espacio de Fourier y de  $R = |\boldsymbol{\xi}|$ . Además representar la transformada de Fourier de  $f$  como  $\hat{f}$ , como se indica en (2.15).

En [33] Lewitt proporciona una la siguiente forma de calcular la transformada de Fourier del blob (2.6) cuando  $m = 2$ .

$$\hat{b}(2, \alpha, a; R) = \frac{(2\pi)^{3/2} a^3 \alpha^2}{I_2(\alpha)} \begin{cases} \frac{I_{7/2}(\sqrt{\alpha^2 - (2\pi a R)^2})}{(\sqrt{\alpha^2 - (2\pi a R)^2})^{7/2}}, & \text{si } 2\pi a R \leq \alpha, \\ \frac{J_{7/2}(\sqrt{(2\pi a R)^2 - \alpha^2})}{(\sqrt{(2\pi a R)^2 - \alpha^2})^{7/2}}, & \text{si } 2\pi a R \geq \alpha, \end{cases} \quad (2.16)$$

en donde  $J_i$  es la función Bessel de orden  $i$ . Existe una relación entre la convolución y la transformada de Fourier dada por el siguiente teorema

**Teorema. Convolución:** Sean  $f$  y  $g$  dos funciones en  $\mathbb{R}^n$ , entonces

$$\begin{aligned} \widehat{f \circledast g} &= \hat{f} \times \hat{g}, \\ \widehat{f \times g} &= \hat{f} \circledast \hat{g}. \end{aligned} \quad (2.17)$$

El operador  $\times$  es la multiplicación punto a punto. Por lo tanto, la convolución de (2.14) puede expresarse en el espacio de Fourier como:

$$\hat{v}(\mathbf{x}) \approx \widehat{\Pi} \times \hat{b}(\xi). \quad (2.18)$$

En [15] se obtienen formulas para la transformada de Fourier de las rejillas antes expuestas, definidas como:

$$\widehat{\Pi}_{G_\Delta} = \frac{1}{\Delta^3} \Pi_{G_{1/\Delta}}, \quad (2.19)$$

$$\widehat{\Pi}_{B_\Delta} = \frac{1}{4\Delta^3} \Pi_{F_{1/2\Delta}}, \quad (2.20)$$

$$\widehat{\Pi}_{F_\Delta} = \frac{1}{2\Delta^3} \Pi_{B_{1/2\Delta}}. \quad (2.21)$$

Cabe hacer notar que las rejillas *bcc* y *fcc* son reciprocas en el espacio de Fourier (ver (2.20) y (2.21)).

Aunque el arreglo de pulsos mas común es aquel definido por  $\Pi_{G_\Delta}$ , en [40] se ha demostrado que el muestreo mas eficiente del espacio  $\mathbb{R}^3$  es con  $\Pi_{B_\Delta}$ . Por esa razón en el análisis de [36] se utiliza la distribución  $\Pi_B$  para el conjunto  $\{\mathbf{p}_j\}$ .

Recordemos que deseamos aproximar una función constante y que la transfromada de Fourier de dicha función es un pulso en el origen. Por esta razones los autores de [17]

sugieren que el primer cero de (2.16) caiga exactamente en el pulso mas cercano al origen del tren de pulsos  $\widehat{\text{III}}_{B_\Delta}$ , que como sabemos de (2.20) es  $\frac{1}{4\Delta^3} \text{III}_{F_{1/2\Delta}}$ .

Revisando (2.16), como  $I_m$  nunca vale cero, debemos hacer que  $J_{7/2}(u) = 0$ , el primer valor que esto ocurre es  $u = 6.987932$ . Por lo que debemos hacer

$$\sqrt{(2\pi aR)^2 - \alpha^2} = u, \quad (2.22)$$

de donde sabemos que el valor para  $\alpha$  debe cumplir

$$\alpha = \sqrt{(2\pi aR)^2 - u^2}. \quad (2.23)$$

La ecuación (2.23) permite relacionar el tamaño del blob  $a$  y la separación de la rejilla  $\Delta$  con la forma del blob  $\alpha$ . Para mas detalles de esta relación pueden verse [15] y [17]. A pesar de que el análisis anterior produce buenos resultados para reconstrucción a partir de proyecciones, en [17] se demostró que produce artefactos al momento de visualizar una función  $v$  aproximada por (2.2).

Para distintas rejillas el valor de  $R$  en (2.23) es también diferente. En cada rejilla debe ser el lugar del vecino mas cercano en el espacio de Fourier. Usaremos  $\lambda$  para denotar la separación de los puntos de las rejillas en el espacio de Fourier.

Para la rejilla  $sc$  sabemos que su transformada en el espacio de Fourier es otra rejilla  $sc$  por (2.19). Por lo tanto sus vecino mas cercano esta a distancia  $R = \lambda$ , ver la Figura 2.6(a). Pero sabemos también de (2.19) que en Fourier  $\lambda = \frac{1}{\Delta}$ , por lo tanto  $R = \frac{1}{\Delta}$  y el valor de  $\alpha$  es

$$\alpha = \sqrt{4\pi^2 \left(\frac{a}{\Delta}\right)^2 - u^2}. \quad (2.24)$$

De (2.20) se puede ver que la transformada de Fourier de  $bcc$  es la rejilla  $fcc$  por lo tanto estamos a distancia  $R = \lambda\sqrt{2}$  ver Figura 2.6(c) pero sabemos que la separación de la rejilla transformada esta colocada a distancia  $\lambda = \frac{1}{2\Delta}$  (ver (2.21)). Por lo tanto el valor es  $R = \lambda\sqrt{2} = \frac{1}{\sqrt{2}\Delta}$  y el valor de  $\alpha$  es:

$$\alpha = \sqrt{2\pi^2 \left(\frac{a}{\Delta}\right)^2 - u^2}. \quad (2.25)$$

Por último en la rejilla  $fcc$  su transformada es la rejilla  $bcc$  por lo tanto estamos a

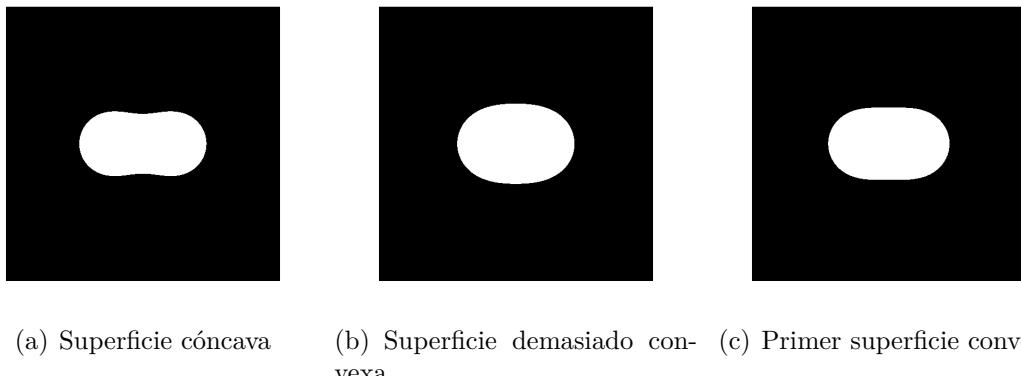
distancia  $R = \lambda\sqrt{3}$ ; Figura 2.6(b), pero la separación de la rejilla transformada esta colocada a distancia  $\lambda = \frac{1}{2\Delta}$ . Por lo tanto el valor es  $R = \lambda\sqrt{3} = \frac{\sqrt{3}}{2\Delta}$  y el valor de  $\alpha$  es:

$$\alpha = \sqrt{3\pi^2 \left(\frac{a}{\Delta}\right)^2 - u^2}. \quad (2.26)$$

Tenemos ahora una relación para  $\alpha$  en función de  $a$  y  $\Delta$  para cada rejilla. Lo único que nos resta por determinar es cual es el valor de  $a$  ideal en cada caso.

Continuando con el procedimiento descrito en [17], se propone el siguiente criterio. Dos blobs colocados en puntos mas cercanos de alguna rejilla, con coeficientes  $c_1 = c_2 = 1$ , al visualizarse con un umbral de  $t = 0.5$  deben formar la primer superficie convexa.

Esto quiere decir que se ponen dos blobs iguales en puntos cercanos de la rejilla, estos blobs forman una superficie implícita como las de la Figura 2.4. Cuando esta superficie es umbralizada a  $t = 0.5$ , se busca que la figura que formen sea la primera superficie convexa. Si los blobs tienen un  $a$  muy grande la superficie tiende a ser muy convexa, si por el contrario  $a$  es pequeño la superficie sera cóncava. En la Figura 2.7 se muestran contornos de dichas superficies. En el primer caso, Figura 2.7(a)  $a$  es pequeña y la superficie es cóncava. En el segundo caso,  $a$  es grande y la superficie es muy convexa, ver la Figura 2.7(b). El último caso Figura 2.7(c) muestra la superficie que estamos buscando.



**Figura 2.7:** Contorno de la superficie formada por dos blobs con los mismos parámetros en puntos vecinos de una rejilla y umbralizada a  $t = 0.5$ .

Para obtener el blob óptimo para visualización se uso el siguiente criterio. Usamos  $\Delta$  en cada rejilla tal que  $R = 1$ . Luego se hizo una búsqueda numérica variando el valor de

Rejilla	$a$	$\alpha$
$sc$	1.13	1.21
$bcc$	2.4	13.36
$fcc$	2.4	13.36

**Cuadro 2.1:** Resultados de optimizar  $a$  y su correspondiente  $\alpha$  para las diferentes rejillas.

$a$  y construyendo blobs que satisfagan la correspondiente ecuación (2.23) de su rejilla. Los resultados obtenidos se muestran en la Tabla 2.1.

Es interesante notar que debido a la relación que guardan las rejillas  $fcc$  y  $bcc$  en el espacio de Fourier (2.21) y (2.20), obtuvimos el mismo blob en ambos casos. También hay que aclarar que el blob esta escalado al momento de fijar  $\Delta$  y dado que las relaciones de  $\alpha$  dependen del cociente  $\frac{a}{\Delta}$  solo es necesario multiplicar el soporte del blob  $a$  para adaptarlo al  $\Delta$  del conjunto de datos.

### 2.2.2 Ponderado de normales por medio de blobs

Si se conociera el conjunto de coeficientes  $\{c_j\}$  y los parámetros de los blobs  $b_j$  que mejor aproximan el volumen de (2.2), podríamos calcular las normales directamente en todo el campo escalar usando (2.10).

En nuestro caso no se tiene información del campo escalar  $f$  que queremos visualizar. Pero sabemos que  $g$  como en (2.2) se aproxima a  $f$  y a su vez  $v$  es una aproximación de  $g$ . Se desea visualizar a  $S_\tau$  definido en (1.5) pero solo tenemos la aproximación  $\mathcal{A}_\tau$  proveniente del Algoritmo de Artzy. Por lo tanto vamos a poner  $\mathcal{A}_\tau$  dentro de la superficie implícita  $g$  y usaremos la superficie para calcular normales de iluminación de  $\mathcal{A}_\tau$ .

Como la salida  $\mathcal{A}_\tau$  del Algoritmo de Artzy esta formada por una malla que esta dentro de  $g$  y se conoce la posición de los vértices de la malla podríamos calcular normales para la malla  $\mathcal{A}_\tau$  usando el gradiente  $\nabla g$  definido en (2.10) y estas deben ser una buena aproximación a las normales de  $S_\tau$ .

Debido a la falta de información solo podemos usar ciertos criterios para construir la aproximación de (2.2). Los criterios propuestos son los siguientes:

- Vamos a poner el conjunto  $\{\mathbf{p}_j\}$  en el centro de cada vóxel de la rejilla  $sc$ .
- Para todos los vóxeles que el algoritmo de Artzy encontró como interiores (Es decir que tenían un valor  $v(\mathbf{k}) \geq \tau$ ), los coeficientes serán  $c_j = 1$ .
- Para todos los vóxeles que el algoritmo de Artzy encontró como exteriores ( $v(\mathbf{k}) < \tau$ ), los coeficientes serán:  $c_j = 0$ .
- Las funciones base  $\{b_j\}$  son blobs y todos tendrán el mismo conjunto de parámetros  $m$ ,  $\alpha$  y  $a$  determinados por el método expuesto en la sección anterior.

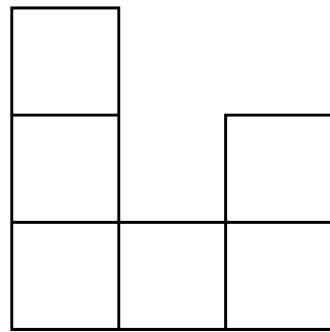
Bajo las condiciones arriba propuestas tenemos una manera de calcular (2.10) y por consecuencia tenemos una manera de calcular el conjunto de vectores normales  $\vec{n}$  en cada vértice de la malla  $\mathcal{A}_\tau$ . Usando esta información y el modelo de iluminación (1.15) esperamos tener resultados visiblemente mas agradables que los arrojados por el Algoritmo de Artzy.

Ilustrando los criterios arriba expuestos esto quiere decir que asumiremos que hay un blob en cada vóxel que esta dentro de la malla  $\mathcal{A}_\tau$  que como se comentó en el capítulo anterior es una superficie cerrada. Con esos blobs se forma una superficie implícita a la que le podemos calcular su normal en cualquier punto y luego usamos esa normal en los vértices de la malla  $\mathcal{A}_\tau$ . Todo esto se simboliza en la Figura 2.8, donde por simplicidad se muestra en solo en 2D.

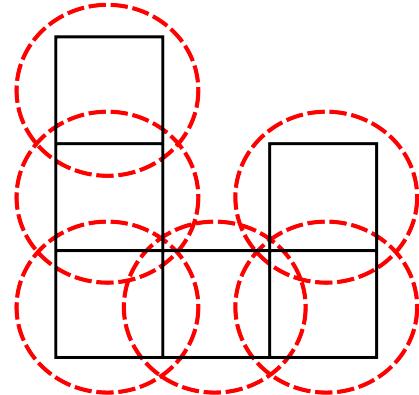
En la Figura 2.8(a) se muestra una posible malla  $\mathcal{A}_\tau$  arrojada por el Algoritmo de Artzy. Se colocan blobs  $b_j$  en los centros de los vóxeles interiores de la malla  $\{\mathbf{p}_j\}$  estos forman una superficie implícita  $g$  como la de (2.2), que se muestra en verde en la Figura 2.8(c). Como  $\mathcal{A}_\tau$  esta dentro de la superficie  $g$  podemos usar (2.10) para calcular el gradiente en cualquier punto  $\mathbf{p}$ , en particular nos interesa calcularlo en los vértices de la malla  $\mathcal{A}_\tau$ .

Para hacer la ponderación de las normales de la salida del Algoritmo de Artzy, usamos el Algoritmo 2. Las entradas de este algoritmo son la malla de salida del Algoritmo de Artzy  $\mathcal{A}_\tau$  y los parámetros  $a$ ,  $m$  y  $\alpha$  del blob usado para construir la superficie implícita.

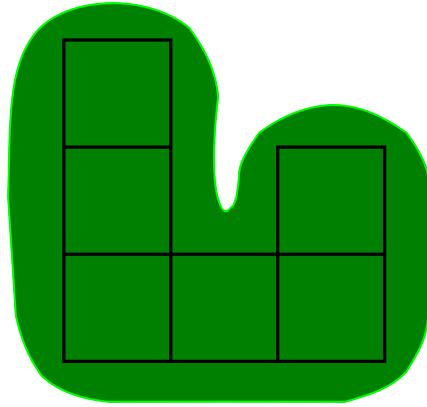
En resumen, una vez que se tiene la aproximación  $\mathcal{A}_\tau$ , se hace el posprocesamiento descrito en el Algoritmo 2, para obtener un conjunto de normales a la superficie, usando alguno de los blobs óptimos de la sección anterior.



(a) Malla obtenida por el Algoritmo de Artzy  $\mathcal{A}_\tau$



(b) Conjunto  $\{b_j\}$  con  $c_j \neq 0$



(c) Superficie implícita

**Figura 2.8:** Al colocar los blobs en los centros de los véxeles estos definen una superficie implícita en donde podemos calcular al gradiente en cada punto.

---

**Algoritmo 2** Ponderado de Normales

---

**Entradas:** Una malla de salida del algoritmo de Artzy  $\mathcal{A}_\tau$  y los parámetros  $a$ ,  $m$  y  $\alpha$  de un blob  $b$ .

**Salidas:** Un conjunto de normales  $M$  a cada punto vértice de la  $\mathcal{A}_\tau$

```

1: Calcula el tamaño de la escena  $\Gamma$  de donde viene  $\mathcal{A}_\tau$ 
2: Calcula el conjunto  $I$  de todos los vóxeles en  $\Gamma$  interiores de la malla  $\mathcal{A}_\tau$ .
3: for all  $\mathbf{k} \in \Gamma$  do
4:   if  $\mathbf{k} \in I$  then
5:     Calcula la vecindad  $N_\mathbf{k} = \{\mathbf{a} | a > |\mathbf{a} - \mathbf{k}|\}$  del punto  $\mathbf{k}$ .
6:     for all  $\mathbf{p} \in N_\mathbf{k}$  do
7:       if  $\mathbf{p} \in \mathcal{A}_\tau$  then
8:         Calcular el vector  $\mathbf{q} = \mathbf{p} - \mathbf{k}$ 
9:         Calcular la magnitud del gradiente  $\eta = \frac{\partial}{\partial r} b(a, \alpha, m; |\mathbf{q}|)$ 
10:        Acumular a la normal  $\mathbf{n}_\mathbf{p} \in M$  del punto  $\mathbf{p}$  la contribución de este gra-
           diente:  $\mathbf{n}_\mathbf{p} = \mathbf{n}_\mathbf{p} + \eta \frac{\mathbf{q}}{|\mathbf{q}|}$ 
11:       end if
12:     end for
13:   end if
14: end for
15: for all  $\mathbf{n} \in M$  do
16:    $\mathbf{n} = \frac{\mathbf{n}}{|\mathbf{n}|}$ 
17: end for

```

---



# Capítulo 3

## Experimentos

Para probar la metodología expuesta en el capítulo anterior se realizaron varios experimentos que hemos dividido en dos partes: una con experimentos visuales sobre datos obtenidos de tomógrafos y la segunda parte con experimentos numéricos sobre modelos analíticos. Para todos nuestros experimentos hemos asumido, sin perdida de generalidad, que tenemos volúmenes discretizados  $G_1$  (en otras palabras  $\Delta = 1$ ). Por lo tanto hemos escalado los blobs de la Sección 2.1 de manera consistente (por ejemplo, para el blob diseñado para la rejilla  $bcc$  el radio es escalado por  $\frac{1}{\sqrt{2}}$ ).

Para la visualización de todas las mallas hemos utilizado el API gráfico OpenGL<sup>TM</sup>[45] la cual permite iluminar con el modelo de Phong (1.15) pero utiliza para sombrear el modelo de Gouraud (1.16) [43]. El software utiliza una luz direccional, ver Sección 1.2 cuya componente ambiental es igual a cero y sus otros dos componentes son de color blanco. La dirección de dicha luz es paralela al vector perpendicular al plano de proyección y tiene un sentido opuesto al observador.

Las propiedades del material son constantes en todos los casos y se proporciona en la Tabla 3.1 en relación al modelo de iluminación (1.14).

$\rho_a$	$\rho_d$	$\rho_s$	$\gamma$
(0.7, 0.7, 0)	(0.9, 0.9, 0)	(0, 0, 0)	1

**Cuadro 3.1:** Propiedades del material con el que se visualizan las mallas.  
Estos valores se usan en el modelo de iluminación (1.14)

### 3.1 Experimentos visuales

Para los primeros experimentos utilizamos conjuntos de datos disponibles en varios sitios públicos, por ejemplo de [42]. Estos conjuntos de datos han sido obtenidos por medio de TAC pero desconocemos el método usado para la obtención de la aproximación  $v_{G_\Delta}$  (los TAC usan métodos de aproximación y muestreo conocidos como reconstrucción por medio de proyecciones [25]). Por lo tanto desconocemos las funciones originales y nuestras comparaciones son puramente visuales.

Primero usamos un conjuntos de datos con 324 de ancho, 301 de alto y 56 de profundidad que se obtiene de la reconstrucción de una langosta. Para este conjunto de datos usamos un umbral  $\tau = 26$  seleccionado manualmente. En la figura 3.1 es interesante notar que en las zonas donde la superficie es bastante uniforme el método de ponderación de normales no suaviza tanto como el MC.

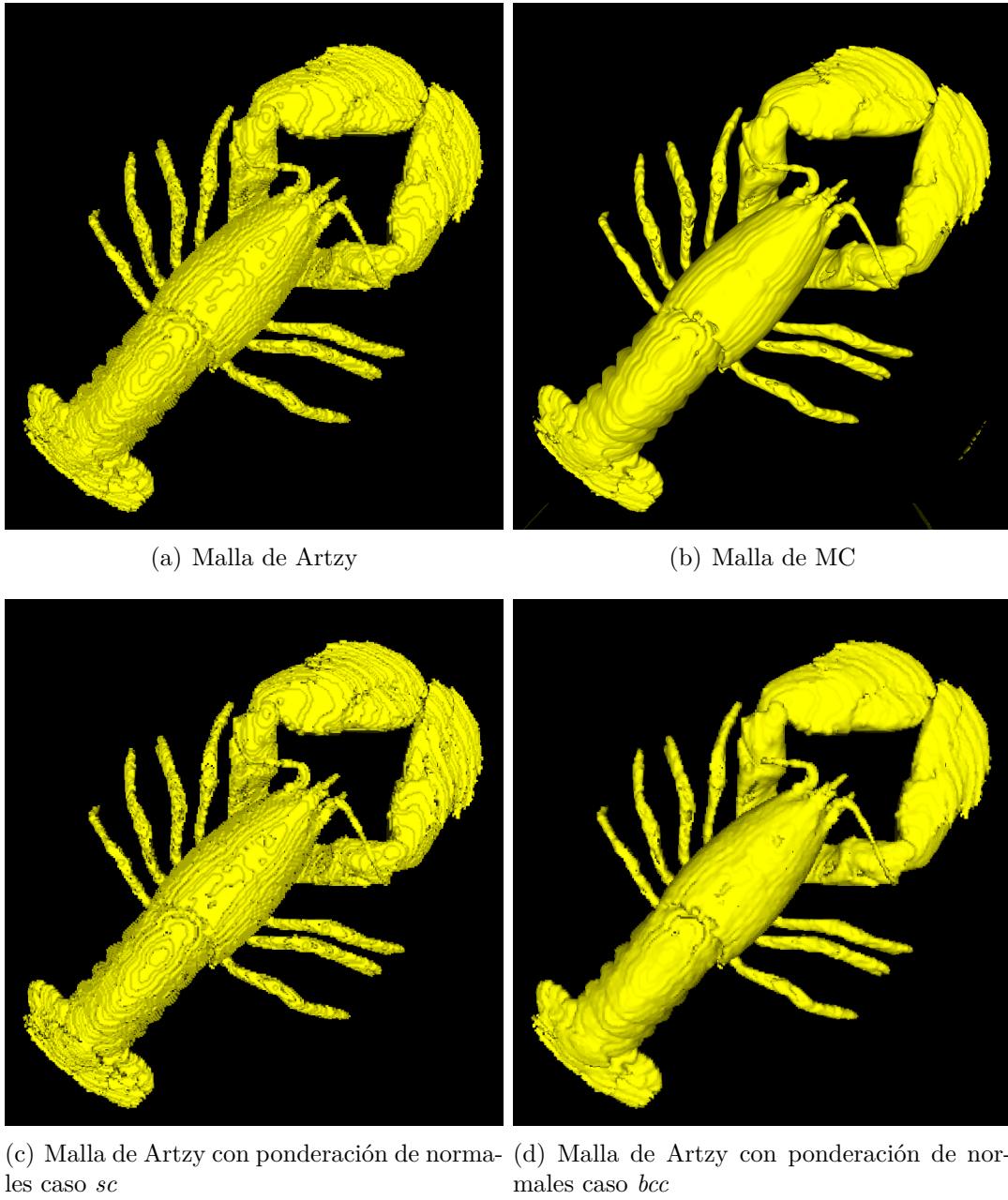
En la Figura 3.2 el conjunto de imágenes muestra una vista diferente de la misma superficie de la Figura 3.1. En esta vista queda de manifiesto que el método no suaviza demasiado en zonas donde la topología del modelo es compleja, es decir hace un buen trabajo pues no pierde detalles finos de la malla.

El siguiente experimento visual fue hecho sobre un conjunto de datos de reconstrucción de una cabeza humana, con tamaño 256 en cada dimensión, en estos datos se seleccionó un umbral  $\tau = 103$  que permite ver muchas zonas relativamente homogéneas. Se pueden ver los resultados en la Figura 3.3.

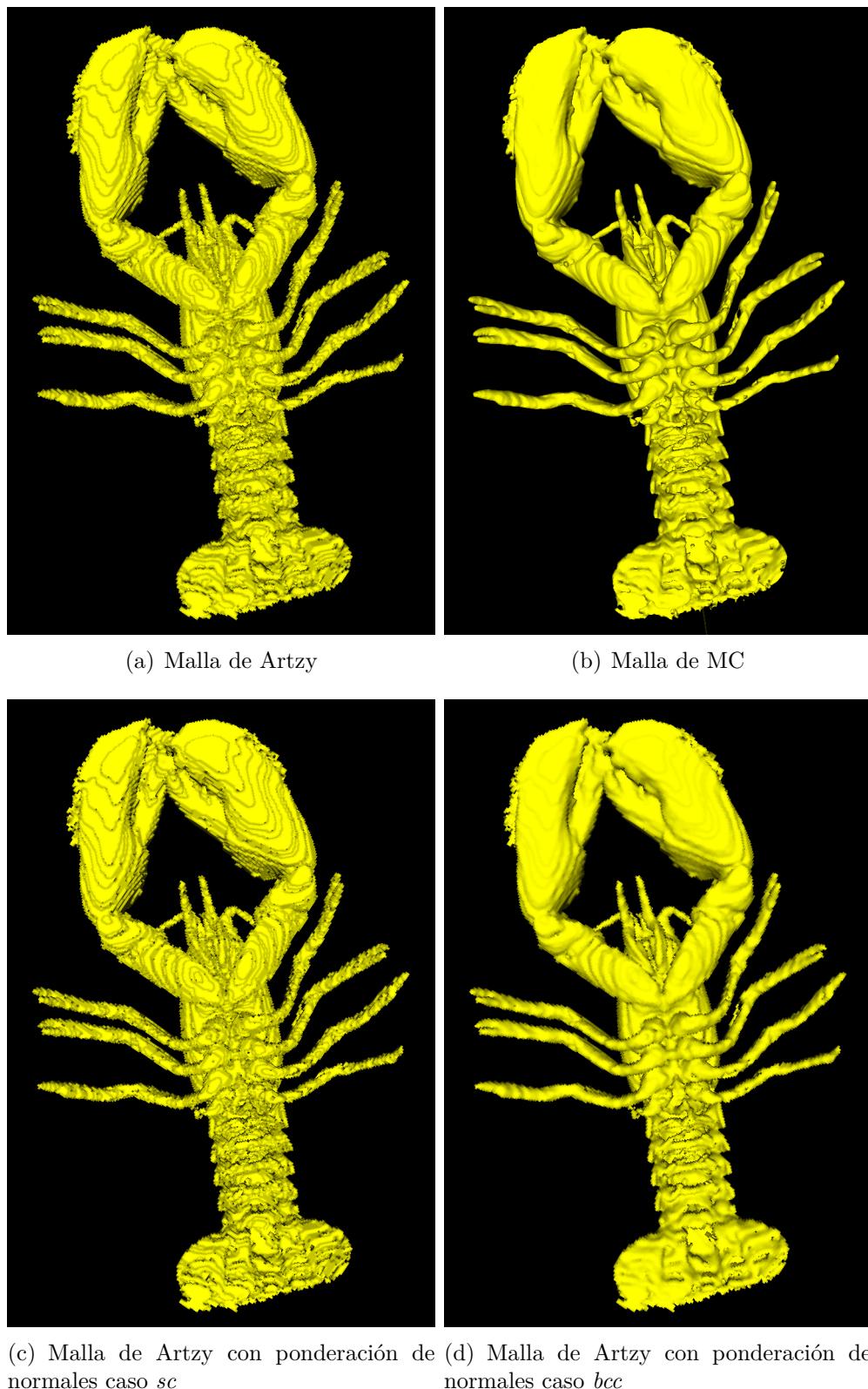
De esta figura es importante resaltar que aun cuando las mallas de Artzy (Figura 3.3(a)) y MC (Figura 3.3(b)) producen resultados parecidos en las zonas de la nariz. Una vez que se hace el posprocesado, la misma zona se ve mucho mas suave en la Figura 3.3(d).

### 3.2 Experimentos con maniquíes (*phantoms*)

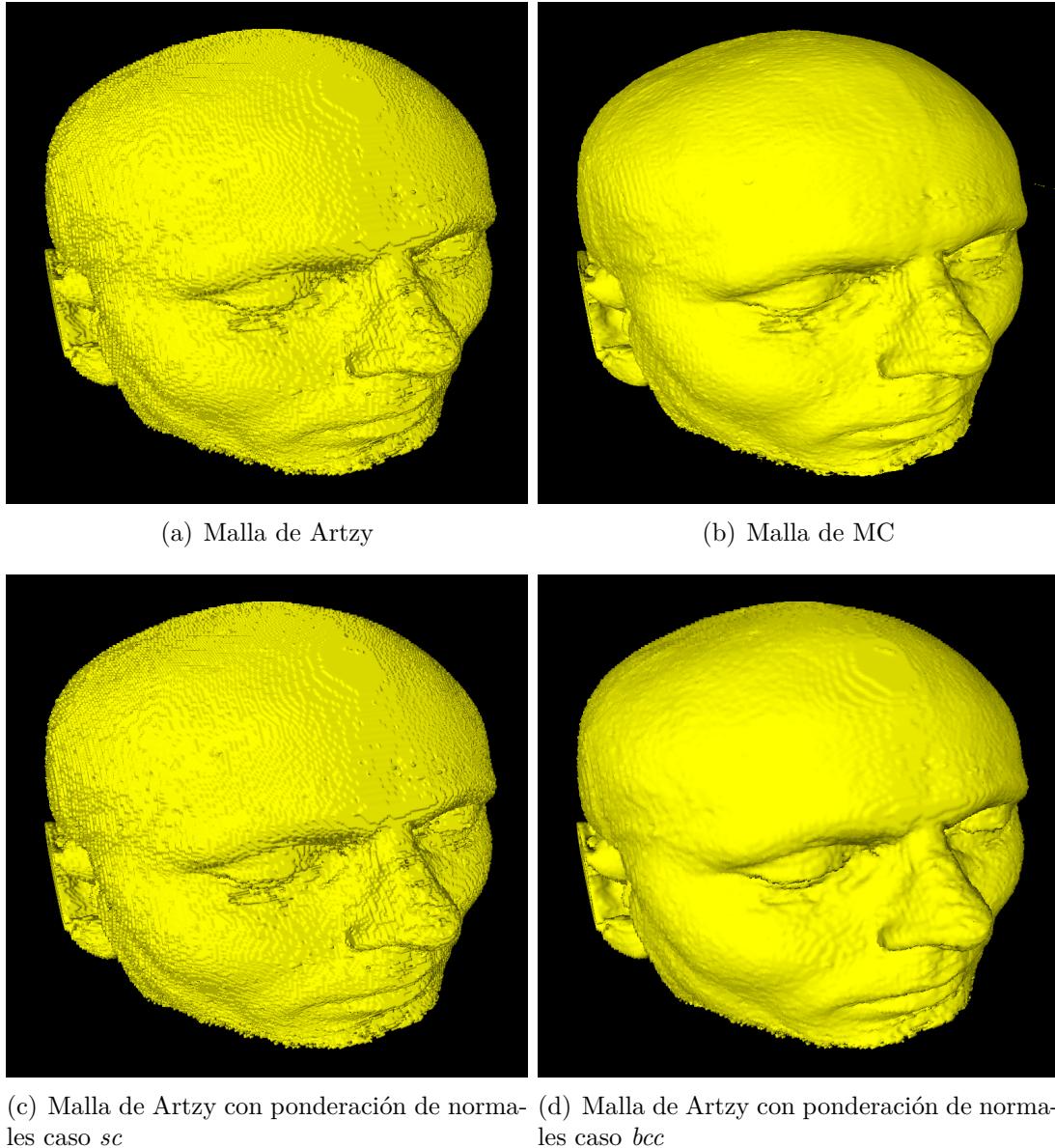
En los experimentos realizados en la sección anterior no se conoce la función original y por lo tanto es difícil realizar una comparación numérica, es por esta razón que diseñamos otros experimentos usando una función conocida la cual fue muestreada “adecuadamente” para generar la aproximación  $v_{G_\Delta}$ .



**Figura 3.1:** Comparación visual de superficies formadas a partir de la reconstrucción de una langosta.



**Figura 3.2:** Comparación visual de superficies de la reconstrucción de una langosta. Son las mismas mallas que las de la Figura 3.1 vistas desde otro punto de vista.



**Figura 3.3:** Comparación visual en el modelo de una cabeza humana.

Para estos experimentos deseamos comparar nuestra aproximación con la “realidad”. Como las superficies son diferentes y solo modificamos las normales, realizamos la comparación de las normales del objeto real con las normales de nuestra aproximación.

### 3.2.1 Medida de error

Una medida de error de las normales puestas sobre la malla y las normales analíticas es el error medio de los ángulos entre esos vectores. Definido de la siguiente manera:

$$ME = \sum_{i=1}^m \frac{|\arccos \langle \mathbf{n}_j, \mathbf{n}_j^* \rangle|}{m}. \quad (3.1)$$

Con respecto a la ecuación (3.1) cabe señalar que los vectores para iluminación son unitarios, por lo que  $|\mathbf{n}_j| = |\mathbf{n}_j^*| = 1$  y por eso podemos calcular al ángulo por medio del producto punto.

En donde  $\mathbf{n}_j$  es la normal analítica de la superficie  $S_\tau$  y  $\mathbf{n}_j^*$  es la normal asignada en ese punto de la malla por el método de ponderado de normales.

### 3.2.2 Criterio de muestro

Para este análisis se usa como base una esfera unitaria que forma un campo escalar de densidad uniforme y también unitaria, es decir:

$$s(\mathbf{x}) = \begin{cases} 1, & \text{si } |\mathbf{x}| \leq 1, \\ 0, & \text{en otro caso.} \end{cases} \quad (3.2)$$

Se sabe que la transformada de Fourier de esa función esta dada por la siguiente expresión [44]

$$\hat{s}(\boldsymbol{\xi}) = \frac{J_1(2\pi|\boldsymbol{\xi}|)}{\boldsymbol{\xi}}. \quad (3.3)$$

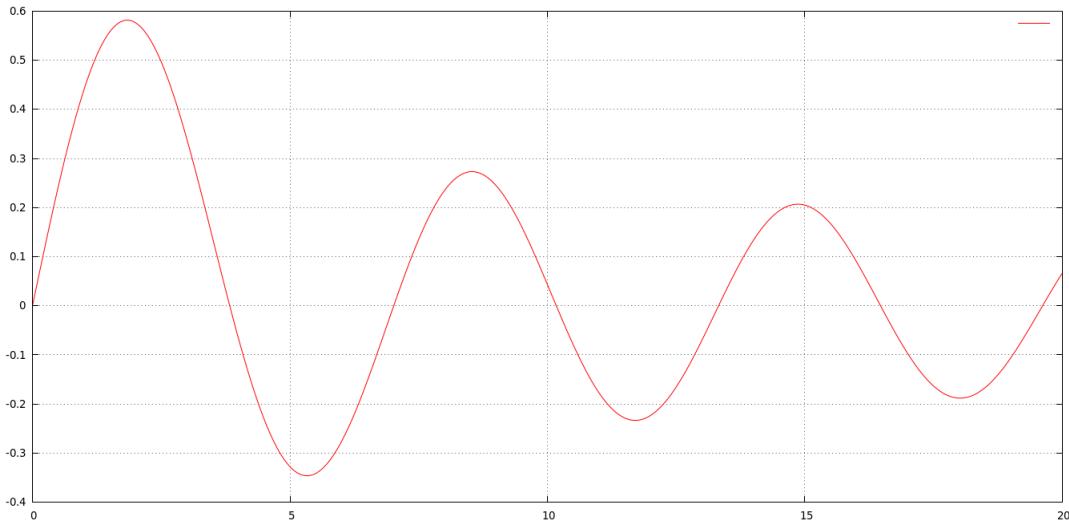
Sabemos también de [44] que como la función es radialmente simétrica y su transformada también lo es podemos hacer el resto del análisis en una sola dimensión. Una forma de encontrar el valor de discretización es encontrar primero una frecuencia de corte de

la transformada  $\hat{f}$ .

Un criterio razonable es cortar en los ceros de  $\hat{s}$ . Esta función es cero cuando el numerador es cero, por lo tanto podemos buscar los valores  $u$  para los cuales

$$\begin{aligned} J_1(2\pi\xi) &= 0, \\ 2\pi\xi &= u, \\ \xi &= \frac{u}{2\pi}. \end{aligned} \tag{3.4}$$

la función  $J_1$  tiene varios valores  $u$  tales que  $J_1(u) = 0$ , su gráfica puede verse en la Figura 3.4. Numéricamente se determinaron los primeros cinco valores  $u$  para los que  $J_1(u) = 0$ . Luego para cada uno de ellos se determinó un valor en el espacio de Fourier con (3.4) que se tomó como frecuencia de corte.



**Figura 3.4:** Gráfica de la función  $J_1$  en la frecuencia.

Una vez calculada la frecuencia de corte  $\xi$ , el ancho de banda es  $2\xi$ , como en el espacio de Fourier las magnitudes son inversas al espacio real entonces  $\frac{1}{2\xi}$  es la distancia de muestreo  $\Delta$  para cada caso.

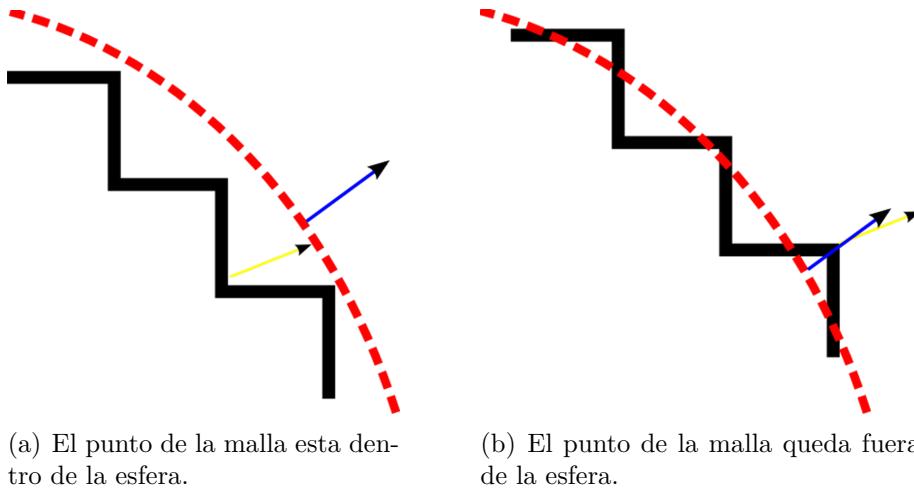
La reconstrucción fue muestreada a cada una de estas discretizaciones. Finalmente, a cada superficie se le aplica el ponderado de normales y se mide su error medio.

A cada una de las normales asignadas  $\mathbf{n}_j$  se mide su error con respecto a la normal en la

Valor de $u$	Frecuencia de corte $\xi$	Distancia de muestreo $\Delta$	Número de véxeles
3.83	0.61	32.79	4
7.01	1.12	17.91	7
10.17	1.62	12.35	10
13.32	2.12	9.43	14
16.47	2.62	7.62	17

**Cuadro 3.2:** En esta tabla se muestran las discretizaciones correspondientes para distintos ceros de la función  $J_1(\xi)$ .

superficie de la esfera. Sin embargo, debido a que  $\mathcal{A}_\tau$  es una aproximación a  $S_\tau$  es muy posible que los vértices de la malla  $\mathcal{A}_\tau$  no estén sobre la superficie analítica de la esfera. Para poder hacer la comparación si un vértice  $\mathbf{p}$  de la malla  $\mathcal{A}_\tau$  no está en la superficie es proyectado a un punto de la superficie y luego se mide el error de su normal con respecto a la del punto donde fue proyectado, esto se ve en la Figura 3.5. Puede ser que tenga que ser proyectado hacia afuera si originalmente estaba dentro de la superficie, Figura 3.5(a), o que tenga que ser proyectado hacia adentro si originalmente estaba fuera de la superficie, como en la Figura 3.5(b).



**Figura 3.5:** Esquema para la comparación de normales en la superficie de la esfera. La malla se muestra en negro, la superficie de la esfera en rojo con líneas punteadas. La normal asignada con la ponderación es el vector amarillo, la normal contra la cual se compara es el vector en azul.

Los resultados del experimento se resumen en la Tabla 3.3. En la segunda columna se muestra el ángulo de mayor error medido en radianes. La tercera columna tiene el error

medio medido en grados.

Discretización	Ángulo de mayor error	ME en grados
4	0.00049	0.10
7	0.02561	0.75
10	0.06333	2.23
14	0.18293	4.43
17	0.18766	4.03

**Cuadro 3.3:** Resultados del experimento numérico sobre una esfera.

De la tabla anterior podemos apreciar que en el error es muy pequeño y parece aumentar conforme la discretización de la esfera aumenta hasta un cierto umbral en donde empieza a disminuir.

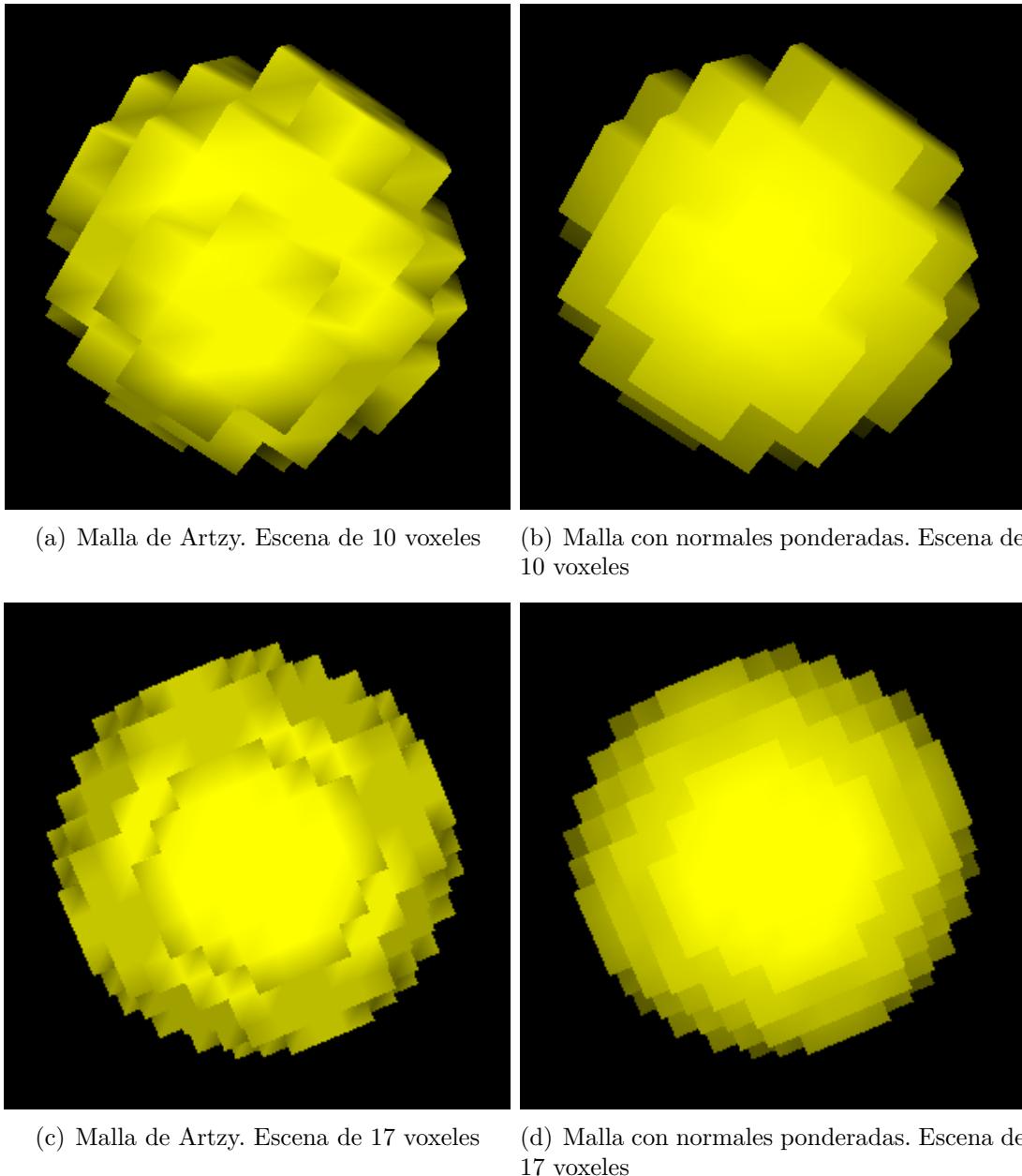
Aunque el experimento es numérico se consideró prudente, en el caso de la esfera, mostrar la imagen de dos realizaciones del experimento. En la Figura 3.6 se muestra en el lado izquierdo la malla de Arzty y en el lado derecho la malla con el ponderado de normales. En las dos primeras imágenes se muestra para una escena de 10 véxeles de tamaño y en el segundo renglón para una escena de 17 véxeles.

Como los blobs son funciones esféricas se espera que se adapten mejor a la superficie de una esfera. Para poder hacer otra comparación se creo también un *phantom* de un cubo cuyo lado mide lo mismo que el diámetro de la esfera en una escena del mismo tamaño. Para poder comparar el cubo con la esfera se realizaron las mismas discretizaciones.

Hay también que aclarar que la superficie  $\mathcal{A}_r$  encontrada por el algoritmo de Artzy coincide totalmente con la superficie del phantom del cubo. Por esta razón las normales de los vértices de la malla se comparan con las normales analíticas en el mismo punto. Los resultados se pueden ver en la Tabla 3.4.

Discretización	Ángulo de mayor error	ME en grados
4	0.433081	13.51
7	0.433081	11.88
10	0.433081	8.18
14	0.433081	5.72
17	0.433081	5.32

**Cuadro 3.4:** Resultados del experimento numérico sobre un cubo.



**Figura 3.6:** Resultados visuales de la ponderación de normales en el caso de la esfera.

En este experimento llama la atención que el blob parece no adaptarse tan finamente como en el caso de la esfera pues el error promedio es considerablemente mayor que en el modelo anterior. También es de llamar la atención que el caso de mayor error se mantiene constante.



# Conclusiones

Los resultados visuales obtenidos en la Sección 3.1 nos indican que se pueden alcanzar mejoras manipulando solamente los parámetros de la iluminación y que esto puede hacerse con una técnica como la presentada en este trabajo, sin conocer ninguna información *a priori* del conjunto de datos que queremos visualizar.

Aunque los resultados son buenos con el blob utilizado para la rejilla *bcc*, este blob es ideal para un modelo promedio. Con esto queremos decir que hay modelos para los que este blob es bueno, por ejemplo el de la Figura 3.3, sin embargo no existe un blob ideal para todos los modelos. Una posible linea de investigación es buscar una manera de optimizar el blob para un modelo en particular conociendo solamente la malla  $\mathcal{A}_\tau$  que aproxima la superficie.

Un punto importante es también que la calidad de la malla producida por el Algoritmo de Artzy es mejor que la del algoritmo estándar de MC. Los experimentos nos demostraron que en general el número de caras de la malla de Artzy es menor que la producida por MC además de tener la característica de que sus vértices tienen grados más homogéneos lo cual es una ventaja desde el punto de vista computacional, pues en general se ocupa menos espacio para guardar las mallas de Artzy.

Otra posible linea de investigación sería utilizar el Algoritmo de Artzy cuando el volumen esta muestreado en la rejilla *bcc*. En [19] se hace esta implementación que da como resultado vóxeles con forma de dodecaedro rómbico. Estos vóxeles son mas eficientes al llenar el espacio, y tiene la ventaja de tener caras en forma de rombos (y en doce orientaciones distintas). Por lo que se sospecha que usar en esta malla la aproximación aquí descrita mejoraría mucho su visualización.

La malla de Artzy como estructura discreta nos es mas útil que la de MC, pues podemos hacer operaciones como medir el volumen interior de la superficie o el área de la malla

envolvente con mucha mas facilidad que en la malla de MC. Esto es debido a que su geometría es mas simple y sobre todo a que por ser topológicamente correcta no da lugar a ambigüedades (en todo momento sabemos que hay un conjunto de véxeles interiores y otro conjunto de véxeles exteriores). Por lo tanto podemos concluir que vale la pena seguir trabajando con el Algoritmo de Artzy aun cuando MC sea el algoritmo de rastreo de superficies mas usado actualmente.

También cabe señalar que dado que asumimos total desconocimiento del origen del conjunto de datos “cualquier” criterio que usemos para adaptar mejor una superficie implícita es valido. Por esta razón en futuros experimentos se deberían explorar nuevos criterios para optimizar los blobs, particularmente se piensa que criterios puramente geométricos (sin usar técnicas de procesamiento de imágenes) podrían dar buenos resultados.

Hay que señalar que los resultados son puramente visuales y altamente dependientes del modelo de iluminación. Aunque en este trabajo nos limitamos al modelo de iluminación de Phong y al sombreado de Gouraud, bien valdría la pena explorar modelos alternativos tanto para iluminación como para sombreado.

También como trabajo futuro se podría intentar que el GPU se hiciera cargo no solo de los cálculos de iluminación si no también de los cálculos de extracción de superficie, es decir que el Algoritmo de Artzy debería de poderse implementar en GPU.

La elección del blob Kaiser-Bessel para la construcción de la superficie implícita fue en gran medida al conocimiento previo del grupo de trabajo y a que ha dado buenos resultados para fines de visualización. Sin embargo, este blob es relativamente complejo de evaluar. Por esto no se descarta la posibilidad de usar algún otro blob y que pueda dar resultados parecidos y con mucho menos costo computacional.

Tradicionalmente se han usado técnicas de procesamiento de imágenes para mejorar la visualización de imágenes en biomedicina. Los resultados de éste trabajo demuestran que es posible mejorar la visualización de una imagen 3D usando técnicas de iluminación. Por lo tanto, debe de seguirse investigando como mejorar las imágenes no solo con procesamiento de imágenes, si no también por técnicas de graficación por computadora.

# Bibliografía

- [1] M. Alexa y col. «Computing and Rendering Point set surfaces». En: *IEEE Transactions on Visualization and Computer Graphics* 9.1 (ene. de 2003), págs. 3-15.
- [2] Ehud Artzy, Gideon Frieder y Gabor T. Herman. «The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm». En: *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. Seattle, Washington, United States: ACM, 1980, págs. 2-9. ISBN: 0-89791-021-4. DOI: <http://doi.acm.org/10.1145/800250.807461>.
- [3] Ehud Artzy y Gabor T. Herman. «Boundary detection in 3-dimensions with a medical application». En: *SIGGRAPH Computer Graphics* 15.1 (1981), págs. 92-123. ISSN: 0097-8930. DOI: <http://doi.acm.org/10.1145/988460.988465>.
- [4] Mike Baley y Steve Cunningham. *Graphics Shaders Theory and Practice*. Inglés. Massachusetts: A K Petters, Ltd, 2009.
- [5] James F. Blinn. «A Generalization of Algebraic Surface Drawing». En: *ACM Transactions on Graphics* 1 (3 jul. de 1982), págs. 235-256.
- [6] James F. Blinn. «Simulation of wrinkled surfaces». En: *SIGGRAPH Computer Graphics* 12 (3 ago. de 1978), págs. 286-292.
- [7] J. Bloomenthal. «Skeletal Design of Natural Forms.» Tesis doct. University of Calgary, 1995.
- [8] J. E. Bresenham. «Algorithm for computer control of a digital plotter». En: *IBM Systems Journal* 4.1 (1965), págs. 25-30.
- [9] Samuel R. Buss. *3-D Computer Graphics: a mathematical introduction with OpenGL*. Inglés. New York: Cambridge, 2003.

- [10] Cinthya Ceja Mendoza y Beatriz Pimentel Caro. «Visualización de isosuperficies usando RayCasting y modelo Blobby por GPU». Tesis de licenciatura de Ingeniería en Sistemas Computacionales, Instituto Tecnológico de Morelia.
- [11] S.Y. Chen y col. «Improvement on dynamic elastic interpolation technique for reconstructing 3-D objects from serial cross sections [biomedical application]». En: *IEEE Transactions on Medical Imaging* 9.1 (mar. de 1990), págs. 71-83. ISSN: 0278-0062.
- [12] Wei Chen y Gabor T. Herman. «Efficient controls for finitely convergent sequential algorithms». En: *ACM Transactions on Mathematical Software* 37.2 (2010), págs. 1-23. ISSN: 0098-3500. DOI: <http://doi.acm.org/10.1145/1731022.1731024>.
- [13] Robert A. Drebin, Loren Carpenter y Pat Hanrahan. «Volume rendering». En: *SIGGRAPH Computer Graphics* 22.4 (1988), págs. 65-74. ISSN: 0097-8930. DOI: <http://doi.acm.org/10.1145/378456.378484>.
- [14] Eric Galin y Samir Akkouche. «Incremental Polygonization of Implicit Surfaces». En: *Graphical Models* 62.1 (2000), págs. 19-39.
- [15] Edgar Garduño. «Visualization and extraction of structural components from reconstructed volumes.» Tesis doct. University of Pennsylvania, 2002.
- [16] Edgar Garduño y Gabor T. Herman. «Implicit surface visualization of reconstructed biological molecules». En: *Theoretical Computer Science* 346.2 (2005), págs. 281-299. ISSN: 0304-3975. DOI: <http://dx.doi.org/10.1016/j.tcs.2005.08.027>.
- [17] Edgar Garduño y Gabor T. Herman. «Optimization of basis functions for both reconstruction and visualization». En: *Discrete Applied Mathematics Journal* 139.1-3 (2004), págs. 95-111. ISSN: 0166-218X. DOI: <http://dx.doi.org/10.1016/j.dam.2002.12.002>.
- [18] Edgar Garduño, Gabor T. Herman y R Davidi. «Reconstruction from a few projections by  $\ell_1$ -minimization of the Haar transform». En: *Inverse Problems* 27.5 (2011).
- [19] Edgar Garduño, Gabor T. Herman y H. Katz. «Boundary tracking in 3D binary images to produce rhombic faces for a dodecahedral model». En: *IEEE Transactions on Medical Imaging* 17.6 (1998), págs. 1097-1100.

- [20] Frieder Gideon y col. «Large Software Problems for Small Computers: An Example from Medical Processing». En: *IEEE Software* 2.5 (1985), págs. 37-47.
- [21] Rafael C. Gonzalez y Richard E. Woods. *Digital Image Procesing*. Inglés. 2<sup>a</sup> ed. Prentice Hall, 2002.
- [22] H. Gouraud. «Continuous Shading of Curved Surfaces». En: *IEEE Transactions on Computers* C-20.6 (1971), págs. 623-629.
- [23] H. Hagen, H. Miller y G. M. Nielson. *Focus on Scientific Visualization*. Inglés. 1<sup>a</sup> ed. Berlin: Springer-Verlag, 1993.
- [24] Hans Hagen y col. «Scientific Visualization: Methods and Applications». En: *Proceedings of the 19th spring conference on Computer graphics*. Budmerice, Slovakia: ACM, 2003, págs. 23-33.
- [25] Gabor T Herman. *Fundamentals of Computerized Tomography*. Inglés. 2<sup>a</sup> ed. London: Springer, 2009.
- [26] Gabor T Herman. *Geometry of Digital Spaces*. Inglés. 1<sup>a</sup> ed. Boston: Birkhäuser, Springer, 1996.
- [27] A. Hilton y col. «Marching triangles: Range Image Fusion for Complex Object Modelling». En: *Proceedings of the International Conference on Image Processing*. Vol. 2. Sep. de 1996, págs. 381-384.
- [28] L. Ibanez, C. Hamitouche y C. Roux. «Ray casting in the BCC grid applied to 3D medical image visualization». En: *Engineering in Medicine and Biology Society. Proceedings of the 20th Annual International Conference of the IEEE* 2 (1998), págs. 548-551.
- [29] D. Kalra y A. H. Barr. «Guaranteed ray intersections with implicit surfaces». En: *SIGGRAPH Computer Graphics* 23 (3 jul. de 1989), págs. 297-306. ISSN: 0097-8930.
- [30] Tomomichi Kaneko y col. «Detailed shape representation with parallax mapping». En: *Proceedings of the ICAT 2001*. 2001, págs. 205-208.
- [31] E. Keppel. «Approximating Complex Surfaces by Triangulation of Contour Lines». En: *IBM Journal of Research and Development* 19.1 (ene. de 1975), págs. 2-11. ISSN: 0018-8646.
- [32] R M Lewitt. «Alternatives to voxels for image representation in iterative reconstruction algorithms». En: *Physics in Medicine and Biology* 37.3 (1992), pág. 705.

- [33] Robert M. Lewitt. «Multidimensional digital image representations using generalized Kaiser-Bessel window functions». En: *Journal of the Optical Society of America* 7.10 (1990), págs. 1834-1846.
- [34] William E. Lorensen y Harvey E. Cline. «Marching cubes: A high resolution 3D surface construction algorithm». En: *SIGGRAPH Computer Graphics* 21.4 (1987), págs. 163-169. ISSN: 0097-8930. DOI: <http://doi.acm.org/10.1145/37402.37422>.
- [35] S. Matej y R.M. Lewitt. «Efficient 3D grids for image reconstruction using spherically-symmetric volume elements». En: *Nuclear Science Symposium and Medical Imaging Conference* 3 (1994), págs. 1177-1181.
- [36] S. Matej y R.M. Lewitt. «Practical considerations for 3-D image reconstruction using spherically symmetric volume elements». En: *IEEE Transactions on Medical Imaging* 15.1 (feb. de 1996), págs. 68-78.
- [37] Shigeru Muraki. «Volumetric shape description of range data using "Blobby Model"». En: *SIGGRAPH Computer Graphics* 25 (4 jul. de 1991), págs. 227-235.
- [38] Timothy S. Newman y Hong Yi. «A survey of the marching cubes algorithm». En: *Computers and Graphics* 30.5 (2006), págs. 854-879.
- [39] H. Nishimura, M. Hirai y T. Kawai. «Object modeling by distribution function and a method of image generation.» En: *Transactions of IECE of Japan* J68-D (1985), págs. 718-725.
- [40] Daniel P. Petersen y David Middleton. «Sampling and reconstruction of wave-number-limited functions in N-dimensional euclidean spaces». En: *Information and Control* 5.4 (1962), págs. 279-323. ISSN: 0019-9958.
- [41] Bui Tuong Phong. «Illumination of Computer-Generated Images». Tesis doct. University of Utah, 1973.
- [42] Stefan Roettger. *The Volume Library*. <http://www9.informatik.uni-erlangen.de/External/vollib/>. Mayo de 2011.
- [43] Dave Shreiner y col. *OpenGL Programming Guide*. Inglés. 4<sup>a</sup> ed. Boston: Addison-Wesley, 2004.
- [44] Elias M. Stein y Guido Weiss. *Introduction to fourier analysis on euclidean spaces*. Inglés. 1<sup>a</sup> ed. Princeton: Princeton University Press, 1971.

- [45] The Khronos Group. *The Industry Standard for High Performance Graphics.* <http://www.opengl.org/>. Mayo de 2011.
- [46] U.S. National Library of Medicine. *The Visible Human Project.* [http://www.nlm.nih.gov/research/visible/visible\\_human.html](http://www.nlm.nih.gov/research/visible/visible_human.html). Mayo de 2011.
- [47] Carmen Villar. *Apuntes de Graficación por Computadora usando OpenGL.* UNAM, 2008.
- [48] Wikipedia, the free encyclopedia. *Anatomical terms of location.* [http://en.wikipedia.org/wiki/Anatomical\\_terms\\_of\\_location](http://en.wikipedia.org/wiki/Anatomical_terms_of_location). Mayo de 2011.
- [49] Lance Williams. «Pyramidal parametrics». En: *SIGGRAPH Computer Graphics* 17 (3 jul. de 1983), págs. 1-11. ISSN: 0097-8930.
- [50] G. Wyvill, C. McPheeers y B. Wyvill. «Data structure for soft objects». En: *The Visual Computer* 2 (1986), págs. 227-234.
- [51] Xi Yongjian y Duan Ye. «A novel region-growing based iso-surface extraction algorithm». En: *Computers and Graphics* 32.6 (2008), págs. 647-654. ISSN: 0097-8493.