UNIVERSIDAD DE GRANADA

Dpto. Lenguajes y Sistemas Informáticos

Visualización expresiva de volúmenes

MEMORIA QUE PRESENTA

Germán Arroyo Moreno

PARA OPTAR AL GRADO DE DOCTOR

Marzo de 2006

DIRECTORES

Dr. D. Francisco Velasco Anguita Dr. D. Domingo Martín Perandrés

Editor: Editorial de la Universidad de Granada
Autor: Germán Arroyo Moreno
D.L.: Gr. 900 - 2006
ISBN: 84-338-3875-x

La memoria titulada “Visualización Expresiva de Volúmenes”, que presenta Germán Arroyo Moreno para optar al grado de doctor ha sido realizada dentro del programa de doctorado “Métodos y Técnicas Avanzadas de Desarrollo de Software”del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada. Ha sido dirigida por los doctores D. Francisco Velasco Anguita y D. Domingo Martín Perandrés, del Departamento de Lenguajes y Sistemas Informáticos.

Granada, Marzo de 2006

Doctorando Director Director

Fdo.: Germán Arroyo Moreno Fdo.: Francisco Velasco Anguita Fdo.: Domingo Martín Perandrés

Agradecimientos

A mis padres, Pilar y Pedro, sin ellos este trabajo nunca habría comenzado, a ellos les debo lo que soy.

A Roberto por estar siempre ahí cuando lo he necesitado, su apoyo y el tiempo que siempre me ha dedicado.

A Beatriz por su incondicional apoyo y paciencia.

A mis directores Francisco y Domingo, por su excelente dirección, sus útiles consejos y todo el tiempo que me han dedicado.

A José Miguel y Juan Carlos por sus útiles comentarios, en especial para el último capítulo.

A Antonio, Betty, Francis, Gloria, Marc, Mario, Mayte, Richard y Rodrigo, por todos los buenos momentos pasados juntos, y ser los mejores amigos que nadie pudiera tener.

En general a todos los que de una forma u otra me han ayudado en la realización de este trabajo.

El trabajo presentado en esta memoria ha sido parcialmente subvencionado por el Ministerio de Ciencia y Tecnología y fondos FEDER a través del proyecto de investigación TIN2004-06326-C03-02.

Índice

Índice de figuras

Índice de tablas

Prólogo

El tema principal de este trabajo se puede encuadrar entre la visualización directa de volúmenes, la visualización expresiva y el análisis de imágenes. Es un trabajo multidisciplinar que se encuadra perfectamente en el área de la informática gráfica, cubriendo la necesidad de un subárea donde el trabajo con datos científicos pueda ser procesado para que el sistema perceptual humano pueda trabajar de forma más sencilla con estos datos.

Introducción

Estas nuevas representaciones de los datos científicos tienen múltiples aplicaciones en campos como la medicina, la psicología, la docencia, la arqueología, etc, donde los datos pueden ser obtenidos de forma sencilla mediante TAC (tomografía axial computerizada), RM (resonancia magnética) y otros métodos de adquisición de datos muy comunes entre los científicos de estas áreas.

Motivación

Queremos destacar en este apartado, el porqué de la decisión tomada para trabajar en este subárea como puede ser la "visualización expresiva de volúmenes".

Desde el comienzo del estudio de los diferentes trabajos en análisis de imágenes y en el análisis y visualización de datos volumétricos, podemos notar que el trabajo desarrollado en la visualización directa de volúmenes dista mucho de obtener una gran calidad, y cabe plantearnos el porqué.

Un volumen no es más que un concepto de algo que no pueden verse en la realidad de forma clara y concisa ya que no podemos ver en el interior de los objetos, y aun realizando un corte a un modelo físico real, la heterogeneidad del modelo no se refleja en la parte visualizada, si no en sus propiedades, la mayoría de las veces, físicas.

De hecho, cabe preguntarse por qué, si para visualizar un modelo que en la realidad no es fácilmente distinguible, usamos un modelo de visualización e iluminación basado en un modelo físico real. Es ahí donde entramos en el campo de la visualización expresiva, la cual no usa un modelo realista sino que obtiene imágenes comprensibles para el ser humano.

Sin embargo, no queríamos enfocar el problema de la visualización expresiva en este tipo de datos desde un punto de vista artístico aunque se hayan realizado algunos trabajos que podrían entrar dentro de este apartado o subárea. El fin de este trabajo ha sido generar imágenes destinada a científicos. Por ello podemos subdividir todo el trabajo en tres tipos diferentes de visualización expresiva que se han tocado con mayor o menor intensidad según el fin último que hemos ido persiguiendo:

- Imágenes para diagnóstico: con las técnicas expresivas podemos realzar un modelo, sin necesidad de un alto coste en el análisis de los datos que lo componen.
- Imágenes artísticas: aunque no nos hemos centrado demasiado en el tema, hemos mostrado algunos ejemplos de obtención de imágenes artísticas a partir de un modelo vo-

lumétrico.

- Ilustración científica: hemos generado imágenes expresivas entre el arte y el diagnóstico para generar imágenes que pueden ser usadas para explicar conceptos o para trabajar con científicos que no son expertos en tratar con los datos originales.

Es por ello que una parte imprescindible en la base de todo este trabajo es la psicología en el campo de la percepción visual humana y el análisis de imágenes (ya que el volumen puede discretizarse como una serie de imágenes paralelas). Debido a que debemos conocer qué realzar y cómo realzarlo.

Objetivos

El objetivo principal que se persigue es el de apoyar a los científicos a visualizar de forma más sencilla el volumen de un caso real. Además de mostrar como trabajar de forma unificada con volúmenes o en general con cualquier modelo puntual definido como un subconjunto de los volúmenes.

De forma particular los objetivos que se han ido persiguiendo han sido:

- Obtener un modelo de iluminación no realista que mejore al modelo realista.
- Obtener un modelo de iluminación no realista que sea capaz de producir ilustraciones parecidas a la de los libros de temática científica.
- Obtener un método de obtención y estilización de siluetas.
- Obtener un método de análisis que se pueda aplicar a cualquier modelo puntual.
- Obtener un modelo basado en luces no necesariamente realista (luces virtuales) unificado para trabajar de forma sencilla con volúmenes.
- Desarrollar un sistema de composición de luces virtuales y un modelo de trabajo válido y sencillo para la visualización directa expresiva de volúmenes.

Estructura de la memoria

El orden de cada capítulo de esta memoria ha sido cuidadosamente elegido para que el lector no se pierda entre las diversas áreas que toca este trabajo (debido al campo multidisciplinar en el que se encuentra).

Se tiene en mente para el orden elegido que el volumen puede analizarse desde el punto de vista de la iluminación (no solo como aplicación de funciones de transferencia), que un volumen no es más que un conjunto de datos en el espacio, y que como tal, también una imagen es un volumen aunque bidimensional. También se tiene en cuenta que un conjunto de imágenes a lo largo del tiempo puede ser considerado un volumen (ya que pueden tratarse como consecutivas en el espacio y luego trasladarlas al dominio del tiempo). Por último se tiene en mente que todos estos trabajos pueden unificarse en un único concepto que es la transformación de datos volumétricos (o datos puntuales), en imágenes mediante unas nuevas luces virtuales, que definimos en esta memoria.

El primer capítulo realiza una introducción más o menos básica sobre la informática gráfica, centrándose cada vez más en la visualización científica de volúmenes.

En el segundo capítulo se revisan conceptos previos necesarios para un mejor seguimiento de los siguientes capítulos. Se divide en tres secciones más o menos alejadas de la temática de la visualización de volúmenes clásica:

- Visión: en esta sección se repasan los conceptos que serán necesarios para orientar el trabajo a obtener las imágenes más fácilmente perceptibles por el ser humano.
- Análisis de imágenes: en esta sección se repasan los conceptos necesarios para comprender operaciones matemáticas que se usarán para obtener información de la imagen.
- Visualización expresiva: en esta sección se repasan los conceptos necesarios para comprender las operaciones expresivas realizadas para la simulación de los distintos métodos que los artistas suelen utilizar y el porqué de su uso.

Los siguientes capítulos tienen todos aportaciones originales (todos los trabajos han sido parcialmente publicados, excepto el último de los capítulos).

En un tercer capítulo se describe un nuevo método de iluminación basado en el sistema de color HSV y en un sistema de clasificación asistido por el usuario.

En el cuarto capítulo se describe un nuevo método de detección de siluetas sobre una visualización basada en trazado de rayos y una mejora al mismo, siendo ambos independientes del sistema de visualización empleado.

En el quinto capítulo se describe un nuevo método para la ilustración a partir de la definición de luz virtual clásica.

En el sexto capítulo se describe un nuevo método para la creación de imágenes o vídeos artísticos a partir del análisis de imágenes y vídeo.

En el séptimo capítulo es un capítulo de unificación de todo el trabajo realizado, y por tanto de los capítulos anteriores. Se propone un nuevo método de trabajo basado en la composición de funciones y se realiza la definición del concepto de **luces virtuales volumétricas** que permiten modelar imágenes expresivas a partir de modelos volumétricos de forma muy sencilla. Además se crea el concepto de función volumétrica que extiende a las funciones de transferencia que eran utilizadas de forma clásica en la visualización de volúmenes.

En resumen, el trabajo permite crear una nueva subárea para el tratamiento no realista de datos científicos, mediante el uso de unos cauces compuestos de luces virtuales volumétricas y funciones volumétricas, basadas en el análisis de imágenes y en el sistema visual humano.

Capítulo 1

Introducción

En este capítulo se realizará una revisión de la informática gráfica hasta llegar a los conceptos de visualización de datos científicos, y en concreto, a la visualización directa de volúmenes.

Introducción

La informática gráfica trata de visualizar datos geométricos mediante un dispositivo electrónico, normalmente bidimensional (como puede ser una pantalla). Sin embargo, estos datos geométricos suelen ser siempre tridimensionales, y suelen representarse de distintas formas según la aplicación que se realice. En este trabajo nos referiremos a los datos geométricos como modelo tridimensional, ya que consideraremos que los datos se encuentran siempre en un espacio de tres dimensiones y no en un plano.

1.1. Representaciones de objetos tridimensionales

Los modelos tridimensionales son representados en el computador mediante la asimilación algorítmica de una abstracción matemática, de forma que internamente se representan una serie de datos discretos, que están relacionados entre sí mediante una formulación matemática.

De esta forma, un modelo tridimensional es una estructura de datos almacenada en la máquina junto con un software (algoritmos) que permiten que el usuario interactúe con dicha información.

Estos modelos se pueden clasificar dependiendo de esta formulación matemática y conceptual, de tal forma que tenemos principalmente tres tipos de modelos tridimensionales:

- Modelo geométrico
- Modelo sólido
- Modelo volumétrico

En un modelo geométrico se suele representar principalmente la información geométrica del modelo tridimensional. Esta representación es sencilla, pero por contra no podemos determinar información muy útil como puede ser si un punto está o no en el interior del mismo, y por ello es muy difícil realizar operaciones con él.

En un modelo sólido lo que se almacena es información topológica, de tal forma que se introduce información adicional que permite determinar las propiedades de los objetos de forma concisa y eficiente.

Sin embargo en este tipo de modelos se presupone un interior homogéneo que no suele ser válido en muchas aplicaciones científicas. En este caso hay que representar el modelo

tridimensional mediante un modelo volumétrico, en el que también se almacenan datos del interior del objeto.

Según la topología empleada en el modelo abstracto podemos diferenciar entre:

- Modelos basados en topología puntual
- Modelos basados en topología algebraica

1.1.1. Modelos basados en topología puntual

Según este enfoque un sólido es un subconjunto *S* cerrado y acotado de puntos en el espacio euclídeo $\mathbb{R}^3$ [70].

Al subconjunto *S* se le exigen las siguientes condiciones:

- Rigidez: Un sólido no debe deformarse tras una transformación rígida (translación o rotación), por ello debe cumplirse que todos los subconjuntos de puntos que pueden obtenerse mediante transformaciones rígidas de S representan al mismo sólido que S.
- Regularidad: Los sólidos no poseen planos, líneas o puntos aislados. *S* debe ser regular, es decir, debe ser igual al conjunto resultante de calcular la clausura al interior de *S*. Un conjunto regular y acotado se denomina *r-set*.
- Representación finita: Todo aquello que se desee almacenar en un computador debe tener una representación finita.

Este tipo de modelos se suele representar mediante un *buffer* 3D. El sólido se discretiza en una rejilla para su manipulación y visualización. Todos los objetos se transforman a un metaobjeto (unidades de esa rejilla). Por tanto pueden almacenar información heterogénea del interior haciéndolos más apropiados para la representación de volúmenes. Entre sus características más relevantes nos encontramos [46]:

- Insensibilidad a la complejidad de la escena una vez que los objetos se han convertido a la rejilla. Este preproceso puede verse influido por la complejidad de las escenas, pero la manipulación y la visualización de la misma no.
- Insensibilidad a la complejidad de los objetos, ya que se desarrolla sólo una vez durante el proceso de discretización quedando almacenada dicha información en un metaobjeto.
- Independencia del punto de vista, ya que los metaobjetos almacenan todos los atributos de visualización.
- La información del interior se representa fácilmente.
- Sobre el *buffer* 3D se pueden hacer operaciones de bloques fácilmente así como realizar cambios de resolución en la rejilla de modo jerárquico.
- La información proveniente de muestreo y simulaciones puede ser almacenada directamente en un *buffer* volumétrico.

Aunque también presentan desventajas. En concreto, las necesidades de memoria son altas y necesita una gran potencia de cálculo, que se ve aliviada con el desarrollo del hardware específico. Al tener una resolución finita en el modelo también se tiene problemas en el cálculo de propiedades y la precisión de los cálculos.

1.1.2. Modelos basados en topología algebraica

Otra alternativa es representar un sólido mediante la superficie que lo delimita del exterior [70]. Este modelo matemático es posible en tanto en cuanto el interior de un sólido es homogéneo y no se almacena información sobre dicho interior.

Haciendo uso de esta alternativa se representa un objeto 3D mediante un modelo 2D. Las condiciones que tiene que cumplir una superficie para ser sólido son:

- La superficie debe ser cerrada y completa.
- La superficie no debe intersecar consigo misma.
- La superficie debe delimitar el interior y el exterior unívocamente.

Una superficie que cumple estas condiciones se denomina *frontera* del sólido. Formalmente se define como el espacio topológico *2-variedad* (*topologic 2-variety space*).

Los modelos basados en frontera suelen implementarse en el computador mediante poliedros. Entre sus ventajas [46] tenemos que el espacio que se requiere para su almacenamiento es mínimo y el tiempo de transformaciones de carácter geométrico es un proceso fácil y rápido proyectando los poliedros a un dispositivo de salida 2D. Sin embargo, no posibilitan el almacenar directamente información sobre su interior.

1.1.3. Introducción a los métodos de subdivisión espacial

Debido a la gran cantidad de datos que se suelen manejar cuando se representa un modelo tridimensional basado en topología algebraica, lo usual es utilizar un algoritmo que determine que partes del modelo son visibles y que partes son ocultas para el observador. De tal forma que solamente se tratan computacionalmente aquellas que el usuario está observando en ese momento.

Existen varias técnicas de ordenación espacial que describimos a continuación:

- Bintrees: esta representación parte de un cubo inicial que engloba todo el mundo representado, y se va dividiendo en dos mitades iguales sucesivamente siguiendo los ejes principales de forma cíclica. Se marca como negro (o relleno) la celda que contiene sólido completo, y blanco (o vacío) aquella celda que no contiene nada del sólido. Aquellas celdas intermedias, en las que parte contienen al modelo y parte no, se subdividen. El proceso se repite hasta que todas las celdas son negras o blancas [85], o hasta un límite de subdivisiones establecido previamente.
- BSP: (*Binary space partition*) Esta representación es similar a la anterior salvo que en esta ocasión la partición se hace dependiendo del sólido, con el objeto de reducir el número de particiones [93]. La idea es que los planos de división vayan coincidiendo con los planos del sólido.
- Octrees: Se parte de un cubo inicial, al igual que en los *bintrees*. Y se va dividiendo recursivamente en ocho subcubos hijos de igual tamaño [83, 84, 99]. A este esquema básico se han añadido otros que mejoran la eficiencia de esta subdivisión, como son los *face octrees*, los *face-and-edge octrees*, los *extended octrees*, o los SP-Octrees, en los cuales se almacena información adicional en alguno de los nodos (tal y como puede ser planos, aristas, o vértices) [13, 74, 16, 73]. Existen otras aproximaciones en las cuales

la subdivisión no se realiza mediante cubos, si no que se realizan en otro sistema de coordenadas que no son el cartesiano [95, 17].

Todas estas técnicas, aunque en principio utilizadas para la representación de sólidos, pueden adaptarse fácilmente para la visualización de volúmenes.

1.2. Visualización de datos científicos

La visualización científica comprende el campo de la informática que trata de estudiar y definir algoritmos y estructuras de datos para la visualización de datos científicos. El objetivo principal de este campo es la obtención de imágenes comprensibles por el ser humano. La aplicación de este campo es muy amplia, pudiendo tener un ámbito entre la geología, la dinámica de fluidos, ciencias medioambientales o medicina, entre otras muchas.

En nuestro caso concreto, los datos científicos tendrán una heterogeneidad, tanto en el interior como en el exterior, del modelo analizado. De forma que hablamos siempre de una visualización de volúmenes [12].

Podemos, por tanto, analizar el cauce general de procesamiento (*pipeline*), según el cual se suele visualizar un volumen, de forma comprensible por el sistema de visión humano en forma de imagen.

Figura 1.1: Cauce de procesamiento general para la representación de datos científicos

Veremos cada uno de los pasos del cauce de la figura 1.1 con mayor detenimiento en las siguientes secciones.

1.2.1. Extracción de datos volumétricos

El primer paso para la visualización debe ser siempre la adquisición de los datos a representar, básicamente existen tres formas de obtención:

- Muestreo: en este caso el volumen es obtenido mediante el muestreo o la discretización de un modelo físico real o fenómeno natural. Existen varias técnicas, tales como tomografía computerizada (TAC) o resonancia magnética (RM), que obtienen una serie de imágenes a partir de un eje tridimensional. Estas imágenes suelen venir dadas en unos valores de grises (no necesariamente entre 0 y 255), en la cual se representan distintas funcionalidades de la anatomía o fisiología del cuerpo humano o de objetos o fenómenos naturales de otro tipo. No suele ser común la obtención de datos en color, y en estos casos, el color no tiene ninguna relación con el color real del interior del modelo real. Además, la gran mayoría de las veces una visualización directa del modelo real hace que la parte observada sea imperceptible para el ser humano debido a la similitud de los tonos visualizados y a la cantidad de datos ligeramente diferenciados de su alrededor, además, muchas veces los datos son invisibles para el ser humano (la luminosidad en los rayos X por ejemplo), y al traspasarlos al computador en tonos grises se pierde información valiosa.

- Simulación: En este caso, el volumen es producido mediante una simulación por computador. La simulación de fenómenos meteorológicos o el flujo de un cauce de agua simulado mediante dinámica de fluidos son ejemplos de este método de obtención.

- Técnicas de modelado: en este caso los datos volumétricos se generan mediante un modelo geométrico, mediante una herramienta de diseño asistido (CAD). En el caso de este tipo de datos, no suele ser común este método de adquisición por la dificultad que entraña el crear modelos heterogéneos en su interior.

1.2.2. Datos volumétricos

Una vez extraidos los datos hace falta reconstruirlos y almacenarlos en el computador de algún modo. La estructura computacional de un conjunto de datos volumétricos pueden ser representados en forma de aproximaciones discretas de un campo escalar, vectorial o mediante tensores. La representación más común es mediante un conjunto S de muestras (x,y,z,w), el valor w representa alguna propiedad en la localización (x,y,z), de tal forma que $S \subset \mathbb{R}^3 \times \Gamma$ siendo Γ el conjunto de todas las posibles propiedades de un punto.

A cada uno de los datos de la cuaterna (x,y,z,w) se le denomina *voxel* y es el equivalente tridimensional a un *pixel*.

Los datos volumétricos podrían caracterizarse según su topología o en términos de su geometría como:

- Rectilíneos: las muestras se distribuyen en una retícula regular, la cual divide el dominio en células de idéntico tamaño.

- Curvilíneas: las muestras caen en una serie de retículas definidas en un espacio computacional. Y es definida como una función que define una determinada posición en el espacio.

- No estructurada o irregular: tal y como indica su nombre, los datos no tienen una estructura regular. Las muestras son dadas como una lista de localizaciones espaciales con una serie de campos adicionales. La conexión entre cada una de las celdas o *vóxeles* es especificada de forma explícita y las celdas pueden tener una forma arbitraria.

Puesto que los datos pueden llegar en muchas formas y tamaños diferentes, debe haber un modelo de representación que aporte restricciones al modelo y al sistema de visualización del mismo. La definición de este *modelo de representación*, en el caso de los datos curvilíneos, viene dado directamente por la estructura geométrica definida implícitamente.

El esquema de representación más ampliamente usado es el *modelo de enumeración espacial* o el *modelo de vóxeles* [46]. El *modelo de vóxeles* se basa en la descomposición regular del espacio tridimensional en un conjunto de células cúbicas idénticas, cada una de ellas conocida como *voxel* . Sus bordes son paralelos a los ejes de coordenadas. Esta estructura espacial permite que un *voxel* se pueda representar como un vector (i,j,k) donde $1 \leq i,j,k \leq n$, siendo n el número de *vóxeles* por eje desde el cual puede ser obtenida toda la información geométrica y topológica. Cada *voxel* tiene asociado una serie de valores. Podemos tomar dos aproximaciones según la localización de estos valores:

- La *aproximación por celdas* representa muestras en las ocho esquinas de los vértices de cada *voxel*. El interior es estimado mediante la interpolación de los valores de sus vértices. Hay muchas funciones posibles de interpolación. Una de la más común es la

función conocida como *interpolación de primer orden* o *interpolación trilineal*, que será comentada en profundidad en posteriores capítulos.

- La *aproximación por voxel* asigna una muestra al centro del *voxel* . El interior del *voxel* es considerado homogéneo.

Puesto que los modelos que se han utilizado en nuestras pruebas son obtenidos a partir de instrumentación médica, no existe rejillas curvilíneas, sino que lo común son rejillas rectilíneas, son por ello usados cortes con una distribución regular. Es por ello que la definición del *modelo de vóxeles* a partir de estos datos no representa ninguna dificultad, ya que es suficiente alinear cada uno de los cortes del modelo a la retícula regular.

1.2.3. Filtrado

Una vez que los datos a visualizar son conocidos, el siguiente paso es tratar los datos. El proceso de filtrado es un pre-proceso a los datos del volumen para extraer información relevante del mismo. Dicha información no está explícitamente reflejada en los datos que obtuvimos en el paso anterior. De esta forma, el proceso de filtrado refleja las operaciones propias de corrección de la interpolación de datos erróneos, eliminación de ruido de la imagen, clasificación, etc.

En las aplicaciones médicas, por ejemplo, las dos operaciones más utilizadas suelen ser:

- Remuestreo y re-estructuración: Cada fabricante y cada maquinaria en particular tiene una estructura nativa para las imágenes obtenidas en cada uno de los cortes y unas características propias que inducen un ruido concreto. Es necesario, por ello, el remuestreo y la re-estructuración de los datos para definir el esquema de representación. Esta operación es específica tanto para un conjunto particular de datos como para el hardware usado para generar la información.

- Segmentación y clasificación: Los valores de un conjunto de datos se suelen usar para representar distintos elementos o propiedades del mismo. Por ello, la segmentación y la clasificación nos ayudan a distinguir mejor cada uno de estos elementos. La segmentación se define como la división de una imagen en regiones coherentes, usando algún criterio sintáctico (características locales de la imagen). La clasificación es, sin embargo, una etiquetación a cada región y se basa en distintos criterios tales como: múltiples umbrales, geometría de la imagen, datos estadísticos, etc. La clasificación es muy utilizada en *funciones de transferencia*, que comentaremos detalladamente en posteriores capítulos.

1.2.4. Mapeado

El paso de mapeado transforma datos geométricos del modelo, tales como pueden ser las distancias, en elementos concretos de visualización. Estos elementos concretos de visualización son atributos gráficos que se pueden representar, tales como color o transparencia. El mapeado es el proceso principal en el cauce de procesamiento de la visualización por cuanto decide qué primitivas geométricas deberían generarse y qué propiedades debería asignarse a sus atributos. En general, la representación geométrica seleccionada de un objeto tridimensional que puede ser visualizado mediante una pantalla bidimensional.

Es importante realizar una visualización fiel de los datos pero también el realizarla de forma rápida, por ello el hardware gráfico representa una fuerte restricción a cómo ha de realizarse dicha visualización.

Han sido desarrollados muchos algoritmos para mapear modelos volumétricos, sin embargo todas pueden ser clasificadas según tres grupos principales:

- Rebanadas (*slicing*): mediante esta técnica lo que se realiza es una simplificación del problema, reduciendo las dimensiones a dos. De esta forma se obtiene una rebanada bidimensional (normalmente paralela a uno de los ejes principales) a partir de los datos obtenidos. Después se usa alguna técnica de coloreado, realce de bordes o similar. Con esta técnica solamente se muestra una pequeña porción de todos los datos que teníamos.

- Ajuste mediante superficies: esta técnica genera una aproximación por polígonos a partir de una *isosuperficie*. La superficie aproxima un subconjunto del volumen dado una propiedad determinada, conocida como *isovalor*. Esta es una buena aproximación para objetos que están claramente delimitados mediante bordes, como huesos en TAC, pero no es adecuado para objetos amorfos que tienen una expresión matemática difícil de representar mediante superficies. Además necesita de varios pasos de extracción si se quiere visualizar al mismo tiempo partes interiores y exteriores del modelo. La extracción de superficies es un paso ampliamente utilizado, existen dos tipos básicos: algoritmos 2D y algoritmos 3D:

  - Los primeros algoritmos siguen una aproximación 2D [19, 31, 20]. Los volúmenes tridimensionales se visualizan como una serie consecutiva de rebanadas 2D. Estos métodos parten de dos fases principales: la extracción de los *isocontornos* 2D cercanos en cada rebanada del volumen. El principal problema de este método es que la mayoría de las veces se necesita un gran proceso manual de ajuste. Además no es siempre posible especificar de forma automática como emparejar los contornos en las rebanadas.

  - En la aproximación 3D la superficie se reconstruye directamente. No hay un paso previo de extracción de contornos. Según la terminología de Kalvin [44] estos algoritmos pueden ser caracterizados por los elementos fronterizos usados para reconstruir las *isosuperficies* como: *algoritmos formados por bloques* (*block-form*), cuando el elemento frontera es una cara, y *algoritmos basados en emparejamiento de cubos* (*marching cubes*) cuando el elemento frontera es un vértice. Los *algoritmos formado por bloques* consideran los *vóxeles* como cubos homogéneos y genera superficies conectando las caras incompletas por *vóxeles* en lados opuestos del *isovalor*. Como todos los parches son ortogonales a uno de los ejes cardinales, la imagen final tiene un aspecto de bloques [98]. Los algoritmos basados en el *emparejamiento de cubos* aproximan la superficie de cada celda por polígonos cuyos vértices caen entre puntos adyacentes de la muestra original con valores en lados opuestos del *isovalor*. Aunque el coste de procesamiento y el espacio en memoria es mayor que los métodos por cubos, estos métodos ofrecen una aproximación más cercana a la *isosuperficie* que ajustan [56].

- Visualización directa: cuando los datos volumétricos son modelados mediante una técnica de extracción de superficies se pierde información. En muchas aplicaciones, tales como la medicina o geología, es interesante que la información sea fidedigna y

que no haya errores. Es por ello que la técnica de visualización por superficies no es válida en estos casos. En la técnica de visualización directa el volumen es considerado como si de un gel translúcido se tratara, la imagen es obtenida directamente asignando color y opacidad a cada uno de los valores de la muestra original y aplicando un modelo de iluminación o filtros para determinar la apariencia final del modelo. Existen dos métodos que clásicamente se han utilizado para este tipo de representaciones: el método de ordenación por la imagen[54] y el método ordenado por el objeto[101, 78].

- En el método ordenado por la imagen existe una única técnica con múltiples variantes que ha sido utilizado incluso en los trabajos más recientes[23]. Se trata del trazado de rayos (*ray casting*) que veremos en profundidad en la siguiente sección.
- En el método ordenado por objeto existen varias técnicas. La primera de ellas por antigüedad es la técnica conocida como *splatting*[101, 69, 52] mediante el cual se proyectan celdas sobre el plano de la imagen obteniendo unas manchas llamadas *splats* que se combinan para obtener la imagen final. La otra técnica utilizada más recientemente es la visualización mediante texturas 3D o 2D [32]. Básicamente consiste en discretizar el volumen en una serie de texturas 3D o texturas 2D, posteriormente, se dibujan una serie de planos paralelos según un eje dado (normalmente el mismo vector que el del observador, o bien el eje principal más próximo al del observador), tras una serie de planos consecutivos con su textura correspondiente tenemos un resultado final que da la apariencia de tridimensionalidad. Veremos esta técnica con detenimiento en el siguiente apartado.

1.3. Visualización directa de volúmenes

En este apartado nos centraremos principalmente en la visualización directa y en las diferentes técnicas que actualmente son utilizadas para este tipo de representaciones.

1.3.1. Trazado de rayos

En la visualización directa el volumen es visualizado mediante la superposición de capas translúcidas. El paso fundamental en este tipo de visualización es el asignar un material a cada valor correspondiente, además, se suele calcular una normal asociada para la iluminación como hemos comentado previamente.

La base de la mayoría de las técnicas para el proceso de dibujado de volúmenes es la integral de visualizado de volumen (*volume rendering integral*), en su forma difusa. Esta integral es una simplificación de la teoría de la física del transporte de la luz en el caso de omisión de dispersión y efectos de la frecuencia [43].

El volumen es visualizado como un conjunto de partículas con cierta densidad μ, puesto que la luz viaja en línea recta, podemos considerar que la cantidad de luz en un rayo para la longitud de onda λ viene dada por

$$I_\lambda = \int_0^L C_\lambda s\mu(s) e^{-\int_0^s \mu(t)dt} ds \tag{1.1}$$

donde

- L es la longitud del rayo.
- $C_\lambda s$ es la luz, de longitud de onda λ, reflejada en la muestra s en la dirección del rayo. El cálculo de $C_\lambda(s)$ puede estar basado en el modelo estándar de reflexión de Phong.
- μ es el coeficiente de atenuación, el cual define la cantidad de luz que se pierde por unidad de longitud, hasta que llega a extinguirse totalmente. La medida dada por $\mu(s)$ indica la densidad del material en el punto s.

La integral acumula la intensidad a lo largo de toda la longitud del rayo, atenuándose de acuerdo a la densidad del material a través del cual pasa. Esta atenuación viene representada por el término de la exponencial [65].

Para poder computar esta integral en un tiempo eficiente debemos evaluarla numéricamente. Si n es el número de pasos a lo largo del rayo cuyos valores de muestra han sido tomados, usando una aproximación por sumatorias simples de Riemman tenemos:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i\Delta s)\mu(i\Delta s)\Delta s \prod_{j=0}^{i-1} exp(-\mu(j\Delta s)\Delta s)$$

Reemplazando la exponencial de esta sumatoria por los dos primeros términos de su expansión de Taylor:

$$exp(-\mu(j\Delta s)\Delta s) = 1 - \mu(j\Delta s)\Delta s$$

y definiendo la transparencia $t(j\Delta s)$ como:

$$t(j\Delta s) = exp(-\mu(j\Delta s)\Delta s)$$

de igual forma:

$$\mu(i\Delta s)\Delta s = 1 - t(i\Delta s) = \alpha(i\Delta s)$$

por tanto, también se da:

$$t(j\Delta s) = 1 - \alpha(j\Delta s)$$

donde $\alpha = 1 - t$ es la opacidad. Esta aproximación convierte la ecuación anterior en la fórmula que suele ser usada para la visualización de volúmenes:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i\Delta s)\alpha(i\Delta s) \prod_{j=0}^{i-1} (1 - \alpha(j\Delta s)) \tag{1.2}$$

Los valores C y α son conocidos únicamente en los datos del modelo. Para calcular los valores en la muestra dada por $i\Delta s$ se aplica un proceso de interpolación.

Si el volumen es isotrópico y tienen como distancia la unidad entre dato y dato, la ecuación anterior se simplifica, quedando la siguiente expresión:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i)\alpha(i) \prod_{j=0}^{i-1} (1 - \alpha(j)) \tag{1.3}$$

En la práctica no se suele utilizar una longitud λ cualquiera, sino que se aplica solamente a los valores R, G y B en un espacio de color RGB [29], y se realiza de forma independiente para cada canal, siendo el resultado final la suma de intensidades de muestras individuales.

La computación puede realizarse de forma recursiva procesando una vez cada muestra acumulando el color y la opacidad de forma separada:

$$C_{out} = C_{in} + (1 - \alpha_{in})\alpha_i C_i \tag{1.4}$$

para cada muestra i, y

$$\alpha_{out} = \alpha_{in} + (1 - \alpha_{in})\alpha_i C_i \tag{1.5}$$

En la técnica de trazado de rayos se genera la imagen píxel a *pixel*, lanzando un rayo por cada píxel de la pantalla e intersecando dicho píxel sobre las partes del volumen. Este tipo de métodos son una implementación sencilla de la visualización propio volumen. La proyección de la imagen generada se obtiene evaluando la integral en forma de sumatoria discretizada. La composición de los valores muestreados a lo largo del rayo pueden tratarse desde atrás hacia delante, o en orden inverso [54]. En este último caso se puede llegar a realizar muy rápidamente los cálculos de transparencias, además se puede dejar de calcular los valores que se encuentran no visibles debido a la alta opacidad de los ya calculados hasta el momento. La principal desventaja de los métodos de trazado de rayos es el alto coste computacional asociado al producto del número de rayos y muestras por rayo. Aunque bien es cierto que existen numerosas mejoras al rendimiento en esta técnica, tales como el uso de planos de corte para acelerar los cálculos en las partes no relevantes del modelo [61] o el uso de ordenación espacial mediante *octrees* del modelo [102, 50], e incluso con arquitecturas de propósito específico [77].

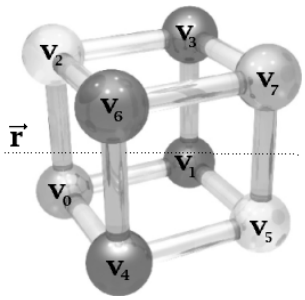

Figura 1.2: Este dibujo representa ocho datos vecinos que pueden ser intersecados por un rayo $\vec{r}$ cualquiera.

Normalmente se suele calcular el material (y en general cualquier propiedad asociada) entre ocho datos vecinos utilizando una interpolación trilineal. Si consideramos los ocho vecinos situados como en la figura 1.2, y con propiedades determinadas por la función f, la expresión resultante es:

$$F(x,y,z) = a + bx + cy + dz + eyz + gxz + hxy + ixyz \tag{1.6}$$

donde

$$
\begin{aligned}
a &= f(v_0) \\
b &= f(v_1) - f(v_0) \\
c &= f(v_2) - f(v_0) \\
d &= f(v_4) - f(v_0) \\
e &= f(v_6) - f(v_4) - f(v_2) + f(v_0) \\
g &= f(v_5) - f(v_4) - f(v_1) + f(v_0) \\
h &= f(v_3) - f(v_2) - f(v_1) + f(v_0) \\
i &= f(v_7) - f(v_6) - f(v_5) - f(v_3) + f(v_4) + f(v_2) + f(v_1) - f(v_0)
\end{aligned}
$$

y (x,y,z) es el punto dado por la intersección de $\vec{r}$ con el tetraedro de la figura 1.2, con $x,y,z \in [0,1]$.

En la segunda aproximación, llamada en muchas ocasiones proceso de visualizado (*rendering*) hacia delante, el proceso de visualizado consiste en ajustar cada *voxel* uno sobre otro para formar la imagen final. De esta forma un simple *voxel* contribuye a varios píxeles. El método clásico es la técnica del *splatting*, que esencialmente consiste en proyectar cada *voxel* en el *buffer* de imagen de atrás a delante o al contrario. Esta técnica reordena la integral de visualizado del volumen. El punto débil que ha tenido esta técnica en los últimos años ha sido que las imágenes obtenidas suelen ser bastante borrosas. El algoritmo original calculaba la iluminación antes de interpolar, lo que producía este efecto de suavizado. Mueller y otros [69] reordenan la computación de la iluminación apareciendo sus imágenes más claras y los bordes más marcados. Los diferentes métodos que han sido propuestos difieren básicamente en la complejidad computacional y en la calidad del proceso de proyección de los *vóxeles*.

Los algoritmos híbridos combinan las ventajas de ambas aproximaciones, un ejemplo de estas aproximaciones es el algoritmo propuesto por Lacroute [52], el cual trata de orientar los datos para que puedan ser dibujados más rápidamente.

1.3.2. Visualización basada en texturas 3D

Mención aparte requiere la técnica del visualizado de modelos volumétricos mediante texturas 3D. En los últimos tiempos el hardware a avanzado en una dirección muy favorable al mapeado 3D de texturas. Creciendo el número de tarjetas gráficas que implementan algún método de texturizado de este tipo[47].

Esta técnica consiste, fundamentalmente, en dibujar un conjunto de datos volumétricos mediante la composición desde atrás hacia delante de un conjunto de planos que muestrean dicho volumen, el cual se encuentra cargado por completo en la memoria de textura de un subsistema gráfico. Los planos semitransparentes resultantes son mezclados para obtener la imagen final (tal y como muestra la figura 1.3. El tiempo de visualización de la imagen es mínimo comparado con otras técnicas aproximadas mediante software[66, 79].

La visualización de los polígonos se suele hacer sin introducir ningún cambio por perspectiva debido principalmente a que al ser polígonos los que se texturizan al girarlo no se obtiene apreciación de tridimensionalidad, sino que los polígonos son claramente visibles y el modelo queda reducido a rodajas con imágenes. Es por ello que la texturización 3D puede efectuarse de dos formas:

- **Simulando una proyección ortogonal**: es la más sencilla además de la más usada. Los polígonos son paralelos y son equidistantes unos de otros. Además son paralelos al *viewport*. Es equivalente a realizar un trazado de rayos bajo una proyección ortogonal[101].

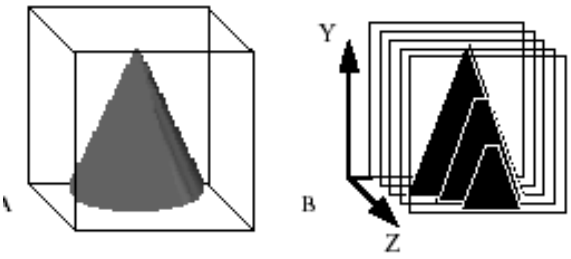

Figura 1.3: Este dibujo representa como se visualiza un volumen (a la izquierda) mediante polígonos texturizados (derecha).

- **Simulando una proyección en perspectiva**: se utiliza menos que la anterior, consiste en muestrear una serie de semiesferas a partir de la posición del observador, el resultado simula a un trazado de rayos pero con perspectiva. De forma idealizada las superficies que muestrean el volumen son normales al rayo de visión, lo cual implica que la superficie debería ser paralela a las "cáscaras" esféricas. Para visualizaciones a largas distancias las "cáscaras" esféricas pueden ser aproximadas suficientemente por planos paralelos perpendiculares a la dirección del observador[67, 100].

El principal defecto de esta técnica viene dado por la limitación de las implementaciones y de las tarjetas gráficas actuales (normalmente suelen soportar texturas de 64x64x64, y excepcionalmente de 256x256x256). Sin embargo el punto fuerte es que las nuevas tarjetas están implementando nuevos algoritmos basados en hardware que potencian el uso de esta técnica (como el motor NVIDIA nfiniteFX Engine que viene incorporado a partir de los modelos de GForce3 de NVidia) [2, 1].

Algunas mejoras han sido introducidas desde la aparición del algoritmo básico, los cambios más importantes son los siguientes:

- La forma más común de representar la información del volumen es un *texel* (que es un valor, normalmente de color, de una textura) por *voxel* (que es cada uno de los pequeños cubos que conforman el volumen). Esta estrategia garantiza el máximo de detalle y calidad de la imagen resultante, sin embargo, el tamaño de la memoria suele hacer imposible el almacenar todo el modelo en ella. Generalmente se utiliza una aproximación basada en trocear la textura original en bloques (*bricks*). Un bloque corresponde a un subconjunto de los datos del volumen, por definición más pequeño que la textura original. Cada bloque se computa de forma independiente, desafortunadamente, la carga y descarga de estos bloques en la memoria de texturas de las tarjetas gráficas degrada drásticamente el número de imágenes por segundo [51]. El proceso de descomposición en bloques se puede mejorar según los siguientes criterios:

  - **Partición de datos:** el volumen puede ser descompuesto en el número mínimo de bloques de máximo tamaño. Este criterio reduce los elementos redundantes que han sido almacenados entre bloques adyacentes para preservar la continuidad.
  - **Representación de *voxels* no vacíos:** la descomposición de bloques trata de minimizar el porcentaje de *voxels* vacíos representados en la memoria de textura. Las regiones vacías son detectadas en una fase de pre-procesamiento.

  - **Representación de multiresolución:** la descomposición de bloques no es arbitraria sino que se basa en grados de interés, distancia al observador, etc.

- Cuando la complejidad de los datos es demasiado elevada se suele utilizar un *octree* u otro tipo de ordenación espacial sobre los bloques, de tal forma que existe una ordenación jerárquica entre ellos y se puede ahorrar mucha memoria. El problema principal de esta técnica es la pérdida de rendimiento.

- Normalmente las texturas suelen componerse con valores RGBα. Si se ilumina de alguna forma se mejora la visualización de la textura, dicha iluminación se realiza en un pre-procesado, lo cual mejora el rendimiento de la visualización del modelo (ya que no se tiene que recalcular la fuente de luz).

Como en los casos anteriores, existen arquitecturas diseñadas para mejorar el rendimiento de este tipo de visualización, como muestra el artículo de Kanus [45], que presenta una arquitectura de memoria basada en caché para gráficos volumétricos, y más concretamente en *voxel*.

Mucho se ha discutido sobre la velocidad de esta técnica con el clásico trazado de rayos y, aunque toda la arquitectura actual está pensada para mejorar las texturas 3D no significa que sea la única que vaya a ser diseñada, un ejemplo lo tenemos en la mejora propuesta por Krüger [51], basada principalmente en la aceleración del *z-test* para finalizar el algoritmo de trazado de rayos cuando el nivel de opacidad es suficientemente alto como para no variar el resultado. La velocidad del algoritmo es mejorado en un factor de 3 para modelos típicos de datos volumétricos en tarjetas ATI 9700. Sin embargo, el hardware está avanzando en dirección a la texturización, ya sean mediante texturas 2D o texturas 3D, lo que ha hecho decantar la balanza a favor de estas técnicas [2, 1].

1.4. Visualización expresiva de datos científicos

Debido a la heterogeneidad de la información de los volúmenes representados, su visualización directa con técnicas fotorrealistas producen imágenes donde es difícil discernir diferentes componentes dentro del volumen. De ahí la necesidad de utilizar técnicas que, si bien no son fotorrealistas, producen imágenes fáciles de interpretar por el sistema perceptual humano. Hay que tener en cuenta que las imágenes más claras no son aquellas más realistas (pensemos en un flujo de aire por ejemplo, el cual es casi invisible para el ser humano), si no aquellas que estén preparadas para el ser humano (un esquema sería más comprensible que los datos realistas del flujo de aire).

La visualización expresiva de datos científicos o simplemente visualización expresiva es un área de informática gráfica donde se crean imágenes para representar una serie de datos, muchas veces imposibles de representar mediante una visualización realista.

Los objetivos de este área incluyen la visualización compatible con el sistema perceptual humano de unos datos que no son fácilmente discernibles por el ser humano, debido principalmente a que en la realidad son invisibles o casi invisibles y a la gran cantidad de datos que hay.

La simulación de arte también es uno de los objetivos de este tipo de visualización, aunque no suele ser lo común ya que en este aspecto, se suele trabajar para crear ilustraciones científicas de forma casi exclusiva.

Este campo se inspira muchas veces en los resultados obtenidos en la visualización expresiva de modelos poligonales [28]. Estas técnicas tratan de simplificar datos irrelevantes del modelo, realzando aquellos datos más llamativos dentro del modelo.

Como hemos referido, en la visualización directa de volúmenes la cantidad de información visual es extremadamente amplia. Es por ello que intentar una simulación física real del comportamiento de la luz en cada uno de los *vóxeles* puede ser confuso o inclusive inútil debido a que en la mayoría de los modelos volumétricos obtenidos a partir de captura de datos científicos, una simulación física hace que el observador no distinga los datos o formas que está visualizando.

Además en la mayoría de los casos tampoco toda la información que se está visualizando es relevante, es por ello que se puede realizar una simplificación que mejora la comprensión de la imagen resultante. A todo esto habría que añadir que en numerosos casos, los datos visualizados no tienen como destino a un especialista, sino a personas que tienen interés en el tema pero que no están entrenados para visualizar datos científicos de una determinada índole. Es el caso, por ejemplo, de un paciente que desea conocer su estado a partir de los datos obtenidos mediante cualquier instrumental científico o que tiene que decidir si operarse o no en situaciones en que el médico pide la opinión del paciente. Otro caso posible puede ser en la información meteorológica, el resultado es obtenido a partir de simulaciones y la visualización del resultado pasa por una fase de abstracción humana antes de ser ofrecida al público.

En todos estos casos existe una fase intermedia de abstracción entre la visualización directa del volumen y la imagen final que se le ofrece al usuario. Sin embargo, no son las únicas aplicaciones que tiene la visualización expresiva. Determinados elementos en el volumen pueden tener un interés especial, sobre todo en forma o tamaño. Muchos de estos elementos quedan difuminados por la cantidad de información intermedia que se visualiza, y que también es relevante debido a la posición e interacción de todos los elementos en el modelo. Es por ello que conviene realzar siluetas, bordes y sombreados de los elementos de interés para poder diferenciarlos claramente.

De acuerdo a estas aproximaciones podemos afirmar que la visualización expresiva es aquel conjunto de técnicas que realizan un proceso intermedio de abstracción parecido al que pueden realizar los seres humanos y que obtienen una imagen final fácil de asimilar y es parecida a como el ser humano la puede producir.

Además podemos afirmar que en la parte de volúmenes existen dos tipos de visualización directa de volúmenes de forma expresiva:

- Realce: Aquella que se dedican a realzar partes, como un proceso más de realce de la imagen resultante [28].
- Abstracción: Aquellos que intentan obtener imágenes más abstraidas de los datos originales, como puede ser ilustraciones o dibujos y que intentan explicar a personas no especialistas lo que están viendo [96].

El cauce de procesamiento para obtener imágenes expresivas se diferencia principalmente del de la visualización directa en que no suele haber un proceso de iluminación, sino que se basa en una serie de funciones (llamadas *funciones de transferencia*) que modifican los datos del volumen según información obtenida a partir de los datos, tal y como muestra la figura 1.4.

El diseño de las funciones de transferencia es un área de investigación activa actualmente, y es un área que está muy relacionada con el análisis de imágenes e incluso con la visión

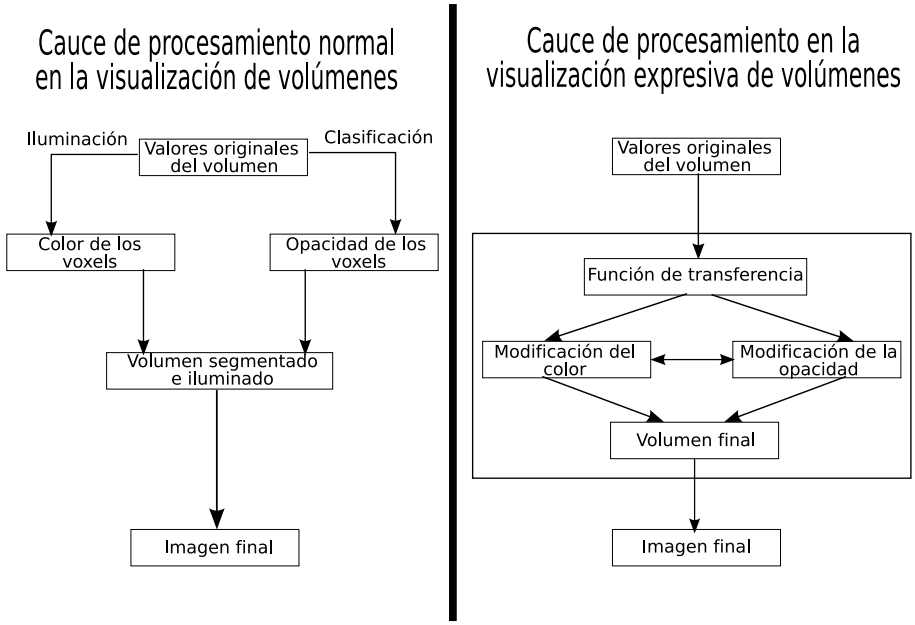

Figura 1.4: Cauce de procesamiento general para la representación de datos volumétricos de forma clásica y de forma expresiva

artificial [48]. Sin embargo, la mayoría de las funciones de transferencia están limitadas por su dependencia de los valores de los *vóxeles* al dominio de una única función y porque no son flexibles desde el punto de vista del usuario, requiriendo un gran esfuerzo por parte del usuario para obtener la imagen deseada. Además no permite realizar ilustraciones parecidas a la que los ilustradores realizan de forma sencilla e intuitiva.

En este trabajo propondremos un nuevo paso de iluminación a partir de las normales extraidas mediante técnicas de análisis de imágenes, que se aplica después de realizar una función de transferencia o una segmentación. Realzando las imágenes obtenidas y realizando ilustraciones parecidas a las que un ilustrador puede realizar.

Capítulo 2

Visión humana, análisis de imágenes y visualización expresiva

Previamente a introducirnos de lleno en los trabajos más recientes acerca de la visualización expresiva de volúmenes, vamos a realizar en este capítulo una introducción de los campos necesarios para comprender que métodos y técnicas se suelen aplicar y cómo se aplican sobre los datos obtenidos.

Introducción

Analizaremos primero el sistema de visión humano, que tomaremos como referencia tanto para el análisis de los datos, como para la representación expresiva de datos. En un segundo apartado veremos las técnicas más comunes de análisis de imágenes y que utilizaremos posteriormente en todos los capítulos, ya que esta es una parte común en todo procesamiento de imágenes y volúmenes. Por último se repasarán los métodos más comunes de visualización expresiva y las técnicas más empleadas por los artistas.

2.1. Percepción humana

La interacción del ser humano-máquina se ve reflejada en la figura 2.1, el ser humano (1) tiene un medio artificial (2) con el cual es capaz de comunicarse o interaccionar con la máquina (3) sobre un modelo matemático (4) tratado computacional (6) en base a unos conocimientos previos humanos (5).

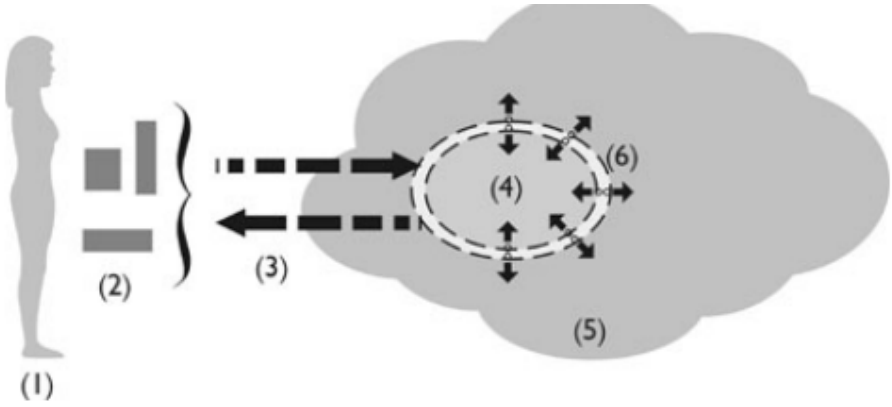

Figura 2.1: Representación de la interacción humano-máquina.

Ver no es sólo representar una imagen en la mente, sino también comprenderla y obtener un modelo mental de lo que se está percibiendo por los sentidos (en este caso la vista). Por tanto la visión es una tarea de procesamiento de información. Esta información no se procesa

toda al mismo tiempo, sino que sigue una determinada secuencia de pasos hasta que se obtiene el modelo mental de un mundo 3D. Estos pasos se realizan por niveles, desde lo más físico a lo más abstracto (proceso ascendente) y de lo más abstracto a lo más físico (proceso descendente). En casos en los que hay suficiente información (y no hay demasiada ambigüedad) se emplea un proceso ascendente, mientras que con figuras que no nos denotan información suficiente se emplea el proceso contrario (descendente).

Podemos comprobar con un sencillo experimento cómo trabaja la visión humana a grandes rasgos, si no conocemos nada sobre la figura 2.2 acerca de qué se esconde detrás, no tenemos ninguna representación mental del objeto (intentamos un proceso ascendente sin éxito), cuando nos dicen que hay un dálmata nuestro sistema visual actúa de forma inversa y vemos "lo que queremos ver", se hace un proceso de emparejamiento de nuestro modelo mental con la figura que estamos observando y obtenemos un resultado muy alto, haciendo que nuestro sistema visual nos informe sobre lo que hay en la figura.

Nos centraremos en el modelo ascendente, en el cual la ambigüedad es, a priori, menor para imágenes sintéticas (ya que no hace falta tener un modelo mental clave para obtener una representación mental no ambigua).

Las etapas que se suelen diferenciar en cuanto a procesamiento visual son las siguientes [62]:

- **Imagen**: en esta fase hay una representación de la intensidad
- **Esbozo primitivo**: en esta fase se representan las propiedades de la imagen bidimensional, tales como cambios de intensidad, bordes, geometría local bidimensional, etc.
- **Esbozo $2\frac{1}{2}$D**: aquí se representan las propiedades de las superficies visibles en un sistema de coordenadas centrado en el observador, orientaciones, distancias, discontinuidades, reflectancia, algo de iluminación, etc.
- **Representación del modelo 3D**: en este paso ya existe una representación centrada en el objeto de la estructura tridimensional y de la organización de la forma observada, junto con alguna descripción de las propiedades de su superficie.

Vamos ahora a examinar sobre los cambios que puede producir la visualización expresiva en estas descomposiciones.

Cambios que Afectan a la Fase de Imagen

Dado que el ser humano ha evolucionado a lo largo de millones de años para sobrevivir en ambientes naturales, su percepción para cada una de las longitudes de onda (rojo, verde, azul, o mezcla de ellas) es diferente, de hecho, el ser humano es muy bueno detectando diferentes tonalidades de verdes. Esta discrepancia se refleja directamente en la percepción de la intensidad, de tal forma que una imagen es más clara para el ser humano si tiene mayor cantidad de verde en un espacio de color denominado YUV, en la cual la luminancia viene dada por la componente Y (en la cual hay parte de longitudes de onda del verde), mientras que U y V son componentes cromáticas (dadas en su mayoría por rojo y azul) [35].

De esta forma podemos realizar una supresión de componentes cromáticos sin que el ojo humano, en su más bajo nivel de procesamiento note diferencia. Eliminar componentes de color verde hace que el ser humano pierda la capacidad de distinción de luminancia en una

Figura 2.2: Dálmata de Marr, solamente se puede apreciar cuando se conoce lo que se está mirando.

relación de 7/9. Lo cual puede afectar en procesos posteriores de detección de contornos, manchas, etc[94].

Por tanto hay que evitar supresiones de tonos de verdes en grandes cantidades. También hay que tener en cuenta que determinados colores afectan el comportamiento humano, detallados estudios psicológicos demostraron que paredes pintadas de rojos y amarillos hacen que una persona trate de salir de dicha habitación lo más rápidamente posible. El rojo es un color que suele afectar a la agresividad de la gente. Aunque los colores dependen más de las connotaciones culturales en los diferentes países, es una regla general que la gente de países tropicales responda más favorablemente a los colores cálidos, y la gente de los países nórdicos prefieran los colores más fríos.

Cambios que Afectan al Esbozo Primitivo

En esta fase existen varios pasos intermedios en los que se detectan:

- **Ceros**: cuando la función de la intensidad se hace cero (cambios de intensidad)
- **Manchas**: cuando difuminamos una imagen aparecen manchas
- **Terminaciones y discontinuidades**: unión de ceros muy cercanos
- **Segmentos de bordes**: unión de terminaciones cercanas
- **Líneas virtuales**: unión de terminaciones según su orientación
- **Grupos**: agrupación de objetos como uno sólo
- **Organizaciones curvilíneas**: expresiones matemáticas regulares se observan como líneas
- **Límites**: el borde perfectamente calculado

Es, por tanto, este esbozo primitivo donde el ser humano distingue formas en su más bajo nivel de detalle, detecta manchas de color y bordes que delimitan estas manchas, el cambio mediante visualización expresiva puede ser poco relevante o drástico según la fase que alteremos.

Si damos por supuesto que la detección de los cambios están basados en la diferencia de los cambios de intensidad (también llamada detección de ceros), un ser humano no podrá distinguir figuras (o al menos, no fácilmente) donde el cambio de intensidad es prácticamente nulo. Esto es algo que todos podemos comprobar, un cuadro dibujado con un color (de una determinada longitud de onda) en un fondo de similar tono e intensidad se hace indistinguible, y si mezclamos movimiento podemos llegar a percibir un ligero cambio de tonalidad pero nos es imposible mantener la trayectoria del objeto. Es por ello que hay que evitar que los objetos de mismo tono se solapen unos sobre otros, y que los escenarios sean del mismo tono que los objetos que existen sobre él [38].

El ser humano no sólo distingue bordes, sino que una parte importante del procesamiento del esbozo primitivo se basa en el relleno del objeto, es decir, la mancha que produce. Esta mancha puede estar formada por elementos de intensidad similar, no tiene por qué ser la misma, e incluso hacer uso de agrupación [3]. Trabajos recientes en visualización expresiva se orientan a obtener una proyección de los elementos como manchas de colores, similar a las figuras obtenidas por el pincel de un pintor, en estos dibujos no se refuerza la continuidad de los bordes (como algo bien delimitado), sino que se hace que las manchas sean continuas y de tonos aproximados (tal y como aparece en la figura 2.3).

Puesto que las terminaciones y discontinuidades, segmentos virtuales y líneas virtuales tienen una frontera tan estrecha entre ellas, vamos a considerar a todos estos pasos como la detección de los bordes en una imagen. Para objetos tridimensionales se pueden definir dos tipos de bordes (los mismos que términos que aparecen en visualización expresiva):

- **Silueta**: límite entre la parte visible y la no visible de un objeto.
- **Líneas de forma interna**: líneas que marcan cambios bruscos de intensidad en el objeto y que no son silueta.

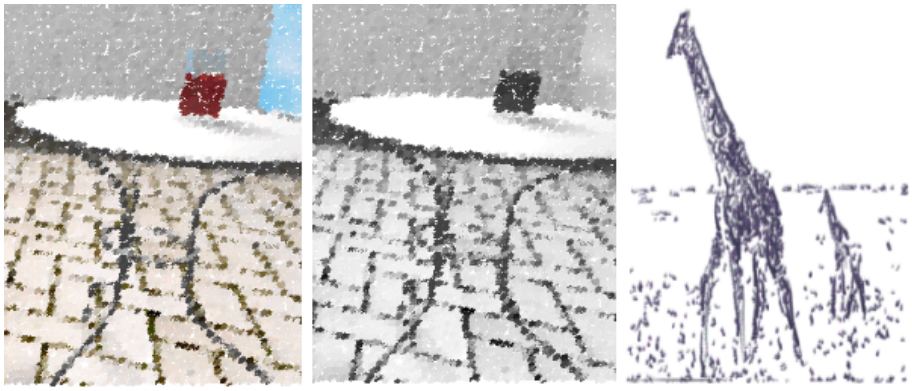

Figura 2.3: Como se puede apreciar los dibujos no tienen siluetas o líneas de forma ninguna, tan solo son manchas de colores o de tonos de gris.

Como hemos visto, una mancha de color puede ser captada como objeto en las primeras fases de procesamiento, sin embargo no es el único ni el más importante elemento en la visión al más bajo nivel, los bordes también son una parte fundamental en el cálculo de la profundidad (estado de $2\frac{1}{2}$D). A la hora de discernir diferentes objetos, el sistema de percepción humano produce un paso de abstracción, creando una serie de elementos nuevos imaginarios, que no existen en la realidad (nuestro cerebro las obtiene mediante diversos métodos), es por ello que las técnicas de visualización expresiva ayudan a percibir mejor estos objetos (ya que simulan parte de estos comportamientos del cerebro o se aprovecha de ellos para generar la imagen final).

Un aspecto importante del esbozo primitivo es la construcción de límites entre regiones partiendo, no sólo de la intensidad o del color (como hemos visto), sino también de claves que pueden estar originadas por discontinuidades en la orientación de la superficie o en la distancia al observador. Debido a que el ser humano tiende a agrupar objetos por similitud de patrones (se realiza primero una comparación entre formas y se determinan si se pueden emparejar o no), podemos plantearnos realizar mundos tridimensionales en los cuales se modifiquen las imágenes resultantes para aparecer ante un observador como un cambio de patrones.

El ser humano agrupa objetos siguiendo los criterios que a continuación exponemos (figura 2.4):

- **Proximidad**: objetos que están a poca distancia unos de otros se suelen tratar como un todo.
- **Tamaño**: objetos que tienen aproximadamente el mismo tamaño se tratan como uno sólo.
- **Orientación**: objetos que están orientados en una misma dirección aproximada también se tratan como uno sólo.
- **Geometría**: si determinados objetos se presentan como una misma figura geométrica también son un todo.

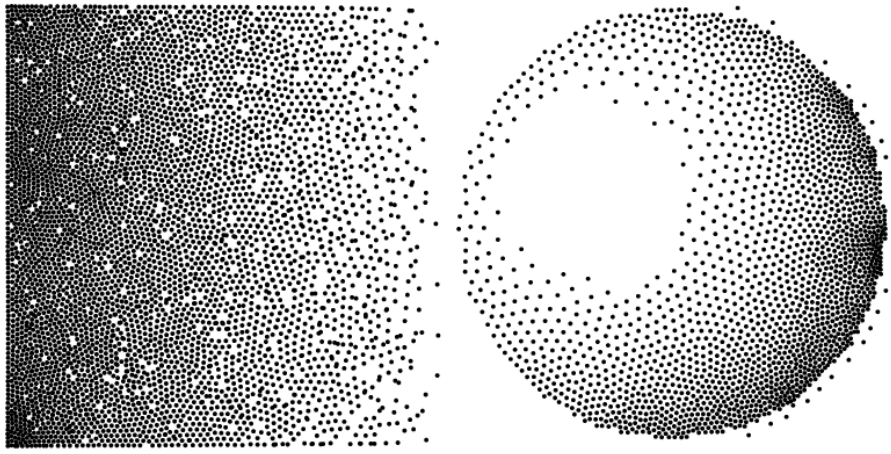

Figura 2.4: Diferentes figuras que el ojo humano identifica mediante cambios en el patrón.

Es por ello que objetos tridimensionales pueden llegar a ser expresados, no como proyección, sino como una alteración de distintos patrones que nos dan a entender una forma aprovechando esta asociación que realiza el ser humano en las primeras fases de la visión.

Cambios que Afectan al Esbozo $2\frac{1}{2}$D

En esta fase de la visión humana se conoce qué está delante y qué detrás, para ello ha de realizarse un proceso de triangulación. Hasta ahora hemos tratado de expresar los cambios que se pueden producir en la visión con una imagen generada por ordenador, sin embargo, para realizar la triangulación, necesitamos dos imágenes en las cuales podamos obtener unos puntos comunes desplazados en x posiciones, a partir de un cálculo trigonométrico simple se puede obtener su posición tridimensional [62].

Los parámetros importantes en este cálculo son los siguientes:

- **Disparidad** (α): discrepancia angular en la posición de la imagen de un objeto en ambos ojos.
- **Distancia** (d): distancia física objetiva del observador respecto a los objetos.
- **Profundidad** (p): distancia subjetiva de los objetos tal y como los percibe el observador.
- **Observador** (O): es el sujeto que contempla la escena, viene dado por dos parámetros: su posición (O_p) y la dirección en la que mira (O_d).
- **Objetos** (T): son los objetos a los que se está mirando, y cada uno viene definido por una única posición matemática.

Los tres primeros puntos definen la visión del observador, mientras que los dos últimos puntos definen el mundo tridimensional en el que se mueve (en nuestro caso el mundo virtual generado).

El cambio en la disparidad sólo se da en visión estéreo y viene reflejado directamente en las dos imágenes que devuelve el computador, lo único que se realiza en este paso intermedio es una proyección sobre la pantalla al igual que ocurriría con una imagen real sobre la retina humana. Este cambio en el parámetro de disparidad (α) puede provocar problemas graves de disociación de la imagen, debido a que nuestros ojos tienen una cierta separación, este parámetro debería ser ajustado para cada usuario, ya que en caso contrario, no se podría asimilar ambas imágenes como una sola.

En cuanto a los cambios en el parámetro de profundidad (p) no pueden ser dados debido a que es un parámetro intrínseco al usuario, sin embargo, se puede ajustar la distancia (d) junto a deformaciones de los objetos para asemejar un cambio más brusco en la percepción de las distancias, el cambio se produce debido a la deformación de la profundidad que el ser humano realiza en la fase de esbozo $2\frac{1}{2}$D, muy utilizado en las viñetas de los tebeos (ver figura 2.5).

Figura 2.5: En esta imagen se producen una serie de deformaciones que producen discrepancias en las distancias detectadas por el sistema visual humano.

Los cambios que tienen en cuenta al observador y a los objetos tienen las mayores posibilidades, ya que en un modelo matemático tridimensional, el sujeto está posicionado con unas coordenadas en el espacio, en función de hacia donde mira (vector O_d) se proyecta una perspectiva del objeto. Sin embargo, podemos variar el proceso y transformar un objeto según O_d, en lugar de hacer que el objeto siempre se mueva coherentemente con la dirección y posición del observador, cambios pequeños son bien admitidos (figura 2.6).

Cambios más bruscos en la escena pueden estar permitidos, si en lugar de mostrar al usuario una proyección sobre la pantalla de la misma forma que se genera en nuestra retina

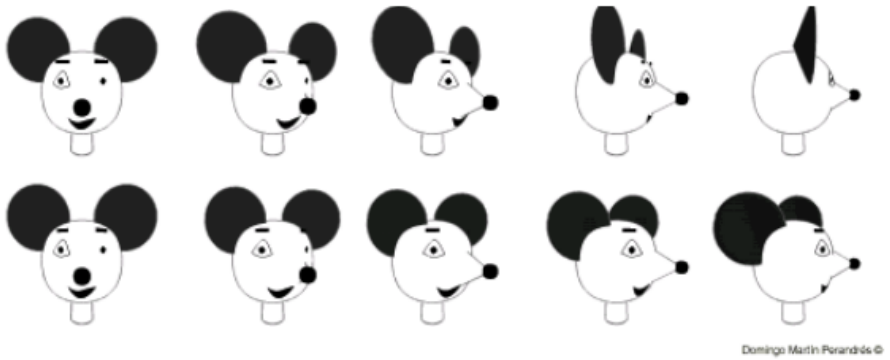

Figura 2.6: Rotaciones dependientes de la dirección en la que el observador está mirando la escena (generado tridimensionalmente por computador).

(llamada proyección en perspectiva) lo hacemos con una proyección paralela (u ortogonal), en la que los objetos no se hacen más pequeños con la distancia, los resultados obtenidos son parecidos a los que se obtienen en un entorno de ventanas, en el que el usuario conoce qué está detrás y lo distingue de lo que está delante, pero no es capaz de detectar distancias en profundidad. Este tipo de cambios se asemejan a mostrar al usuario un esbozo $2\frac{1}{2}$D directamente. Debido a que no existe una deformación en este tipo de proyecciones resulta ventajoso en determinadas situaciones en las que el usuario necesita conocer en todo momento el tamaño exacto de los objetos con los que está trabajando, sin necesidad de estimarlo en un proceso mental posterior en la visión.

Contraste simultáneo

Hasta ahora no nos hemos referido al color, una característica perceptual humana importante es que no todos los tonos son igualmente percibidos. Se ha demostrado que el color verde es más luminoso que el resto de tonos, esto significa que cualquier elemento de verde aparecerá más iluminado que el equivalente en rojo o en azul. Además de esto, el sistema perceptual humano es sensible al contraste de luminancia entre distintos tonos adyacentes, es decir, que el sistema perceptual detecta la luminosidad de forma relativa a los tonos de alrededor, en lugar de ser absoluto en términos de sus valores propiamente dichos (ver figura 2.7 para comprobar cómo el mismo color se aprecia más amarillento cuando tiene fondo rojo, y más rojizo cuando tiene fondo amarillo) [38].

Con respecto a esto existe una ley llamada ley de Weber, que dice así: si la luminancia de un objeto f_0 es tan solo notable por la diferencia de luminacia f de su alrededor, entonces su radio es

$$\frac{|f_s - f_0|}{f_0} = c \tag{2.1}$$

donde c es una constante, $f_0 = f$, $f_s = f + \Delta f$, donde Δf es pequeña comparada con las diferencias de luminancia notables por el sistema perceptual humano. Podemos escribir, por tanto

$$\frac{\Delta f}{f} \simeq \mathrm{d}(\log(f)) = \Delta c \tag{2.2}$$

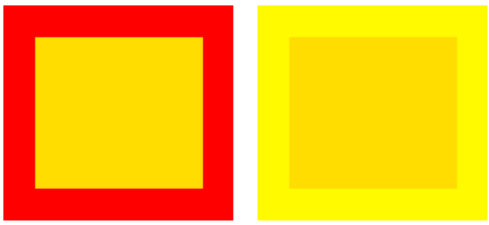

Figura 2.7: Ejemplo de contraste simultáneo, el centro de ambos rectángulos es del mismo color.

Donde d era la distancia que se comentaba en la sección anterior. El valor de la constante puede encontrarse de forma experimental, y es $c = 0,02$, lo que significa que se necesitan al menos 50 niveles para el contraste en una escala de 0 a 1.

2.2. Análisis de imágenes

Puesto que el volumen no es más que una imagen tridimensional, podemos tratarlo como tal, y aplicar los mismos algoritmos que se suelen aplicar en imágenes para obtener información adicional. De hecho, podría decirse que una imagen es un subconjunto del conjunto volumen (en el capítulo 7).

Es por ello que se suelen utilizar técnicas de análisis de imágenes como paso previo a la visualización de volúmenes, ya sea de forma realista o de forma expresiva.

2.2.1. Notación y definiciones

En esta sección vamos a definir la notación y discutir algunas nociones matemáticas preliminares que serán utilizadas en todo el capítulo, y en general, siempre que estemos realizando operaciones de análisis de imágenes. Comenzaremos definiendo la notación que usaremos y revisaremos algunas propiedades de la transformada de Fourier, además revisaremos el fundamento de los resultados de la teoría de matrices y la importancia en el análisis de imágenes.

Una representación de una señal continua se representará como una función de una variable: $f(x)$, $u(x)$, $s(t)$, etc. Una señal muestreada unidimensional se escribirá como una sola secuencia de índices: u_n, $u(n)$, etc.

Una imagen continua se representará como una función de dos variables independientes: $u(x,y)$, $u(x,y)$, $f(x,y)$, etc. Una imagen muestreada se representará como una secuencia n-dimensional de números reales (con $n \geq 2$): $u_{m,n}$, $v(n,m)$, $u(i,j,k)$, etc.

A menos que se especifique lo contrario, los símbolos i, j, k, l, m, n, ... se usarán para especificar índices enteros de *arrays* y vectores. La letra latina **j** representará $\sqrt{-1}$. El conjugado complejo de una variable compleja como z, se denotará como z^*. Ciertos símbolos se redefinirán en los lugares apropiados en el texto para mantener una notación clara.

Vamos a definir algunas propiedades útiles de varias funciones unidimensionales bien conocidas en análisis de imágenes, y que serán utilizadas posteriormente para el análisis de la transformada de Fourier y las operaciones de convolución [30].

La tabla 2.1 muestra varias funciones unidimensionales que son bien conocidas, y que aparecerán frecuentemente. Estas versiones bidimensionales son funciones de la forma separable:

$$f(x,y) = f_1(x)f_2(y) \tag{2.3}$$

Por ejemplo, las dos funciones bidimensionales delta son definidas como:

$$\text{Dirac: } \delta(x,y) = \delta(x)\delta(y) \tag{2.4}$$

$$\text{Kronecker: } \delta(m,n) = \delta(m)\delta(n) \tag{2.5}$$

las cuales satisfacen las siguientes propiedades:

$$\left.\begin{array}{l} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x',y')\delta(x-x',y-y')\mathrm{d}x'\mathrm{d}y' = f(x,y) \\ \lim_{\varepsilon\longrightarrow 0}\int_{-\varepsilon}^{\varepsilon}\int_{-\varepsilon}^{\varepsilon}\delta(x,y)\mathrm{d}x\mathrm{d}y = 1 \end{array}\right\} \tag{2.6}$$

$$\left.\begin{array}{l} x(m,n) = \sum_{m'=-\infty}^{\infty}\sum_{n'=-\infty}^{\infty} x(m',n')\delta(m-m',n-n') \\ \sum_{m'=-\infty}^{\infty}\sum_{n'=-\infty}^{\infty}\delta(m,n) = 1 \end{array}\right\} \tag{2.7}$$

Las definiciones y propiedades de las funciones $rect(x,y)$, $sinc(x,y)$ y $comb(x,y)$ pueden ser definidas de forma similar [76].

Función	Definición	Función	Definición
	$\delta(x) = 0, x \neq 0$	Rectángulo	$rect(x) = \begin{cases} 1 & \text{si } \lvert x\rvert \leq \frac{1}{2} \\ 0 & \text{si } \lvert x\rvert > \frac{1}{2} \end{cases}$
Delta de Dirac	$\lim_{\varepsilon\longrightarrow 0}\int_{-\varepsilon}^{\varepsilon}\delta(x)\mathrm{d}x = 1$	Signo	$sgn(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x = 0 \\ -1 & \text{si } x < 0 \end{cases}$
Propiedad de suavizado	$\int_{-\infty}^{\infty} f(x')\delta(x-x')\mathrm{d}x' = f(x)$	Sinc	$sinc(x) = \frac{\sin \pi x}{\pi x}$
Propiedad de escalado	$\delta(ax) = \frac{\delta(x)}{\lvert a\rvert}$	Comb	$comb(x) = \sum_{n=-\infty}^{\infty}\delta(x-n)$
Delta de Kronecker	$\delta(n) = \begin{cases} 0 & \text{si } n \neq 0 \\ 1 & \text{si } n = 0 \end{cases}$	Triángulo	$tri(x) = \begin{cases} 1-\lvert x\rvert & \text{si } \lvert x\rvert \leq 1 \\ 0 & \text{si } \lvert x\rvert > 1 \end{cases}$
Propiedad de suavizado	$\sum_{m=-\infty}^{\infty} f(m)\delta(n-m) = f(n)$		

Cuadro 2.1: Sistemas bidimensionales y preliminares matemáticos.

Algunas de estas propiedades son utilizadas en detectores de contornos y en seguimiento de los mismos.

2.2.2. Sistemas lineales, invarianza suave y convolución

Un gran número de sistemas de imágenes pueden modelarse como sistemas lineales bidimensionales. Sean $x(m,n)$ y $y(m,n)$ la secuencia de entrada y salida, respectivamente, de un sistema bidimensional, se puede escribir de la siguiente forma (ver figura 2.8):

$$y(m,n) = \mathscr{H}[x(m,n)] \tag{2.8}$$

Figura 2.8: Sistema bidimensional.

Este sistema se llaman lineales si y solo si cualquier combinación lineal de dos entradas $x_1(m,n)$ y $x_2(m,n)$ producen la misma combinación de sus respectivas salidas $y_1(m,n)$ y $y_2(m,n)$, por ejemplo, para las constantes arbitrarias a_1 y a_2

$$\begin{aligned} \mathscr{H}[a_1x_1(m,n)+a_2x_2(m,n)] = \\ a_1\mathscr{H}[x_1(m,n)]+a_2\mathscr{H}[x_2(m,n)] = \\ a_1y_1(m,n)+a_2y_2(m,n) \end{aligned} \tag{2.9}$$

Esto se denomina *superposición lineal*. Cuando la entrada es la función bidimensional de la delta de Kronecker en la localización (m',n'), la salida de la localización (m,n) es definida como

$$h(m,n;m',n') \triangleq \mathscr{H}[\delta(m-m',n-n')] \tag{2.10}$$

y se denomina *respuesta impulso* del sistema. Para un sistema de imágenes, es la imagen resultante en el plano de salida respecto a un punto origen ideal en la localización (m',n') en el plano de entrada. En nuestra notación, el punto y coma $(;)$ se emplea para distinguir el punto de entrada (a la izquierda del mismo) del de salida (a la derecha).

La *respuesta impulso* del sistema es denominada *función de propagación puntual* (*point spread function*), cuando las entradas y salidas representan siempre un valor positivo, como puede ser la intensidad de la imagen.

El término *respuesta impulso* es más general y también se permite para valores negativos y complejos. La *región de apoyo* de una *respuesta impulso* es la región cerrada más pequeña en el plano exterior m,n, en este caso la *respuesta impulso* es 0.

Un sistema es denominado como una *respuesta impulso finita* o una *respuesta impulso infinita* si su *respuesta impulso* tiene, respectivamente, finitas o infinitas regiones de apoyo, respectivamente.

La salida de cualquier sistema lineal puede obtenerse de su respuesta impulso a la entrada aplicando la regla de superposición (ver ecuación 2.10) para la representación de la ecuación 2.7 de la forma siguiente:

$$\begin{aligned} y(m,n) = \mathscr{H}[x(m,n)] = \\ \mathscr{H}\left[\sum_{m'}\sum_{n'}x(m',n')\delta(m-m',n-n')\right] = \\ \sum_{m'}\sum_{n'}x(m',n')\mathscr{H}[\delta(m-m',n-n')] \Rightarrow \\ y(m,n) = \sum_{m'}\sum_{n'}x(m',n')h(m,n;m',n') \end{aligned} \tag{2.11}$$

Un sistema es denominado *invariante espacial* o *invariante de desplazamiento* si una translación en la entrada causa una translación en la salida. Siguiendo la definición 2.10, si el

impulso ocurre en el origen tenemos:

$$\mathscr{H}[\delta(m,n)] = h(m,n;0,0) \tag{2.12}$$

De esta forma, debe ser cierto para sistemas *invariantes de desplazamiento* que

$$\begin{aligned} h(m,n;m',n') &\triangleq \mathscr{H}[\delta(m-m',n-n')] \\ &= h(m-m',n-n';0,0) \\ \Rightarrow h(m,n;m',n') &= h(m-m',n-n') \end{aligned} \tag{2.13}$$

por ejemplo, la respuesta impulso de este sistema es una función de dos variables de desplazamiento únicamente. Esto significa que la forma de la respuesta impulso no cambia como el movimiento del impuso sobre el plano m,n. Un sistema se denomina *espacialmente cambiante* cuando la ecuación 2.13 no es cierta.

Para los sistemas *invariantes de desplazamiento*, la salida llega a ser

$$y(m,n) = \sum_{m'}^{\infty}\sum_{n'}^{\infty} h(m-m',n-n')x(m',n') \tag{2.14}$$

la cual se denomina *convolución* de la entrada con la *respuesta impulso*. La figura 2.9 muestra una interpretación gráfica de esta operación.

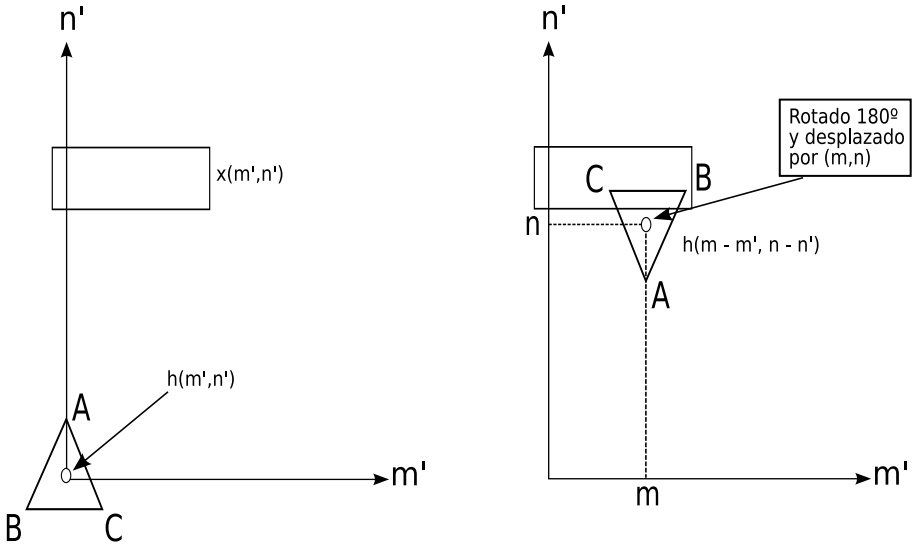

Figura 2.9: Operación de convolución

La figura de la respuesta impulso es rotado desde el origen 180^{o} y después es desplazado por (m,n) y solapado sobre la figura $x(m',n')$. La suma del producto de los *arrays* que conforman ambas figuras $\{x(\cdot,\cdot\}$ y $\{h(\cdot,\cdot\}$ en la región de solapado resulta en (m,n). Usaremos

el símbolo $\circledast$ para denotar la operación de convolución, tanto en el caso continuo como en el discreto.

En el caso discreto tenemos usualmente dos matrices (que corresponderán a *arrays* en una imagen), estas matrices tendrán un tamaño $(M_1 \times N_1)$ y $(M_2 \times N_2)$ respectivamente. De tal forma que el resultado de la convolución será un *array* de tamaño: $[(M_1 + M_2 - 1) \times (N_1 + N_2 - 1)]$.

2.2.3. Transformada de Fourier

Las transformaciones bidimensionales tales como la transformada de Fourier tienen una importancia enorme en todo tratamiento de procesamiento de imágenes, ya que permite trabajar en el dominio de la frecuencia de una señal. En esta sección veremos las propiedades más importantes de la transformada de Fourier aplicadas a una imagen (ver tabla 2.2) [37].

En una dimensión la transformada de Fourier es una compleja función $f(x)$ definida como:

$$F(\xi) \triangleq \mathscr{F}[f(x)] \triangleq \int_{-\infty}^{\infty} f(x)\mathrm{e}^{(-\mathrm{j}2\pi\xi x)\mathrm{d}x} \tag{2.15}$$

La transformada de Fourier inversa de $\mathscr{F}(\xi)$ es

$$f(x) \triangleq \mathscr{F}^{-1}[F(\xi)] = \int_{-\infty}^{\infty} F(\xi)\mathrm{e}^{(\mathrm{j}2\pi\xi x)\mathrm{d}\xi} \tag{2.16}$$

La transformada de Fourier en dos dimensiones y su inversa se definen de forma análoga como la siguiente transformación lineal

$$F(\xi_1,\xi_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\mathrm{e}^{[-\mathrm{j}2\pi(x\xi_1+y\xi_2)]\mathrm{d}x\mathrm{d}y} \tag{2.17}$$

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(\xi_1,\xi_2)\mathrm{e}^{[-\mathrm{j}2\pi(x\xi_1+y\xi_2)]\mathrm{d}\xi_1\mathrm{d}\xi_2} \tag{2.18}$$

Las propiedades de la transformada de Fourier en dos dimensiones se expresan mediante la tabla 2.3. Las más relevantes se describen a continuación:

1. Frecuencias espaciales: $f(x,y)$ es la luminancia y x,y la coordenada espacial, entonces ξ_1,ξ_2 son frecuencias espaciales que representan cambios en la luminancia con respecto a las distancias espaciales.

2. Unicidad: en el caso de las funciones continuas, $f(x,y)$ y $F(\xi_1,\xi_2)$ son únicas con respecto la una a la otra. No hay pérdida de información de ningún tipo. Esta propiedad es ampliamente utilizada en compresión de imágenes.

3. Separabilidad: por definición, el núcleo de la transformada de Fourier es separable. Esto significa que puede escribirse como una transformación separable en x y en y, por ejemplo:

$$F(\xi_1,\xi_2) = \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(x,y)\mathrm{e}^{(-\mathrm{j}2\pi x\xi_1)\mathrm{d}x}\right]\mathrm{e}^{(-\mathrm{j}2\pi y\xi_2)\mathrm{d}y} \tag{2.19}$$

Esto significa que la transformación bidimensional puede ser realizada mediante una sucesión de transformaciones a lo largo de las coordenadas espaciales. Esta es una propiedad realmente interesante que utilizaremos a la hora de aplicar núcleos de convolución separables.

4. Respuesta de frecuencia o autofunciones: Una autofunción de un sistema es definida como una función de entrada que es reproducida a la salida únicamente con cambios en su amplitud. Una propiedad fundamental de un sistema invariante de desplazamiento es que sus autofunciones vienen determinadas por la exponencial compleja $e^{j}2\pi(\xi_1 x + \xi_2 y)$.

5. Teorema de convolución: la transformada de Fourier de la convolución de dos funciones es el producto de sus transformadas de Fourier. Este teorema sugiere que la convolución de dos funciones puede ser evaluada por la transformada de Fourier inversa transformando el producto de sus transformadas de Fourier. Este teorema es muy importante desde el punto de vista computacional ya que permite el algoritmo de la transformada de Fourier rápida.

6. Preservación del producto interno: otra propiedad importante de la transformada de Fourier es que el producto interno de dos funciones es igual al producto interno de sus transformadas de Fourier, por ejemplo:

$$I \triangleq \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)h^*(x,y)\mathrm{d}x\mathrm{d}y = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(\xi_1,\xi_2)H^*(\xi_1,\xi_2)\mathrm{d}\xi_1\mathrm{d}\xi_2 \tag{2.20}$$

7. Transformada de Hankel: La transformada de Fourier de una función circularmente simétrica es también circularmente simétrica y es dada por la llamada transformada de Hankel.

$f(x,y)$	$F(\xi_1,\xi_2)$
$\delta(x,y)$	1
$\delta(x\pm x_0, y\pm y_0)$	$e^{\pm \mathrm{j}2\pi x_0\xi_1} e^{\pm \mathrm{j}2\pi y_0\xi_2}$
$e^{\pm \mathrm{j}2\pi x\eta_1} e^{\pm \mathrm{j}2\pi y\eta_2}$	$\delta(\xi_1 \mp \eta_1, \xi_2 \mp \eta_2)$
$e^{-\pi(x^2+y^2)}$	$e^{-\pi(\xi_1^2+\xi_2^2)}$
$rect(x,y)$	$sinc(\xi_1,\xi_2)$
$tri(x,y)$	$sinc^2(\xi_1,\xi_2)$
$comb(x,y)$	$comb(\xi_1,\xi_2)$

Cuadro 2.2: Parejas de transformación de la transformada de Fourier Bidimensional.

2.2.4. Distribución gaussiana o normal

En el análisis de imágenes es usual el pasar primero un filtro de suavizado. Sin embargo, un filtro de mediana o de media lo único que consigue es emborronar la imagen por igual. Se ha demostrado que una buena aproximación al proceso de emborronado del sistema de visión humano es la función gaussiana.

Propiedad	Función $f(x,y)$	Transformada de Fourier $F(\xi_1,\xi_2)$
Rotación	$f(\pm x,\pm y)$	$F(\pm\xi_1,\pm\xi_2)$
Linealidad	$a_1 f_1(x,y)+a_2 f_2(x,y)$	$a_1 F_1(\xi_1,\xi_2)+a_2 F_2(\xi_1,\xi_2)$
Conjugación	$f^*(x,y)$	$F^*(\xi_1,\xi_2)$
Separabilidad	$f_1(x)f_2(y)$	$F_1(\xi_1)F_2(\xi_2)$
Escalado	$f(ax,by)$	$\frac{F(\xi_1/a,\xi_2/b)}{\lvert ab\rvert}$
Desplazamiento	$f(x\pm x_0,y\pm y_0)$	$e^{\pm j2\pi(x_0\xi_1+y_0\xi_2)}F(\xi_1,\xi_2)$
Modulación	$e^{\pm j2\pi(\eta_1 x+\eta_2 y)}f(x,y)$	$F(\xi_1\mp\eta_1,\xi_2\mp\eta_2)$
Convolución	$g(x,y)=h(x,y)\circledast f(x,y)$	$G(\xi_1,\xi_2)=H(\xi_1,\xi_2)F(\xi_1,\xi_2)$
Multiplicación	$g(x,y)=h(x,y)f(x,y)$	$G(\xi_1,\xi_2)=H(\xi_1,\xi_2)\circledast F(\xi_1,\xi_2)$
Correlación espacial	$c(x,y)=h(x,y)\star f(x,y)$	$C(\xi_1,\xi_2)=H(-\xi_1,-\xi_2)F(\xi_1,\xi_2)$
Producto interno	$I=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)h^*(x,y)\mathrm{d}x\mathrm{d}y$	$I=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(\xi_1,\xi_2)H^*(\xi_1,\xi_2)\mathrm{d}\xi_1\mathrm{d}\xi_2$

Cuadro 2.3: Propiedades de la transformada de Fourier bidimensional.

La función de densidad probabilística de una variable aleatoria u se nota como $p_u(u)$. Para una variable aleatoria gaussiana [53, 103]

$$p_u(u) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}e^{\frac{-|u-\mu|^2}{2\sigma^2}}} \tag{2.21}$$

donde μ y σ^2 son su varianza media, y u denota al valor que toma la variable aleatoria. Para $\mu=0$ y $\sigma^2=1$, se le denomina distribución normal estándar.

2.2.5. Detectores de contornos

Un paso fundamental en el análisis de imágenes, que es utilizado tanto para la detección de la dirección y magnitud de gradientes como para el cálculo de la dirección de las normales de la imagen es el detector de contornos. Este paso suele realizarse después de haber suavizado la imagen mediante algún filtro de emborronamiento.

Para detectar los bordes de una imagen se suele utilizar un proceso que simule la detección humana de bordes y contornos.

Un punto frontera en un contorno se puede definir en una imagen binaria (compuesta tan solo de píxeles negros y blancos) como aquel píxel negro rodeado al menos por un vecino cercano blanco.

Para una imagen continua $f(x,y)$ su derivada asume un máximo local en la dirección del contorno. De esta forma, un método para conocer la dirección de su contorno es calcular el gradiente de f a lo largo de r en una dirección θ:

$$\frac{\delta f}{\delta r}=\frac{\delta f}{\delta x}\frac{\delta x}{\delta r}+\frac{\delta f}{\delta y}\frac{\delta y}{\delta r}=f_x\cos\theta+f_y\sin\theta \tag{2.22}$$

El máximo valor de $\frac{\delta f}{\delta r}$ se obtiene cuando $\frac{\delta}{\delta\theta}\frac{\delta f}{\delta r}=0$. Con esto se puede obtener

$$-f_x\sin\theta_g+f_y\cos\theta_g=0\Rightarrow\theta_g=\tan^{-1}\left(\frac{f_y}{f_x}\right) \tag{2.23}$$

$$\left(\frac{\delta f}{\delta r}\right)_{max}=\sqrt{f_x^2+f_y^2} \tag{2.24}$$

donde θ_g es la dirección del contorno (ver figura 2.10). Basándonos en estos conceptos, se han introducido dos tipos de operadores de detección de contornos, *operadores de gradiente* y *operadores de ámbito*. Para las imágenes digitales esos operadores, también llamados

máscaras, representan aproximaciones diferenciales finitas de cada uno de los gradientes ortogonales f_x, f_y o la dirección del gradiente $\frac{\delta f}{\delta r}$. Denotaremos $\mathtt{H}$ como una máscara $p \times p$ y definiremos, para una imagen arbitraria $\mathtt{U}$, como su producto interno en la localización (m,n) como la correlación

$$\langle \mathtt{U}, \mathtt{H} \rangle_{m,n} \triangleq \sum_i \sum_j h(i,j)u(i+m, j+n) = u(m,n) \circledast h(-m,-n) \tag{2.25}$$

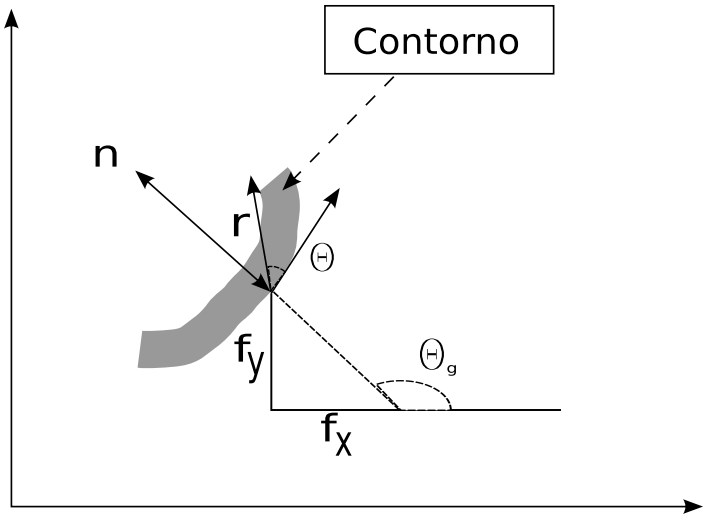

Figura 2.10: Gradiente $f(x,y)$ a lo largo de la dirección r.

Operadores gradiente

Estos operadores se representan por un par de máscaras $\mathtt{H}_1$, $\mathtt{H}_2$, las cuales calculan el gradiente de la imagen $u(m,n)$ en dos direcciones ortogonales. El gradiente bidireccional se define como[97, 90]

$$g_1(m,n) \triangleq \langle \mathtt{U}, \mathtt{H}_1 \rangle_{m,n} \tag{2.26}$$

$$g_2(m,n) \triangleq \langle \mathtt{U}, \mathtt{H}_2 \rangle_{m,n} \tag{2.27}$$

por tanto, la magnitud y dirección del gradiente puede calcularse como

$$g(m,n) = \sqrt{g_1^2(m,n) + g_2^2(m,n)} \tag{2.28}$$

$$\theta_g(m,n) = \tan^{-1} \frac{g_2^2(m,n)}{g_1^2(m,n)} \tag{2.29}$$

La tabla 2.4 muestra distintos operadores de gradiente. Los operadores de Prewitt y Sobel, así como el isotrópico, calculan diferencias de sumas locales horizontales y verticales, de forma que reduce el efecto de ruido en los datos. La localización del punto en la posición

(m,n) se define como una localización del contorno si $g(m,n)$ excede algún umbral t. Los puntos de las localizaciones del contorno constituyen un mapa de contornos $\varepsilon(m,n)$, que se define como

$$\varepsilon(m,n) = \begin{cases} 1 & \text{si } (m,n) \in I_g \\ 0 & \text{en otro caso} \end{cases} \tag{2.30}$$

donde

$$I_g \triangleq \{(m,n); g(m,n) > t\} \tag{2.31}$$

El mapa de contornos ofrece los datos necesarios para seguir las fronteras del objeto en una imagen. Normalmente, t puede seleccionarse usando un histograma acumulativo de $g(m,n)$ tal que del 5 al 10 % de los píxeles con gradientes mayores son declarados como contorno.

	H_1	H_2
Roberts	$\begin{pmatrix} \underline{0} & 1 \\ -1 & 0 \end{pmatrix}$	$\begin{pmatrix} \underline{1} & 0 \\ 0 & -1 \end{pmatrix}$
Prewitt	$\begin{pmatrix} -1 & 0 & 1 \\ -1 & \underline{0} & 1 \\ -1 & \underline{0} & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & -1 \\ 0 & \underline{0} & 0 \\ 1 & 1 & 1 \end{pmatrix}$
Sobel	$\begin{pmatrix} -1 & 0 & 1 \\ -2 & \underline{0} & 2 \\ -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & -2 & -1 \\ 0 & \underline{0} & 0 \\ 1 & 2 & 1 \end{pmatrix}$
Isotrópico	$\begin{pmatrix} -1 & 0 & 1 \\ -\sqrt{2} & \underline{0} & \sqrt{2} \\ -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & -\sqrt{2} & -1 \\ 0 & \underline{0} & 0 \\ 1 & \sqrt{2} & 1 \end{pmatrix}$

Cuadro 2.4: Algunos operadores comunes para gradientes.

Operadores de Laplace y Cruce de ceros

Los métodos previos de estimación de gradientes funcionan mejor cuando las transiciones de los niveles de gris son bastante abruptas. Cuando las transiciones entre las regiones se ensancha, es más ventajoso aplicar las derivadas de segundo orden. Un operador utilizado frecuentemente es el operador laplaciano, definido como

$$\nabla^2 f = \frac{\delta f}{\delta x^2} + \frac{\delta f}{\delta y^2} \tag{2.32}$$

Debido a que las derivadas de segundo orden son muy sensibles al ruido, el operador gradiente es más sensible al ruido que los que previamente han sido definidos. La magnitud umbralizada de $\nabla^2 f$ produce contornos dobles. Una aproximación mejor que la utilización del operador laplaciano es usar su cruce de ceros (*zero-crossing*) para detectar la localización de los contornos. Se suele utilizar un operador que aproxima la laplaciana de la función gaussiana, y es definida por

$$h(m,n) \triangleq c[1 - \frac{(m^2+n^2)}{\sigma^2}]e^{-\frac{m^2+n^2}{2\sigma^2}} \tag{2.33}$$

donde σ controla el ancho del núcleo gaussiano y c normaliza la suma de los elementos de una máscara dada. El cruce de ceros de una imagen dada convolucionada con $h(m,n)$ da la

localización de los contornos. El detector de cruce de ceros es equivalente a un filtro paso baja teniendo un impulso respuesta gaussiano seguido por un operador de Laplace. El parámetro σ controla la amplitud de la respuesta de la salida del filtro pero no afecta a la localización del cruce de ceros. La información de los bordes puede obtenerse buscando en la derivada de segundo orden a lo largo de r para cada dirección θ. De la ecuación 2.22 obtenemos

$$\frac{\delta^2 f}{\delta r^2} = \frac{\delta f_x}{\delta r}\cos\theta + \frac{\delta f_y}{\delta r}\sin\theta = \frac{\delta^2 f}{\delta x^2}\cos^2\theta + 2\frac{\delta^2 f}{\delta x^2}\cos^2\theta + 2\frac{\delta^2 f}{\delta x \delta y}\sin\theta\cos\theta + \frac{\delta^2 f}{\delta y^2}\sin^2\theta \quad (2.34)$$

2.2.6. Extracción de fronteras

La extracción de fronteras es útil para determinar la longitud de un contorno, y lo que trata es de dar conectividad a cada uno de los píxeles del contorno con sus vecinos.

Se dice que un píxel es cuatro- u ocho-conectado cuando tiene la misma propiedad que sus cuatro u ocho vecinos más cercanos respectivamente (ver figura 2.11).

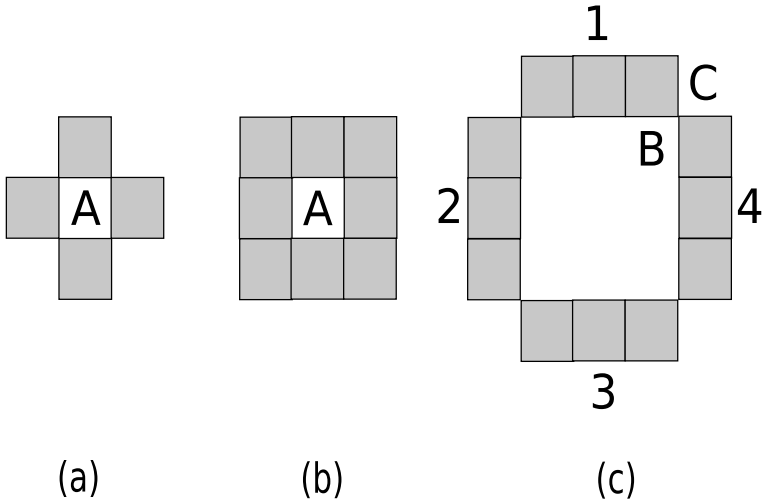

Figura 2.11: Ejemplos de conectividad: píxel *A* y sus vecinos cuatro-conectados (a), sus vecinos ocho-conectados (b), y paradoja de conexión (c) en la que hay ambigüedad en si B y C están conectados.

Veremos a continuación algunos algoritmos muy usados para la extracción de fronteras.

Seguidores de contornos

Los algoritmos seguidores de contornos recorren las fronteras ordenando series de puntos del borde. Es un algoritmo simple para recorrer fronteras cercanas en imágenes binarias:

1. Comienza en un punto arbitrario *A*.

2. Gira a la izquierda y pasa al siguiente píxel si está en la región A, en otro caso gira a la derecha y sigue al siguiente píxel.

3. Continuar hasta llegar al punto inicial 1.

Este algoritmo puede producir un contorno grueso, lo que no es deseable. El algoritmo puede refinarse utilizando *splines* para suavizar la curva en la representación, sin embargo, incluso con estos métodos la frontera aparece demasiado irregular.

Enlazado de bordes y búsquedas heurísticas en grafos

Una frontera puede ser tomada también como un camino a través de un grafo formado por el enlace de varios elementos del borde que están juntos. Las reglas para enlazar estos elementos nos ofrecen un método para conectar los bordes y formar fronteras. Supongamos un grafo con las siguientes localizaciones de los nodos x_k, $k = 1,2,\ldots$ que se forma desde el nodo A al nodo B. Supongamos también que se da una función de evaluación $\phi(x_k)$, la cual devuelve el valor del camino que va desde A hasta B restringido por el nodo x_k. En estos algoritmos examinamos los sucesores del nodo inicial y seleccionamos el nodo que maximiza $\phi(\cdots)$. El nodo seleccionado pasa a ser el nuevo nodo inicial y el proceso se repite hasta llegar a B.

Programación dinámica

La programación dinámica es un método de búsqueda de un óptimo global sobre procesos multi-etapa. Está basado en el principio de optimalidad de Bellman, el cual indica que el camino óptimo entre os puntos dados es también óptimo entre dos puntos del camino.

En estos algoritmos las fronteras se convierten en grafos y se trata de alcanzar el destino sin importar el cómo. Para ello se otorga una serie de pesos a cada uno de los nodos del grafo intentando encontrar la solución óptima que minimice el costo del grafo. El problema es que el tiempo empleado en esta búsqueda suele ser muy superior al de la mayoría de las heurísticas que tienen un comportamiento aceptable.

2.2.7. Algoritmo de Canny

Mención aparte merece el algoritmo de Canny. La necesidad de un algoritmo para determinar la detección óptima de fronteras incentivó la creación de un algoritmo que tuviera las siguientes características:

- Buena detección: el algoritmo debe marcar tantas fronteras reales en la imagen como sea posible.

- Buena localización: las fronteras marcadas deben ser tan cercanas como sea posible a la frontera real en la imagen.

- Respuesta mínima: una frontera dada en la imagen debe ser marcada una única vez, y donde sea posible, el ruido de la imagen no debe crear falsas fronteras.

Para satisfacer estos requisitos Canny [15] usó el cálculo de variaciones, una técnica que encuentra la función que mejor satisface una funcional dada. El detector óptimo es descrito

por una suma de cuatro términos exponenciales, pero cercana a la aproximación de la primera derivada de la gaussiana.

No se puede esperar que ningún algoritmo de detección de fronteras trabaje bien sobre imágenes sin procesar, por ello el paso principal del algoritmo es una máscara gaussiana. La salida aparece como una copia borrosa del original, por lo que un solo píxel no tiene apenas efecto sobre los de alrededor.

Posteriormente, se intenta encontrar el gradiente de la imagen. Para ello se determina una serie de máscaras (en concreto 4 máscaras) para detectar las fronteras horizontales, verticales y diagonales. Los resultados de convolucionar la imagen original con cada una de estas máscaras es almacenado. Para cada píxel, marcamos el resultado más grande en ese píxel, y la dirección de la frontera, la cual produce ese borde. Para cada imagen original, se crea un mapa de gradientes en cada punto de la imagen y la dirección de los gradientes en dichos puntos.

En un paso posterior se realiza el recorrido de las fronteras a través de la imagen. Los gradientes de mayor intensidad son, de forma muy probable, fronteras. No hay un valor exacto en el cual un gradiente de una determinada intensidad cambie de no ser frontera a serlo. De esta forma Canny usa umbrales con histéresis.

Los umbrales con histéresis requieren dos umbrales, uno alto y otro bajo. Asumiendo que las fronteras importantes deberían estar en las líneas continuas, podríamos seguir una sección débil de una línea dada, pero evitando identificar los píxeles ruidosos que no forman la línea. Comenzamos aplicando un umbral alto. Esto marca las fronteras de las cuales podemos estar seguros. Comenzando desde ellas, usamos la información de las direcciones obtenidas previamente para movernos en la frontera. Mientras se recorre una línea, aplicamos el umbral bajo para permitir continuar por zonas débiles hasta encontrar un punto de inicio.

Cuando el proceso se ha completado tenemos una imagen binaria donde cada píxel es marcado como perteneciente a una frontera o no perteneciente.

Los parámetros asociados al algoritmo de Canny son los siguientes:

- El tamaño del filtro gaussiano: el filtro de suavizado usado en el primer paso afecta directamente a los resultados del algoritmo de Canny. Filtros más pequeños causan menos emborronado, y suelen venir bien en imágenes sin demasiado ruido. Sin embargo, un filtro más grande permite detectar de forma más eficiente trazos largos y marcados.

- Umbrales: el uso de dos umbrales permite más flexibilidad que una aproximación con un único umbral. Un umbral demasiado grande o demasiado pequeño suele ser inútil debido a que hace perder demasiada información.

La implementación concreta utilizada por los algoritmos presentados en este trabajo consta de los siguientes pasos:

1. Discretizamos la función gaussiana mediante la función: $S(x,y,\sigma) = ke^{(\frac{-(x^2+y^2)}{2\sigma^2})}$, donde σ es la desviación estandar de la función gaussiana.El valor de k es usado para escalar toda la función de forma que todo su área sea igual a 1. Por ello $k = \frac{1}{2\pi\sigma^2}$.

2. Obtenemos una imagen suavizada $v_{m,n}$ aplicando una máscara de convolución, obtenida a partir de los datos discretizados de la gaussiana, a la imagen original $u_{m,n}$.

3. Calculamos dos gradientes, uno para cada eje de coordenadas, calculado a partir de las máscaras obtenidas mediante la derivada de la función anterior, y convolucionándolas con $v_{m,n}$: $\frac{\delta S(x,y,\sigma)}{\delta x} = \frac{-x}{2\pi\sigma^4} e^{\frac{-(x^2+y^2}{2\sigma^2}}$, $\frac{-x}{2\pi\sigma^4} e^{\frac{-(x^2+y^2}{2\sigma^2}}$.
4. A partir de ambas imágenes gradiente $gx_{m,n}$, $gy_{m,n}$, podemos obtener una imagen de direcciones y otra de magnitudes, para cada pixel se calcula: $d(x,y) = atan\frac{gy(m,n)}{gx(m,n)}$ y $m(x,y) = gx(m,n)^2 + gy(m,n)^2$.
5. Definimos dos umbrales t_l y t_h, y obtenemos una nueva imagen $a_{m,n}$ donde se cumple que todo valor de la misma es mayor a t_l y menor que t_h.
6. Suprimimos los bordes menos significativos mediante histéresis.

La imagen final de fronteras queda en $f_{m,n}$. Una peculiaridad de este algoritmo es que los bordes en forma de Y no quedan conectados por su punto central.

El algoritmo de Canny es adaptable a varios entornos. Sus parámetros permiten el reconocimiento de fronteras con diferentes características dependiendo de los requisitos particulares de una implementación dada. El rendimiento, principalmente con núcleos gaussianos grandes, puede ser lento para el procesamiento en tiempo real.

2.3. Visualización expresiva

2.3.1. Ilustración según los ilustradores

En los libros de ilustraciones se une el texto con las historias contadas. Los escritores componen historias mezclando palabras, de la misma forma, los artistas mezclan elementos visuales para crear imágenes que complementen a dichas historias. Utilizan una gramática visual que parte de los siguientes elementos:

- Líneas
- Colores
- Formas
- Texturas

Los artistas organizan estos elementos en un todo unificado para crear diseños que tengan un significado. Como hemos comentado en la sección de percepción, no todos los elementos visuales nos ofrecen una información a priori, es por ello que debemos mezclarlos siguiendo las reglas de nuestro sistema perceptual.

Podemos definir cada uno de estos elementos según el uso que hace el artista de ellos en los diferentes tipos de ilustración:

La línea

La línea es el elemento visual más básico, y también el elemento imprescindible en ilustración. Los artistas usan las líneas para sugerir movimiento, direcciones, energía y violencia. Sin embargo, también pueden significar algo estático. Por ejemplo, las líneas rectas verticales sugieren calma y estabilidad.

Podemos diferenciar entre dos tipos de líneas, según las siguientes características:

- Según su dirección tenemos:
  - Líneas rectas
  - Líneas curvas
- Según su grosor tenemos:
  - Líneas delgadas
  - Líneas anchas
- Según el trazo tenemos:
  - Líneas dentadas
  - Líneas suaves
- Según la fuerza del trazo tenemos:
  - Líneas claras
  - Líneas oscuras

Sin embargo, en la naturaleza es muy extraño encontrar elementos rectilíneos. Para representar elementos de la naturaleza se suele utilizar formas curvas o dentadas. Las líneas curvas son apreciadas como fluidos debido a su similitud a las ondas, o flujos de agua. Si son circulares y curvadas, entonces aparecen como menos definidas y predecibles.

El color

La forma más común en la que los artistas aplican las emociones en los libros de ilustraciones es mediante la línea y, sobretodo, mediante el color.

Los colores están asociados con los fenómenos naturales debido al proceso ascendente de percepción, tal y como se comentó en la sección 2.1.

Se suele tener una sensación cálida y alegre con los colores cálidos, mientras que los colores fríos suelen dar una sensación de tristeza o soledad. Los colores oscuros facilitan la sensación de lejanía, y falta de luz ambiental o sombras, mientras que los colores claros aparecen siempre como cercanos, muy iluminados o de espacios abiertos, debido al mismo proceso ascendente en la percepción.

La forma

Las líneas se suelen unir para sugerir formas y áreas de color para producir formas. Podemos diferenciar entre dos tipos de formas fundamentales:

- Formas orgánicas o irregulares: son irregulares y curvas, comunes en la naturaleza y en los objetos hechos a mano. Los ilustradores pueden usar formas irregulares para dejar a la imaginación y receptividad de la persona que observa la ilustración la comprensión de lo que está viendo.
- Formas geométricas: son exactas y rígidas. Normalmente suelen ser formadas por líneas rectas, y suelen representar elementos mecánicos. Suelen denotar complejidad y estabilidad.

La forma suele ser tan importante, si no más, que el color. La forma en su conjunto de una ilustración, la cual no tiene color, es observada como una silueta sin detalles interiores. Las imágenes irregulares dan sensación de dinamismo.

La textura

Los ilustradores manipulan los elementos visuales tales como las líneas, el color, y la forma para crear imágenes con texturas. En no pocas ilustraciones se suele prescindir incluso del color para realizar la textura, ya que, como mostramos anteriormente, el ser humano reconoce determinados patrones según la orientación de las líneas y la agrupación de elementos.

2.3.2. Herramientas para ayudar a la creación de arte

Los ilustradores eligen los medios artísticos más apropiados para las historias particulares que desean mostrar. Diferenciaremos entre las distintas técnicas y las revisaremos brevemente.

Tinta y tinta aguada

La tinta es un medio versátil para aplicar con brocha, esponja, tela o incluso con los dedos. También suele aplicarse con pluma para realizar los contornos más finos. Se ha utilizado mucho en dibujos tradicionales japoneses y en ilustraciones de libros occidentales del siglo pasado. Las ilustraciones suelen aparecer como sobrias.

Acuarelas, acrílicos, pasteles y aceites

La acuarela es una de las herramientas artísticas más usada en las ilustraciones de carácter no científico. Puede ser aplicada de diferentes formas, desde técnica aguada a pigmentos gruesos, formando manchas con relieve. El efecto de las tres herramientas de colores, acuarelas, pastel y acrílicos se pueden observar en muchas obras. Los colores aparecen brillantes y muy reales.

Grabados en madera

Los grabados en madera han sido los medios más antiguos e influyentes tanto en las culturas orientales como en las occidentales para realizar arte. Para crear estos grabados, los artistas dibujan una imagen en un bloque de madera y cortan áreas alrededor del diseño. Después mojan tinta sobre su superficie, y el artista presiona el bloque de madera sobre el papel. Las impresiones en color requieren un bloque de madera diferente para cada color en la imagen. Los grabados pueden imprimirse en colores con varios grados de transparencia, y el grano y la textura de la madera puede añadir efectos de composición.

2.3.3. Visualización expresiva

La visualización expresiva (o no fotorrealista, en oposición a las técnicas que intentan simular la física de la luz para realizar imágenes realistas), ha llegado a ser una importante rama de la informática gráfica en los últimos años. La mayoría de los trabajos referentes a visualización expresiva intentan crear mundos virtuales comparables visualmente con los

producidos por un artista humano. Sin embargo no es el único ya que engloba todo aquel proceso en el que hay un refinamiento humano de los datos antes de visualizarlos, es por ello que su objetivo difiere de la clásica visualización realista en la que se pretende simular una fotografía de alta calidad. Sin embargo, esto no es una limitación a las aplicaciones que puede tener, ya que una ilustración técnica puede llegar a ser más útil que una fotografía en muchos casos.

Los elementos más importantes en la visualización expresiva (desde el punto de vista estético) son:

- **Superficiales**: son los elementos de color y texturas.
  - **Modelos de color no realista**
  - **Texturas**
- **Líneales**: son los trazos del "dibujo".
  - **Líneas de forma**
    - **Líneas de forma externas (Siluetas)**
    - **Líneas de forma internas**

Definiremos primero informalmente algunos conceptos básicos de ilustración anteriormente mencionados:

- **Líneas de forma** (*shape lines*): Son las líneas que representa el límite entre distintos colores y matices [64].
- **Silueta** (*silhouette*): Una representación de las líneas más exteriores de un objeto[55]. Dado un observador las líneas de contorno de un objeto que representan el límite entre las partes visibles y no visibles del objeto.
- **Líneas de forma internas** (*edges* o *hightlights*): Son las líneas de forma que no son silueta.
- **Trazo** (*stroke*): Una marca o rasgo en pintura o escritura, una línea, el toque de un lápiz o pluma.
- **Forma** (*form*): La figura y estructura de algo, viene distinguida por el material del cual está compuesto, dándole individualidad o un carácter distintivo, configuración, figura, apariencia externa [55].

Aunque la mayoría de los trabajos se han aplicado sobre modelos poligonales y no sobre modelos puntuales o imágenes, los conceptos definidos son aplicables, en muchos casos a la información extraida a partir de un análisis de la imagen, es por ello que repasaremos todas las técnicas haciendo incapié en las aplicables sobre imágenes.

2.3.4. Líneas de Forma

Las líneas que delimitan la forma y figura de un objeto se convierten en una parte imprescindible en la visualización de un objeto tridimensional. Existen diferentes aproximaciones para obtener estas líneas principales, tales como las siluetas y las líneas de forma interiores. Sin embargo todas ellas pueden ser expresadas de la siguiente forma:

Definición 1. *Dado un modelo geométrico tridimensional M y un observador O, una* ***silueta*** *es el conjunto de líneas del contorno de M que representan el límite entre las partes visibles y no visibles del mismo.*

Definición 2. *Dado un modelo geométrico tridimensional M, las* ***líneas de forma internas*** *son aquellas líneas que representan el límite entre distintos colores y matices del objeto M y no son silueta.*

Prácticamente todas las técnicas de obtención de siluetas y líneas de forma se pueden clasificar en tres grandes grupos:

- **Basados en luces virtuales**
- **Basados en *buffers* geométricos**
- **Basados en algoritmos geométricos**

El modelo de iluminación sobre el cual están basadas estas técnicas es el difuso, en el cual se tiene un ángulo formado por los vectores de la normal de la superficie y la dirección de la luz. La luz decrece según el coseno de dicho ángulo [29].

Las siluetas y líneas de forma son detectadas directamente en el modelo tridimensional y de las luces similares a las luces clásicas pero que obtienen la silueta en lugar de iluminar el modelo. De tal forma que son denominadas *luces virtuales*. El problema fundamental de esta técnica es que un número elevado de luces virtuales puede afectar bastante al rendimiento del algoritmo, de igual forma, una malla compleja afecta también al rendimiento aunque obtiene siluetas más perfectas.

En una segunda aproximación tenemos las técnicas marcadas por el tratamiento de *buffers* geométricos. Un *buffer* geométrico, o simplemente un *G-buffer*, consiste en un *buffer* que contiene aspectos de la geometría del modelo, de tal forma que cada dato del *buffer* se correspondería con una determinada propiedad en un píxel del objeto visible. Mientras que un *buffer* de imagen (o *frame buffer*) contiene información del color que va a ser mostrado en pantalla, un *G-buffer* contiene elementos tales como la profundidad del objeto, que son característicos, no de la visualización del modelo, sino de su geometría. Lo normal es que el *G-buffer* contenga un resultado intermedio del proceso de visualización (*rendering*).

Un *G-buffer* habitual suele ser el *z-buffer* [81], sobre el que se suele aplicar un filtro de detección de fronteras clásico de primer orden (es común el uso de una convolución con una matriz de Sobel), el resultado de dicho filtro se introduce de nuevo en el frame *buffer*, obteniendo las siluetas del mismo (ver figura 2.12).

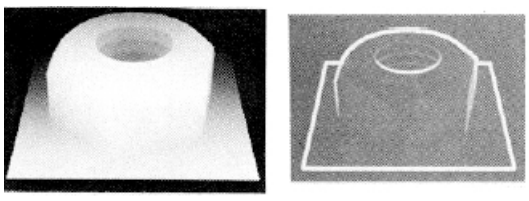

Figura 2.12: La parte izquierda muestra el contenido del *z-buffer* (a), mientras que la parte derecha (b) muestra el resultado tras aplicar un operador diferencial de primer orden

De esta forma, el proceso se aplica realmente sobre superficies bidimensionales, desarrollando algoritmos que trabajan sobre puntos. Un G-Buffer contiene el identificador del objeto, las coordenadas *u*, *v* del parche, las coordenadas de profundidad (coordenadas de pantalla, por ejemplo, del z-buffer), las coordenadas en el mundo o las normales.

Sin embargo el usar un único operador, como el de diferencial de primer orden produce los siguientes efectos no deseados:

- Es difícil detectar las discontinuidades de cambios continuos grandes.
- La oscuridad de las líneas obtenidas depende del grado de la concavidad.
- En los operadores de segundo orden, las discontinuidades de orden 0 se extraen como líneas dobles.

Para corregir estos defectos del algoritmo se propone el uso de operadores de normalización de segundo orden, aunque se han de corregir entonces los fallos mediante otra función, la cual es útil solamente para las imágenes sin ruido obtenidas del *buffer* geométrico. En este tipo de técnicas se suelen representar las siluetas en un paso posterior, modificando el trazo. El problema principal viene dado por la limitación de la precisión del *G-buffer* concreto que se utilice.

En la última aproximación se utilizan diferentes algoritmos que utilizan directamente la geometría del modelo, las aproximaciones suelen ser muy distintas unas de otras, normalmente se basan en la definición de una estructura, normalmente denominada *buffer* de aristas (o *edge buffer*), en la cual se almacenan determinados aspectos de la geometría, tales como vértices y se definen operaciones que determinan que aristas son siluetas y cuales no [14, 89].

2.3.5. Estilos

Normalmente las siluetas no se muestran de un solo píxel de grosor debido al aspecto artificial con el que aparecen, los ilustradores técnicos y artistas conocen de la expresividad de las líneas como el mayor elemento visual básico. Los artistas usan las líneas para sugerir, como hemos comentado en la sección anterior:

- **Dirección**.
- **Movimiento**.
- **Energía**.
- **Emociones**.

Además existen diferentes tipos de materiales y métodos para representarlos que se ajustan mejor a determinados tipos de dibujos. Existen trabajos en los cuales se desarrolla la simulación de un lápiz, pincel o pluma sobre un determinado papel. En otros, simplemente se intenta aproximar el trazo visible al que efectúan los ilustradores.

En los aproximados tenemos los métodos como el de Saito [81], en el cual los contornos se obtienen a partir de cálculos sobre cada píxel y sus vecinos, de forma puntual y no sobre polígonos. Como ventaja tenemos un método simple que no necesita conocer de la correlación espacial entre distintas fronteras, como desventaja, que el método obtiene unos tipos de contornos y trazos muy limitados, y relacionados directamente con la imagen obtenida, de tal forma que las imágenes aparecen como ilustración técnica debido a los trazos rectos.

Por otro lado, la simulación de las herramientas usadas por el artista se basa en considerar el papel como un conjunto de celdas (*voxels*), las cuales pueden estar manchadas o no por el elemento de dibujo en cuestión, la simulación se realiza a nivel casi microscópico [92].

Las características más frecuentemente simuladas han sido la estructura microscópica del papel, la redistribución del pigmento sobre el papel y la redistribución del pigmento en el interior del papel.

Otras simulaciones añaden las características básicas del papel y la extensión del pigmento sobre el mismo [27].

En el lápiz, se suele tener una serie de parámetros que simulan las propiedades del lápiz, sin embargo, en técnicas como la aguada el pigmento es expandido debido al agua y no se puede simular de forma sencilla. La presión de la brocha es también un factor importante a la hora de aplicar este tipo de técnicas.

Todo esto se simula mediante una convolución lineal integral, que es un filtro que transforma los valores de los píxeles a lo largo de la línea obtenida a través del campo vectorial (ver el capítulo 2). Para el papel, se utiliza una serie de discos (intentando simular el talco), que son situados de forma aleatoria.

Para la técnica de aguada se usa una brocha con parámetros de ancho y grosor, las coordenadas de inicio y fin del trazo son muy importantes ya que dependiendo de ello se modificará el tamaño del mismo.

Esta técnica tiene la ventaja de la alta calidad en las imágenes producidas mediante pinceles electrónicos, el problema es la complejidad del modelo y el tiempo empleado en la computadora para realizar la simulación.

2.3.6. Trazos

Las técnicas más comúnmente utilizadas para realzar las siluetas y líneas de forma suelen basarse en el uso de cadenas de polígonos texturizados. Hay que tener en cuenta que cada técnica tiene sus propias formas de aplicarse. Nos centraremos por tanto en las más utilizadas para ilustración, que son el uso del lápiz y la tinta.

Con tinta se procede a la simulación, pero esta vez lo que se simula no son las propiedades físicas del lápiz sino que se simulan los trazos más comunes en un artista. Normalmente se aplican una serie de variables asociadas a cada uno de los ángulos y vectores que conforman los distintos trazos y se generan aleatoriamente para generar la textura que se incluye en el trazo [88], también se suele simular la absorción del papel en este punto[82, 41] (ver imagen 2.13).

Para crear los trazos básicos en tinta, se suele tener una lista de vértices obtenidos en un proceso anterior [11] (ya sea mediante extracción en una imagen o de algoritmos de obtención aristas en modelos poligonales). Estos vértices pueden poseer un atributo asociado que sea el grosor.

En lápiz, el primer paso suele ser ampliar el trazo original mediante vectores perpendiculares a la dirección del mismo [75]. A partir de estos vectores se construye un conjunto de tiras de polígonos con el ancho determinado mediante el grosor dado.

Escalando los vectores perpendiculares a la dirección del trazo obtenemos un trazo diferente en grosor si se aplica uniformemente, o bien, con diferente estilo (sinuoso por ejemplo) si el escalado no se aplica de forma uniforme.

Además se requiere que los vértices adyacentes no excedan en cuanto a distancia los dos o tres píxeles . Lo que se hace es interpolar una curva entre cada uno de los vértices (nor-

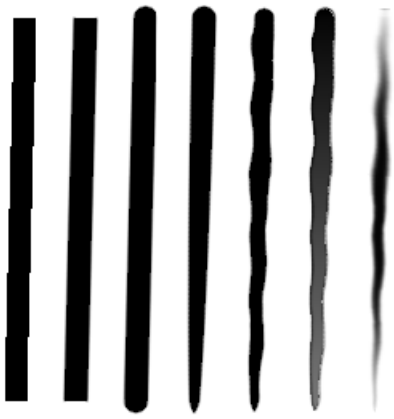

Figura 2.13: Representación de varios trazos de tinta colocados como textura en un conjunto de polígonos.

malmente mediante un *spline* lineal), lo que hace que la curva pase por los puntos exactos y aparezca suavizada. Una vez que se tiene una muestra de los vértices suficientemente suavizada, lo que se hace es alterar sus localizaciones y su anchura, según la ecuación anterior. Esto permite que el trazo sea similar al que un artista usaría.

2.3.7. Materiales

Las texturas pueden realizarse simulando los trazos del artista, y no suele ser demasiado común salvo para el caso de las imágenes digitales. Es por ello que analizaremos detenidamente estas técnicas en la siguiente sección.

En cuanto a las técnicas utilizadas para alterar el trazo de forma que aparezca el material realizado con diferentes técnicas, destaca el trabajo de Secord [86]. En el que se realiza una variación del estilo y dirección de los trazos según la distribución del tono de los polígonos iluminados en el espacio de la imagen.

En este trabajo, al igual que otros [82], se define una función probabilística de densidad.

En estos trabajos se requiere satisfacer una serie de propiedades, en concreto, que la probabilidad de uno de los píxeles esté en el conjunto sea la misma que la intensidad de dicho píxel , además de que satisfaga la definición de la función de densidad probabilística.

Para crear la apariencia de un material, se utiliza una distribución de Poisson. Si se desea otro tipo de material, se modifica la distribución aleatoria por otra distinta, el algoritmo funciona de la misma forma. La distribución de Poisson tiende a formar pequeñas agrupaciones de puntos que están relativamente cercanas.

Según algunos autores, una paleta discretizada ayuda a producir ilustraciones menos estilizadas [86]. Esto, como veremos en la siguiente sección, no es algo general, sino particular de la técnica utilizada.

La desventaja que tenemos con estas técnicas es que en imágenes complejas con ruido no podemos obtener una imagen clara. Además no podemos simular elementos que no sean estáticos, por ejemplo, el agua o el fuego es difícil (si no imposible) de representar mediante estos algoritmos.

Capítulo 3

Cambio en la iluminación en volúmenes

En este capítulo se repasarán algunos trabajos previos muy relacionados con la definición de un nuevo método de iluminación basado en el espacio de color HSV. Además este sistema de iluminación se basa en un método previo de clasificación muy sencillo, basado en el mismo sistema de color y en el funcionamiento de la visión humana.

Introducción

El cambio en la función de iluminación se ha venido empleando durante muchos años en polígonos. Sin embargo tan solo recientemente se ha comenzado a aplicar cambios de coloración parecidos a los que una fuente de luz puede hacer sobre un modelo poligonal. Primero analizaremos algunos trabajos previos muy relacionados para posteriormente analizar con detalle nuestro modelo de iluminación.

3.1. Cambio en la función de iluminación en modelos poligonales

La iluminación basada en el sombreado de Phong no provee información tan rica como la que se ofrece en la ilustración, en la cual se representan fronteras, además de un sombreado ligero que permite diferenciar la forma fácilmente. Es por ello que en ilustración los tonos son muy poco importantes con respecto a la forma.

Entre los diversos trabajos de ilustración realizados sobre polígonos [36] merece la pena comentar el trabajo de Gooch[34] debido a la sencillez del método y a la alta calidad de los resultados obtenidos.

3.1.1. Modificación de la luz de Lambert basada en el tono

Lo primero que se realiza en este método es detectar las líneas y fronteras, además, se necesita introducir las sombras al objeto. La iluminación difusa clásica se basa en la proporción del coseno del ángulo entre la dirección de la luz y la normal de la superficie:

$$I = k_d k_a + k_d \max(0, \hat{I} \cdot \hat{n}) \tag{3.1}$$

donde I es el color en un espacio de color RGB mostrado en un determinado punto de la superficie, k_d es la reflectancia difusa en el punto, k_a es la iluminación ambiente RGB, $\hat{I}$ es el vector unitario en la dirección de la fuente de luz, y $\hat{n}$ es el vector normal a la superficie en el punto.

En este trabajo se modifica la ecuación de forma que se adapta a un gradiente. Para añadir las siluetas a la imagen no se pueden añadir de forma directa ya que en las zonas demasiado claras u oscuras se perderían (ver figura 3.1). Para evitarlo se hace uso algunas de las siguientes heurísticas:

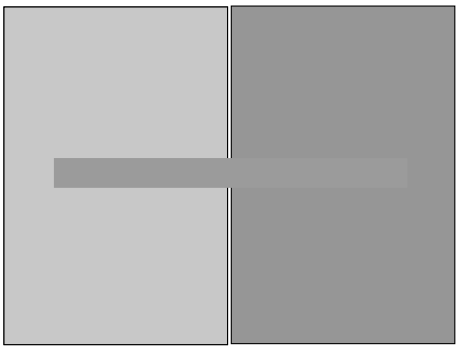

Figura 3.1: Como podemos comprobar una misma silueta no puede ser usada en cualquier tono de fondo.

Elevamos k_a hasta que sea lo suficientemente grande (se entiende que hasta que no haya una diferencia visual suficiente grande con respecto a la silueta negra seguimos aumentando el parámetro). El resultado tiene como efecto secundario una pérdida de los detalles más pequeños.

Otra segunda heurística que se puede aplicar es utilizar una segunda fuente de luz, la cual añade brillos y sombras adicionales. Para hacer que los brillos aparezcan en lo alto del sombreado, podemos bajar k_d hasta que sea visualmente distinto del blanco.

Estos métodos se aproximan a una ilustración técnica pero no consigue realzar la imagen de forma suficiente, perdiendo detalle. Para evitarlo se utiliza un sombreado basado en el tono para objetos de color mate. En un medio con colores como la brocha aguada, el bolígrafo o la pluma, los artistas usan desplazamientos tanto en el tono como en la luminancia. Añadiendo negros y blancos a un color consiguen resultados que los artistas denominan sombras en el caso de los negros y luces en el caso de los blancos. Cuando las escalas de color se crean añadiendo gris a un cierto color, entonces se denominan tonos (ver figura 3.2). De esta forma existe variedad de tonos sin variar demasiado la luminancia.

Cuando el complementario de un color se usa para crear una escala de color se llaman también tonos. Los tonos se consideran un concepto crucial en la ilustración y son especialmente útiles cuando los ilustradores están restringidos a un pequeño rango de luminancia.

Otra cualidad del color usado por los artistas es la temperatura del color. La temperatura de un color están definidos como una clasificación de los tonos: calientes (rojo, naranja y amarillo), fríos (azules, violetas y verdes), o templados (violáceos rojizos y amarillos verdosos). La profundidad del tono viene dada por las características de percepción humana (ver capítulo 2). Además los colores en un objeto cambian de temperatura en las imágenes soleadas porque los fríos y templados varían en una contribución relativa a través de toda la superficie. No solamente depende de la temperatura de un tono sino que también de las relaciones por proximidad de otros tonos, o lo que es lo mismo, el contraste simultáneo. Todas estas técnicas se utilizan en el trabajo de Gooch[34].

Se puede generalizar el modelo de sombreado clásico para informática gráfica con la ecuación siguiente:

$$I = (\frac{1+\hat{l}\cdot\hat{n}}{2})k_{cool} + (1-\frac{1+\hat{l}\cdot\hat{n}}{2})k_{warm} \tag{3.2}$$

El coseno del ángulo formado por $(\hat{l}\cdot\hat{n})$ es el mismo utilizado en el sistema de iluminación

difuso clásico y varía entre $[-1,1]$. Para asegurarnos que la imagen muestra la variación completa, el vector $\hat{l}$ debería ser perpendicular a la dirección del observador. Debido a que el sistema de visión humano asume que la iluminación viene de una posición más alta, se debe elegir una posición de la luz en la parte superior del modelo y a la derecha y mantener esta posición constante.

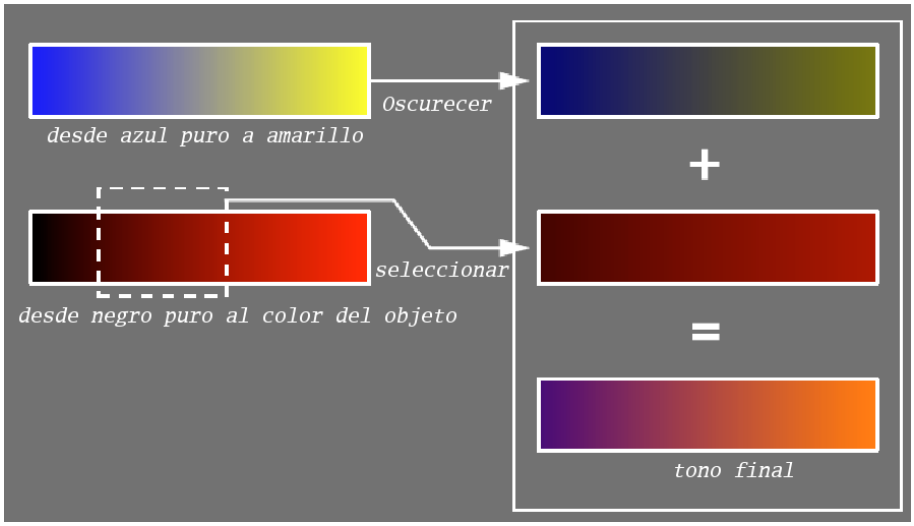

Figura 3.2: Diferentes gradientes de tonos.

Para automatizar la técnica de desplazamiento del tono y añadir alguna variación a la luminancia podemos examinar dos posibilidades extremas de generación de escala de colores (siempre teniendo en cuenta el modelo perceptual humano): se han elegido los tonos azules y amarillos para la parte correspondiente a una gama de colores que va de los fríos a los templados. Este rango de colores se puede expresar como un intervalo entre las siguientes variables: $k_{blue} = (0,0,b), b \in [0,1]$ en el espacio RGB, y $k_{yellow} = (y,y,0), y \in [0,1]$. Esto produce una imagen muy poco natural o realista, y es independiente de la reflectancia k_d.

Si eligiéramos $k_{cool} = (0,0,0)$ y $k_{warm} = k_d$ lo que obtenemos es una imagen muy parecida a la iluminación difusa clásica. Por ello ha de elegirse un compromiso entre ambas estrategias.

Se puede modificar las ecuaciones para que los tonos aparezcan escalados, combinando pigmentos. Para ello se puede mezclar la combinación de los tonos azules/amarillos con los negros y el color del objeto:

$$k_{cool} = k_{blue} + \alpha k_d \tag{3.3}$$

$$k_{warm} = k_{yellow} + \beta k_d \tag{3.4}$$

Siendo b, y, α y β cuatro parámetros libres para ser modificados. Los valores de b y y determinarán la cantidad de desplazamiento de la temperatura. Y los valores de α y β determinarán la cantidad de color y luminancia del objeto. Este modelo es apropiado para un modelo con un único material, siendo totalmente inapropiado para modelos con muchos materiales.

3.2. Visualización expresiva en volúmenes para realce

No existen demasiados trabajos en cuanto a realce de volúmenes modificando la iluminación, los más parecidos modifican la función de transferencia del volumen [80, 28] en los cuales se hace una modificación de tono por profundidad y otra para el sombreado. Comentamos estos trabajos más en profundidad en las siguientes secciones.

3.2.1. Cambio de tonos según la profundidad y orientación

En la visualización tradicional de volúmenes translúcidos se presentan unos pocos tonos de profundidad. Faltan diferencia entre los tonos ya que no existen objetos opacos para mostrar un claro orden de profundidad. La principal dificultad se encuentra en encontrar una función de transferencia que incremente la opacidad en rangos de valores de interés. Mientras que esto puede incrementar la profundidad de los tonos creando la apariencia de superficies dentro de un volumen, también es cierto que oculta toda la información en algunas partes del volumen, perdiendo de esta forma toda ventaja de la visualización de volúmenes.

De forma similar, falta la información sobre las características del volumen. Muchos sistemas de visualización de volúmenes usan unos modelos de iluminación muy simples y normalmente no incluyen efectos de sombreado, aunque se han desarrollado algunos algoritmos más sofisticados [49, 43]. Sin embargo, no producen una gran diferenciación entre la profundidad de los objetos. Sigue siendo, por tanto, un modelo realista que no suele ser válido para el diagnóstico ya que es difícil percibir detalles en el modelo. Se han desarrollado diferentes técnicas de cambio de tonos profundidad y orientación en los modelos volumétricos, inspirados en conceptos artísticos y en la ilustración técnica.

El cambio de intensidad del tono utilizado según la profundidad del elemento visualizado es una técnica muy conocida y utilizada [29]. Esta técnica cambia el color de los objetos que están lejanos del observador, creando un efecto similar a ver la escena a través de una neblina. Ebert ha adaptado la técnica anterior de forma que en lugar de la intensidad, lo que se modifica es el tono. Además se ha añadido una función de cambio de intensidad y tono por profundidad con una modulación ligera del color. Esta modulación del color incrementa la cantidad de azules en los colores de las muestras más distantes, simulando una técnica utilizada hace siglos por pintores como Leonardo Da Vinci, y otros. Esta técnica se basa en la sensación perceptual de los colores, y en concreto la tendencia del azul a ser un color frío y distante.

El cambio de tonalidad por profundidad empieza en el color del *voxel* en la parte frontal del volumen, decrementando en intensidad y moviéndose hacia los colores de la parte del fondo mientras se incrementa la profundidad. La progresión en el cambio de tono no es linear, si no que se usa una función exponencial para controlar la aplicación del cambio de tono en profundidad. Los colores más distantes se mezclan con los más cercanos mediante la siguiente expresión:

$$c_d = (1 - k_{ds} d_v^{k_d e} c_v + k_{ds} d_v^{k_{dv}} c_b) \tag{3.5}$$

donde k_{ds} controla la cantidad de color afectado por el proceso de mezcla (*blending*), d_v es la fracción de la distancia a través del volumen, y c_b es definido como el color del fondo como un gradiente de grises $c_b = (a, a, a)$ para algún valor de a. El color de fondo es un gradiente de azules, $c_b = (a, b, c)$ para $c > a$, esto introduce un desplazamiento en las regiones distantes.

3.2.2. Iluminación basada en tono

Otra técnica muy usada en ilustración es la modificación del tono de un objeto basándonos en la orientación de un objeto relativa a la luz. Esta técnica puede usarse para dar a las superficies mirando en dirección a la luz una tonalidad templada mientras que al resto se le modifica el color para tener una tonalidad fría. Ebert utiliza una técnica de definición de tonos similar a la que usa Gooch en su trabajo.

De esta forma, la contribución de los tonos está formado por la interpolación entre los templados y los fríos basado en el signo del producto escalar entre el gradiente del volumen y el vector de la luz. A diferencia del método de sombreado de Gooch, el objeto iluminado se usa únicamente utilizando la parte positiva del producto escalar, siendo cero en las posiciones ortogonales. El color de un *voxel* es calculado por la suma de los colores gaseosos (incluyendo cualquier cálculo tradicional de la función de transferencia), el tono y el sombreado de todas las luces direccionales. El nuevo modelo de iluminación es dado por la siguiente expresión:

$$c = k_{ta} I_G + \sum_{i}^{N_L} (I_t + k_{td} I_o) \tag{3.6}$$

donde k_{ta} controla la cantidad de iluminación gaseosa (I_G) incluida, N_L es el número de luces y k_{td} controla la cantidad de luz directa, I_t es la contribución de los tonos a una muestra de color, y I_o es el objeto iluminado mediante una contribución de color. Este modelo permite múltiples fuentes de luz, demasiadas luces pueden resultar en una imagen imprecisa y difícil de interpretar. La contribución de los tonos de una fuente de luz simple es interpolada desde los colores templados a los fríos, dependiendo del ángulo entre el vector de la luz y la muestra del gradiente. Viene dada por la siguiente expresión:

$$I_t = ((1{,}0 + \nabla_{fn} \cdot L)/2) c_w + (1 - (1{,}0 + \nabla_{fn} \cdot L)/2) c_c \tag{3.7}$$

donde L es el vector unitario en la dirección de la luz y $c_w = (k_{ty}, k_{ty}, 0)$, $c_c = (0, 0, k_{tb})$, describen lo templado o frío que son los tonos de los colores. Los componentes de iluminación directa vienen dados por el ángulo entre el gradiente del *voxel* y la dirección de la luz, para ángulos mayores a 90 grados. Viene dado por:

$$I_o = \begin{cases} k_{td} I_i (\nabla_{fn} \cdot L) & \text{si } \nabla_{fn} \cdot L > 0 \\ 0 & \text{si } \nabla_{fn} \cdot L \leq 0 \end{cases} \tag{3.8}$$

Podemos observar en la figura 3.3 los resultados obtenidos mediante el método propuesto por Rheingans y Ebert. En la figura 3.4 tenemos la imagen original tras mezclar un filtro de Sobel. Podemos comprobar que obtenemos mejores resultados con un análisis sencillo de la imagen que con estos métodos propuestos.

3.3. Iluminación basada en el modelo de color HSV

3.3.1. Introducción

En este capítulo se propone una nueva aproximación a la visualización de volúmenes: una visualización no basada en la física de la luz sino en técnicas expresivas. Más concretamente en la definición clásica de siluetas y líneas de forma para modelos poligonales [25, 64] pero,

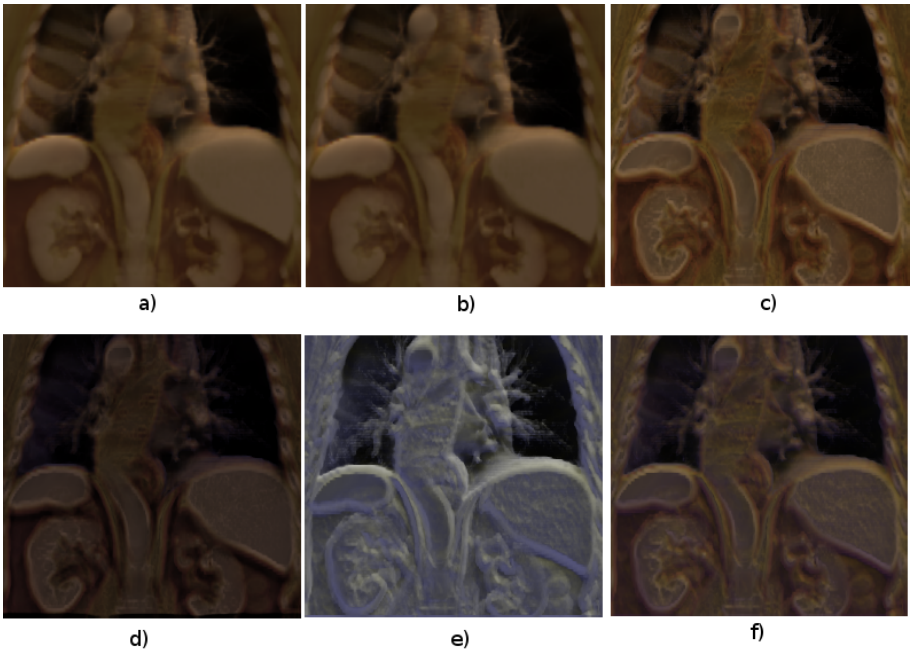

Figura 3.3: a) Iluminación simple sobre un modelo de tomografía axial; b) Iluminación modificando el color y con realce de fronteras; c) Realce de fronteras y siluetas sobre el modelo anterior; d) Cambio de color según la distancia; e) Cambio de tono, las superficies que se dirigen hacia la fuente de luz aparecen con colores templados; f) Cambio de tono mezclado con la iluminación original.

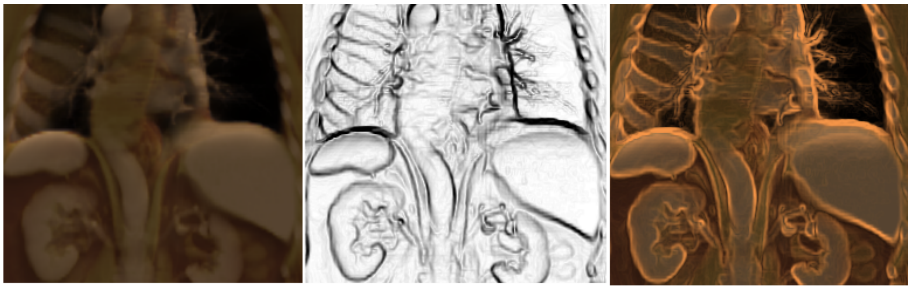

Figura 3.4: A la izquierda imagen iluminada de forma sencilla mediante una técnica clásica de iluminación (Lambert + Phong); en el centro un filtro de Sobel sobre la imagen izquierda; a la derecha resultado de mezclar la imagen central (cambiando el color de los bordes de negro a naranja) y la imagen de la izquierda; los resultados son mejores que los obtenidos por los métodos anteriores.

en este caso, modificadas para una visualización directa de volúmenes y aplicadas sobre un modelo de color HSV[29].

Nuestro método ha sido desarrollado pensando en aplicaciones médicas aunque no son

las únicas a las que es aplicable, sin embargo, los parámetros que hemos probado en este capítulo se ajustan mejor sobre este tipo de modelos.

3.3.2. Aproximación al método

Nuestro método ha sido probado con datos volumétricos procedentes de una serie de imágenes obtenidas mediante tomografía axial (TAC) e imágenes en color del Visible Human Project. Por tanto, la entrada al algoritmo puede ser tanto imágenes en color (RGB) como en escala de grises diferenciándose el algoritmo tan solo en el sistema de clasificación. El cauce general del algoritmo se muestra en la figura 1, los pasos que se encuentran en gris indican lo que el algoritmo incluye como novedad.

En el paso de clasificación de tono lo que se intenta realizar no es un paso de segmentación, sino de elección de tono para cada uno de los distintos valores que tenemos en el modelo tridimensional, mientras que en el paso de iluminación no fotorrealista se ajusta la saturación y el valor de los colores en un modelo HSV de color.

3.3.3. Justificación de la clasificación por tono

Para comprender mejor esta idea es conveniente conocer qué tonos son usados por un artista cuando dibuja partes del ser humano. Los colores de la piel y carne de una persona varían profundamente según la localización del cuerpo que estemos pintando, sin embargo, no se trata tanto de elegir el color más cercano a la realidad, sino el elegir tonos en los que haya una gran diferencia visual.

Mientras que para la pintura tradicional han sido usados desde siempre una mezcla de rojos, amarillos y colores fríos tales como el verde o el azul, para los medios digitales no podemos utilizar las mismas mezclas pero la idea básica es similar, de tal forma que cuando se realiza una pintura clásica utilizando un medio digital los artistas suelen utilizar colores tales como naranjas, amarillos y rosas fríos [87].

La piel suele actuar como una superficie blanca en su interacción con el entorno que le rodea, es por ello que no deberíamos tenerla en cuenta para nuestra aproximación debido a que partes internas del modelo serían mezcladas con las manchas de grises producidas por la piel, además hay que tener en cuenta que existe una ausencia de tono para toda la gama de grises incluyendo el blanco y el negro como tales.

Para las partes internas del cuerpo se deben usar tonos más rojos para ofrecer más realismo, sin embargo la prioridad no es tanto el realismo como que haya una gran diferencia entre los tonos de las partes de interés y el resto de tonos. Para los modelos orgánicos los artistas suelen aconsejar el uso de colores más fríos que los que en la realidad suelen tener [87]. Por todo ello debemos intentar utilizar colores que una persona sea capaz de distinguir mejor que el resto para las zonas de interés intentando usar los tonos más fríos de la zona de los naranjas y de los amarillos en el círculo cromático.

Así mismo, todos los tonos pueden reaccionar entre sí, sin embargo, igual que el efecto que produce el mezclar un tono con otro de forma arbitraria puede ser desastroso en una obra de arte, en los modelos volumétricos donde la cantidad de información visual es sustancialmente mayor puede originarnos unos resultados del todo inaceptables.

Algo a tener en cuenta es que todos los colores se ven afectados por los que los rodean, la influencia que producen estos valores es denominada *contraste simultáneo* [87], la cual hemos explicado anteriormente en el capítulo 2.

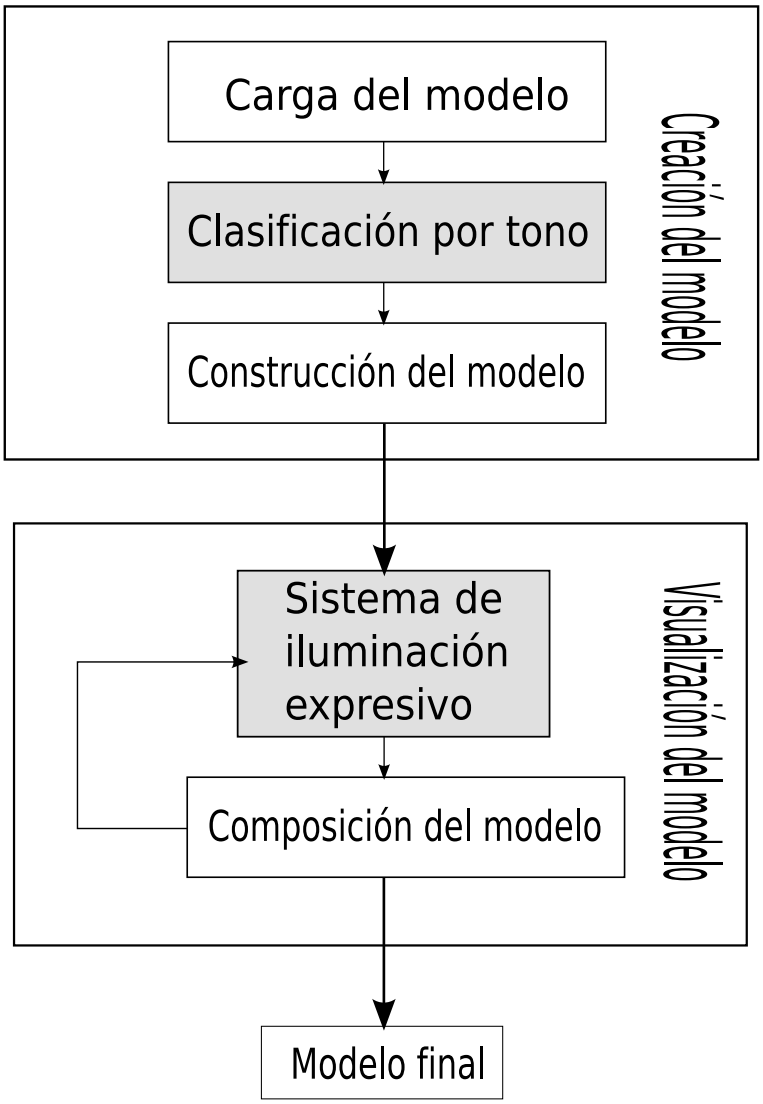

Figura 3.5: Cauce de procesamiento general del método.

Es por ello que hay que evitar el usar colores que estén a poca distancia en el círculo cromático (ver figura 3.6). La distancia al color complementario viene indicado por el parámetro A, colores demasiado cercanos pueden no distinguirse, hay que dejar una distancia indicada por B para poder elegir el siguiente tono, y viene determinada por la ecuación de contraste simultáneo.. Para los elementos que queremos diferenciar es aconsejable usar colores complementarios o colores distanciados en el círculo cromático por la ecuación del contraste simultáneo.

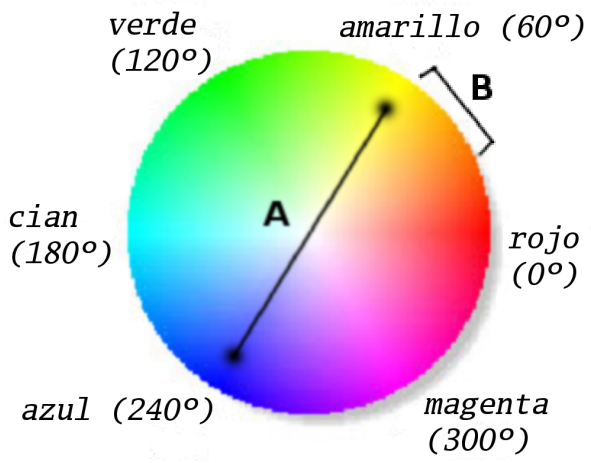

Figura 3.6: Círculo de tonos según el modelo HSV ($v = 1$).

3.3.4. Clasificación por tono

Para los modelos obtenidos a partir de imágenes en color se pueden seguir dos tipos de aproximaciones:

1. Transformar las imágenes en color en imágenes en escala de grises antes de construir el modelo

2. Utilizar un sistema de clasificación previo basado en el tono del color real

En la primera aproximación el valor del gris es tomado como el índice del material asociado, esta aproximación es buena si las zonas de interés vienen dadas por la luminosidad de la imagen, lo cual no suele ser muy común, en la segunda aproximación el algoritmo utilizado para determinar el índice del material es el siguiente:

```
01. Para cada color (c) asociado al punto P:
02.    Transformamos c de RGB a HSV
03.    Sea h el tono asociado a c (componente H)
04.    Tomamos h como valor del material de este punto
```

De esta forma tendremos un índice del material por cada dato que hemos obtenido de las imágenes. En las imágenes basadas en tonos de grises suele ser una buena aproximación el tomar directamente el valor del gris como índice.

Para cada uno de estos índices se define un material asociado que no será más que un tono. Este tipo de clasificación permite que las zonas parecidas tengan un valor de índice de material cercano entre sí y las alejadas un valor lejano.

Una vez seleccionado este tipo de clasificación previa se procede a la selección de tonos, para ello se debe identificar a mano las zonas de interés. Por ejemplo, en los modelos que hemos utilizado obtenidos a partir del TAC los huesos tienen un valor de índice superior a 130 y serán nuestra zona de interés, mientras que la carne, la piel y los órganos se encuentran en los valores inferiores a ese umbral. Lo ideal sería definir un tono a la zona de interés

y otro al resto. Sin embargo, si asignásemos un único tono los cambios de densidad no se apreciarían.

La solución se encuentra en definir intervalos en el círculo de tonos (ver figura 3), de tal forma que entre el 0 y el 130 estén los valores de H correspondientes a los naranjas y entre el 130 y el 255 los correspondientes a los azules o verdes.

Este es el único proceso manual de todo el algoritmo, ya que una vez definido el intervalo de color para cada índice localizado en la zona de interés tan solo tenemos que escalarlo. La ecuación utilizada para ello es la siguiente:

$$\Delta_z = \frac{T_{fin_k} - T_{ini_k}}{R_{fin_z} - R_{ini_z}}$$

Donde T_{ini_k} y T_{fin_k} son los valores de tonos elegidos como inicio y fin del intervalo de tonos dado por k, R_{ini_z} y R_{fin_z} son los valores de los índices elegidos como inicio y fin del intervalo de la zona de interés o cualquier otra zona elegida, que viene dada por z. Δ_z es el incremento de tono (H) para la zona de interés z. La parte del denominador siempre es un número entero debido a que, tanto R_{fin_z} como R_{ini_z} son enteros entre 0 y 255. Cuando $R_{ini_z} = R_{fin_z}$ la expresión utilizada es la siguiente:

$$\Delta_z = T_{fin_k} - T_{ini_k}$$

El algoritmo final queda, por tanto, de la siguiente forma:

```
01. Para cada color (c) asociado al punto P:
02.    Transformamos c de RGB a HSV
03.    Sea h el tono asociado a c (componente H)
04.    Tomamos h como valor del material de este punto
05.    Determinamos la zona z a partir del índice obtenido h
06.    Obtenemos h'
```

donde $h' = (h - T_{ini_k}) \cdot \Delta_z + R_{ini_z}$. Como puede apreciarse con el algoritmo, los colores quedan modificados con el objetivo de mejorar la percepción de las diferentes zonas de interés.

3.3.5. Definición del modelo de iluminación

Hasta ahora sólo ha sido considerado el tono (H) en la terna HSV, pero aún podemos modificar la saturación y el valor (o intensidad) para obtener diferentes efectos. Nuestro algoritmo se basa principalmente en la fuente de luz no fotorrealista que describiremos a continuación, pero antes justificaremos el porqué de la decisión tomada.

Justificación del uso de la saturación y el valor

A parte del tono es importante el valor en un color debido a que muestra cómo de iluminado u oscuro aparece (ver figura 3.7). Es decir, que indica cómo está iluminado este color. Los valores altos en los colores han sido utilizados de forma clásica para simular la iluminación de una fuente de luz y para remarcar las zonas importantes de una imagen.

La saturación sin embargo, es la intensidad del tono, es decir, cómo de gris es el color (ver figura 3.8). Hay que tener en cuenta que es también otra forma de representar la luminosidad de la imagen en función de la cantidad de color que hay. En los procesos artísticos se utiliza junto al valor para simular efectos de iluminación en fuentes de luz.

Figura 3.7: Cambios en el brillo

Figura 3.8: Cambios en la saturación

Es por todo ello que el contraste entre diferentes tonos es una acción simultánea del valor y la saturación sobre varios colores que están en una posición cercana, también son las causantes de que diferentes gradientes aplicados sobre un objeto con diferentes valores y saturaciones sobre un mismo tono aparezcan como una fuente de luz que realmente no existe, como aparece en la figura 3.9.

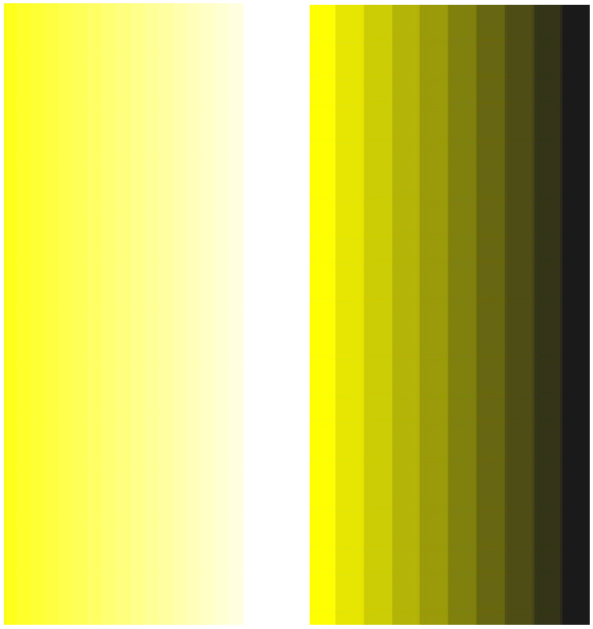

Figura 3.9: Cambios en la saturación o el brillo producen un efecto de sombreado parecido al sombreado difuso clásico.

Definición de la nueva luz

Un modelo de iluminación ampliamente utilizado ha sido el Lambertiano, en el cual, la iluminación depende de los siguientes factores:

- La normal de la superficie del objeto: es muy importante debido a que nos indica la forma del objeto que estamos iluminando.
- La dirección de la fuente de luz: también muy importante debido a que nos indica de donde proviene la luz.
- La distancia a la fuente de luz: un factor basado en la física de la luz, si se omite el objeto aparece sobreiluminado.
- Una constante: basada en la propiedad del material del objeto que se ilumina.

A este modelo le suele ir acompañado el modelo de Phong que básicamente añade el brillo que se produce en materiales semiespeculares basados en la posición del observador. Ambos tipos de luz se aplican normalmente sobre cada una de las componentes RGB de un material en un punto del objeto.

La propuesta que aquí se presenta es crear un nuevo tipo de luz, no basado en el sistema físico que tiene como fundamento estos dos tipos de modelos, sino en el funcionamiento del modelo de color HSV, en concreto en la saturación y en el valor.

Puesto que la dirección de la fuente de luz es importante para determinar de dónde procede la luz, y las normales del objeto son imprescindibles para obtener la forma podemos realizar el producto escalar de ambas obteniendo un gradiente basado en la saturación para un modelo cualquiera. Esto es, sea $\vec{L}$ la dirección de la fuente de luz I_i, $\vec{N}_j$ la normal en un punto dado $\vec{P}_j$.

El resultado de iluminar el punto $\vec{P}_j$ con la fuente de luz I_i es el valor S_j determinado por: $S_j = \vec{L}_i \cdot \vec{N}_j$, que es equivalente a calcular el coseno entre el ángulo formado por estos dos vectores. Para los valores negativos basta con realizar el valor absoluto, debido a que la saturación está definida entre 0 y 1, y es deseado que los puntos traseros queden también iluminados.

Si examinamos el modelo Lambertiano no hay ninguna diferencia, salvo en que aquí el valor es asignado, no a una componente de color, sino a la componente de saturación, que la usamos para sombrear los objetos y así darles volumen.

En cuanto al valor, si observamos la figura 3.7 el valor de luminosidad afecta a la oscuridad del objeto, esto puede ser útil para marcar las fronteras de un objeto, es decir, se puede utilizar los valores más bajos para marcar las zonas donde la normal del objeto está más alejada de la dirección del observador, mientras que aquellas que tengan una normal similar a la dirección del observador pueden tener un valor más alto. El efecto de esto es, si observamos la figura 3.10, que las zonas frontera del objeto aparecen oscurecidas, marcando levemente las siluetas de cada uno de los objetos de la escena y realzando las zonas que están mirando de frente hacia el observador.

Esto es, sea $\vec{C}$ la dirección de la cámara O, $\vec{N}_j$ la normal en el punto $\vec{P}_j$. El resultado de iluminar el punto $\vec{P}_j$ con una fuente de luz L_k, con la cámara O apuntando en la dirección de $\vec{C}$ es el valor V_j determinado por: $V_j = \vec{C} \cdot \vec{N}_j$, que es equivalente a calcular el coseno entre el ángulo formado por estos dos vectores. Para los valores negativos, igual que en la saturación y por la misma razón anterior, podemos realizar un valor absoluto.

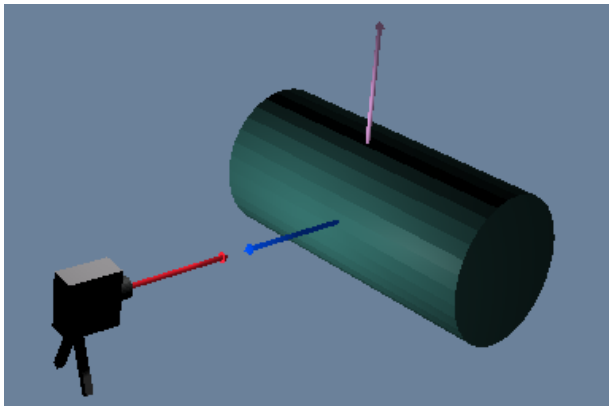

Figura 3.10: Vemos como la normal representada en azul está muy cercana (en ángulo) a la dirección de la cámara y está más clara (un valor mayor), mientras que la normal en rosa está alejada (en ángulo) de la dirección de la cámara y por ello está más oscura (un valor menor).

Por tanto, el tono (H) queda modificado por el material del punto, y permanece invariable por muchas fuentes de luz que se definan. La saturación (S) depende de las fuentes de luz y produce un gradiente similar al modelo de iluminación Lambertiano, mientras que el valor (V) produce un factor de oscurecimiento realzando las zonas de interés con respecto a la cámara.

3.3.6. Resultados

Todos los resultados obtenidos de este método han sido obtenidos mediante un algoritmo de visualización mediante texturas 2D, aunque el método no ha de modificarse para obtenerse mediante un trazado de rayos, texturas 3D u otro tipo de visualización. Para el cálculo de normales ha sido implementado una convolución con tres máscaras, una para cada coordenada 3D. Más concretamente han sido usadas una matriz de Sobel y su traspuesta [42]. Todos los datos tienen asociados un valor de transparencia que se ajustaba en el mismo proceso de clasificación.

Ha sido utilizado un procesador AMD Athlon 1200 MHz, con 512 MB de memoria RAM, con una tarjeta gráfica ATI Radeon X600 y un sistema operativo Linux (Gentoo) con kernel versión 2.6. Las diferencias en los tiempos de procesamiento utilizando este tipo de iluminación y un tipo de iluminación clásico Lambertiano son prácticamente despreciables.

Podemos apreciar en la figura 3.11 como el método es capaz de realzar el modelo de forma que se aprecien mejor las partes más importantes del modelo original.

Incluso sin realizar un paso previo de clasificación por tono podemos obtener una imagen realizada, tal y como muestra la figura 3.12.

Podemos apreciar como los detalles pequeños se aprecian mejor utilizando nuestro método, como por ejemplo en la figura 3.13, los huesos del pie se distinguen muy fácilmente con nuestro método, sin embargo con la iluminación clásica queda oculta por la información de las sombras.

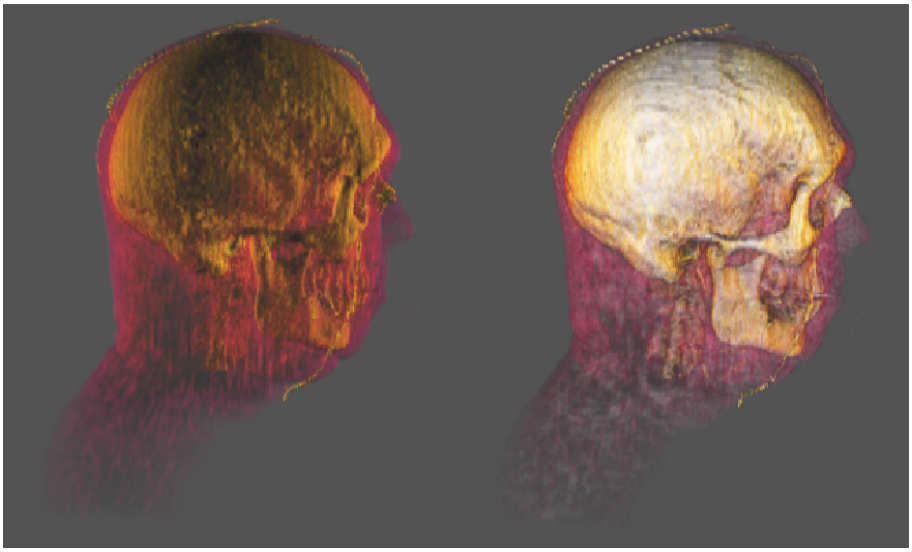

Figura 3.11: Cabeza visualizada con el método clásico y con nuestro método (a partir de una tomografía axial computerizada). Haciendo previamente la clasificación por tono.

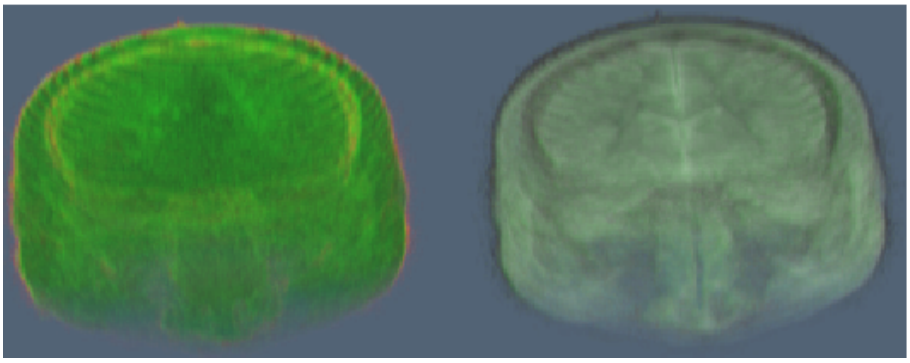

Figura 3.12: Cabeza visualizada con el método clásico (izquierda) y con nuestro método (derecha). No hay clasificación de tono (los datos se han obtenido a partir de una resonancia magnética).

Figura 3.13: Pierna obtenida mediante una iluminación clásica (izquierda) y nuestro método (derecha), con una clasificación previa por tono.

Capítulo 4

Detección de siluetas en volúmenes

En este capítulo se mostrará como detectar siluetas mediante un algoritmo modificado de trazado de rayos. Además se mostrará como una modificación del mismo algoritmo puede ser utilizada para cualquier otra técnica de visualización de volúmenes.

Introducción

Como hemos visto en sucesivos capítulos, una de las partes más importantes en el área de la visualización expresiva es la obtención de siluetas, esto es así debido principalmente a que ayudan al sistema perceptual humano a reconocer fácilmente los objetos que está visualizando sin necesitar un examen exhaustivo de la imagen.

Aunque hay muchos métodos basados en estructuras y en análisis de imágenes, estos métodos están limitados de una u otra forma por la resolución del modelo o por la geometría del objeto. En este capítulo se mostrarán las técnicas más usadas y los trabajos desarrollados basándonos en la detección de siluetas a través de una discretización del modelo por capas.

4.1. Trabajos previos en polígonos

No existen demasiadas referencias a detección o tratamiento de las siluetas en modelos poligonales. Un trabajo que merece la pena destacar por su similitud a lo que se hace en volúmenes es el trabajo de Martín [63].

En este trabajo se discretizan las siluetas en una serie de planos, cada uno de ellos llamados EPA (*layered plane element* o elemento plano apilado) (ver figura 4.1). Cada EPA tiene dos componentes: un área representado por un polígono cerrado y dos o más siluetas cerradas (que pueden ser divididas en dos o más siluetas abiertas en los últimos pasos del método). Por ejemplo, los EPA de la esfera, el círculo pueden ser dibujados como una circunferencia, la silueta puede dibujarse como un círculo relleno sin borde, el polígono o ambos.

Un objeto se puede dividir en dos o más EPAs. Un objeto convexo siempre produce una EPA con un polígono y una silueta, la cual es cerrada (ver figura 4.1, A). Un objeto cóncavo puede producir una o más EPAs (ver figura 4.1, B), cada una de las cuales tienen una silueta cerrada o varias abiertas.

Para obtener una EPA, se crea una lista de puntos de la lista de aristas marcadas como silueta. Cada punto tiene uno o más caminos de la cadena asociada. Un camino de la cadena mantiene la información de la arista previa y la siguiente arista, las cuales obtienen siluetas cerradas. Un punto con solo un camino se denomina un *punto simple*, mientras que con dos o mas caminos de la cadena se denomina *puntos múltiples*. Las aristas de una cadena son convexas en los ángulos entre la normal de ambas caras mayores a 180 grados, o cóncavo en otro caso.

El problema de esta técnica es que está limitada a modelos poligonales y no se puede aplicar sobre modelos volumétricos debido a que para ello habría que realizar un análisis

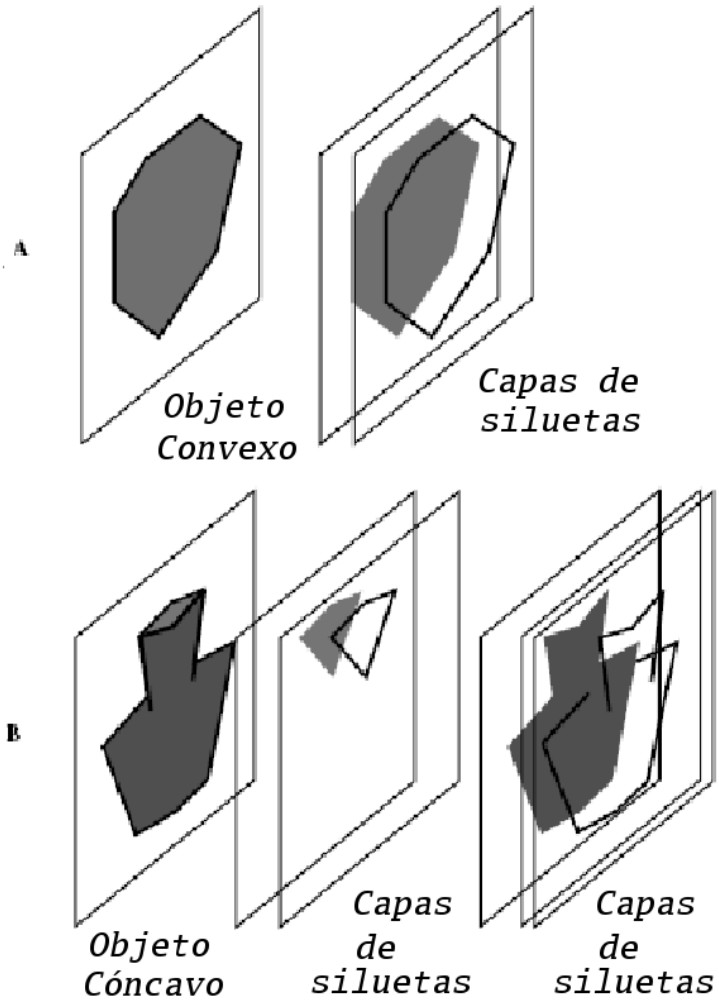

Figura 4.1: Ejemplos de EPAs para un objeto convexo (arriba) y cóncavo (abajo).

individual de cada plano que corta al volumen, además existen casos especiales en que el algoritmo no funciona y estos casos son demasiado frecuentes en volúmenes reales, en los que existen concavidades y convexidades además de huecos en el modelo.

4.2. Trabajos previos en volúmenes

En cuanto a visualización de volúmenes, el trabajo más relevante sobre siluetas es el de Kindlmann [48], que aún trabajando con isosuperficies obtiene unos buenos resultados. Sin embargo los resultados más relevantes utilizando un algoritmo por capas es obtenido por Nagy [72].

Kindlmann propone utilizar la curvatura de una superficie para el cálculo de siluetas. A

continuación comentamos de forma más detallada este trabajo.

4.2.1. Detección de siluetas a partir de la curvatura

Podemos definir la curvatura de una superficie formalmente de la forma siguiente:

Definición 3. *La curvatura de una superficie se define como la relación entre pequeños cambios de posición sobre la superficie y los cambios resultantes entre las normales.*

La medida de la curvatura se basa en el concepto de *tensor geométrico*, que es la curvatura del gradiente[48, 22].

En los volúmenes las superficies son representadas implícitamente como isosuperficies de valores de datos $f(x)$. Asumiendo que estos valores de f se incrementen, nos movemos dentro de los objetos de interés, la normal a la superficie es $n = \frac{-g}{|g|}$, con el valor del gradiente dado por g.

La información de curvatura está contenida en un gradiente ∇n^T, que es una matriz 3×3. Sin embargo, no se evalúa el gradiente de una vector normalizado precalculado, sino que se realiza una convolución de los datos originales.

El cálculo de los vectores nos dice que la matriz Hessiana H representa como cambia el gradiente g como una función de cambios infinitesimales en la posición en $\mathbb{R}^3$. Los cambios en g tienen una componente sobre g (el gradiente cambia de longitud), y el componente sin el plano tangente (también cambia de dirección) como se comenta en la sección 2.2.

El trabajo de Knidlmann se basa en los siguientes pasos para calcular la curvatura en un campo escalar arbitrario:

- Calcular la cantidad de la primera y segunda derivada parcial comprimiendo el gradiente g.
- Calcular el trazo T y la norma de Frobenius F [33] de g. De forma que se cumpla una relación entre ambas.

Si los valores de los datos interiores a las regiones de interés son menores que el fondo, el único cambio en la formulación es el signo del tensor geométrico. Las principales direcciones de curvatura se encuentran fácilmente como los autovectores del mismo. La principal diferencia de este método es que se usan convoluciones con filtros continuos y no discretos, como suele ser usual.

Como desventaja principal de esta técnica tenemos que el tiempo de obtención es elevado y no se permite obtener fronteras mediante visualización directa, de forma que no podemos observar varios componentes al mismo tiempo sin repetir todo el proceso dos veces. También depende a sobremanera del umbral manual que determina el usuario para la zona de interés ya que en función del mismo podemos obtener resultados francamente desagradables o agujeros en el objeto.

4.2.2. Detección de siluetas basada en capas

Mientras que Kindlmann detecta fronteras a base de filtros más o menos complejos, Nagy opta por una solución más sencilla, permitiendo un realce mediante los coeficientes de Fourier de los bordes obtenidos (ver sección 2.2).

El algoritmo propuesto por Nagy se basa en la visualización de volúmenes mediante una técnica de texturizado 3D (ver introducción para más detalles).

De forma general, el algoritmo dibuja cada uno de los cortes desde el frente hacia atrás usando una extracción por isosuperficies. Se define un píxel que será el *pixel de contorno*, si cualquier fragmento de la textura sobrevive al α-test durante la visualización se transforma en un candidato a *pixel de contorno*. El esquema se muestra en la figura 4.2.

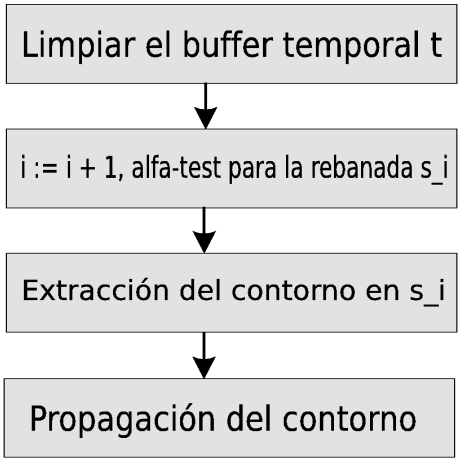

Figura 4.2: Esta figura describe en términos generales, el método de Nagy para obtener siluetas.

De forma más detallada, supongamos que tenemos un píxel de contorno visible c_k en una rebanada s_{i+1} detectada en una posición particular de la pantalla. Para decidir si c_k es un píxel de la silueta se comprueba si el píxel es visible en la capa s_{i+2} en la misma posición. Si no es el caso podemos asumir que tenemos un píxel de la silueta detectado. Si se encuentran múltiples píxeles de contorno en sucesivas capas en la misma posición (por ejemplo, por haber propagado los píxeles de contorno), la dirección del observador local es ortogonal a la normal de la isosuperficie, y por tanto se puede decidir de forma sencilla cual de ellos está más alejado del observador.

Si hay un fragmento que haya sobrevivido al α-test en la misma posición, entonces no tenemos una silueta, en otro caso sí (ver figura 4.3). s_i, s_{i+1} y s_{i+2} son capas paralelas después de sobrevivir al α-test. f, c_k y i denotan al *framebuffer*, el píxel de contorno y la isosuperficie, respectivamente. El código de color pasa los fragmentos en gris claro, los píxeles en gris y las siluetas en negro. Arriba a la izquierda: c_k está en s_{i+1} se detecta como un píxel silueta, desde que este es visible y el fragmento sucesor en s_{i+2} no pasa el α-test. Arriba a la derecha: una situación similar, donde c_k es un píxel de contorno propagado. Abajo a la izquierda: c_k no se reconoce como píxel silueta, desde que el fragmento en s_{i+2} pasó el α-test. Abajo a la derecha: importancia de la decisión de la definición tomada en la imagen abajo a la izquierda, si definiéramos un píxel contorno para ser un píxel silueta solamente debido a su propagación, tendríamos múltiples siluetas sobre la frontera de las superficies más curvas.

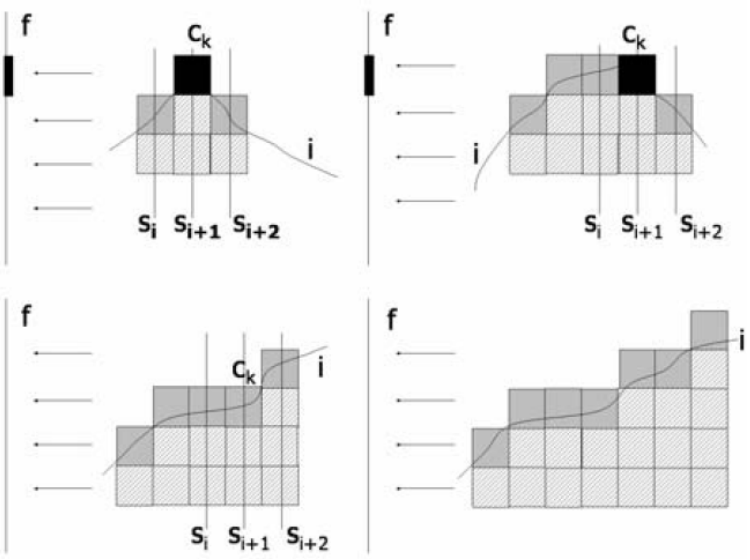

Figura 4.3: Ejemplos para la determinación de píxeles silueta.

4.2.3. Propagación del contorno

El algoritmo en sí mismo trabaja como un cauce clásico de extracción de isosuperficies con operaciones extendidas aplicadas sobre una única capa. Estas reglas requieren un acceso a los resultados temporales. Por ello, para cada fragmento de la textura se tiene una estructura con los siguientes campos (según la notación del autor):

- Dato del volumen (TU 1): Dimensión 3D.
- "Huella digital" (TU 2): Dimensión 2D.
- Contorno (TU 3): Dimensión Dimensión 2D.
- Resultado intermedio (TU 4): Dimensión 2D.

Inicialmente se limpian las texturas con el color de fondo. Después se asocia cada fragmento en un orden desde delante hacia atrás, desactivándose el test por profundidad y la escritura en el *buffer* de profundidad.

Los pasos del algoritmo se describen a continuación con detalle:

1. Se visualiza el volumen con el α-test activo (figura 4.4). Se almacena el contenido del framebuffer en TU 1 y se llama a su "huella digital" en TU 1. De esta forma obtenemos de forma natural dos clases de píxeles, definidos como vacío (□) y lleno (▨).
2. Visualizar un polígono con la textura con el color dado en TU 1. Un píxel lleno se modifica como píxel del contorno, si no todos los píxeles en la 8-vencidad son llenos. Se almacena el resultado en TU 2 y se nombra como contorno.
3. Visualizar la textura a partir del contorno obtenido en TU 2 y el resultado intermedio en TU 3. La tabla de decisión se muestra más abajo y nos dice como combinar dos

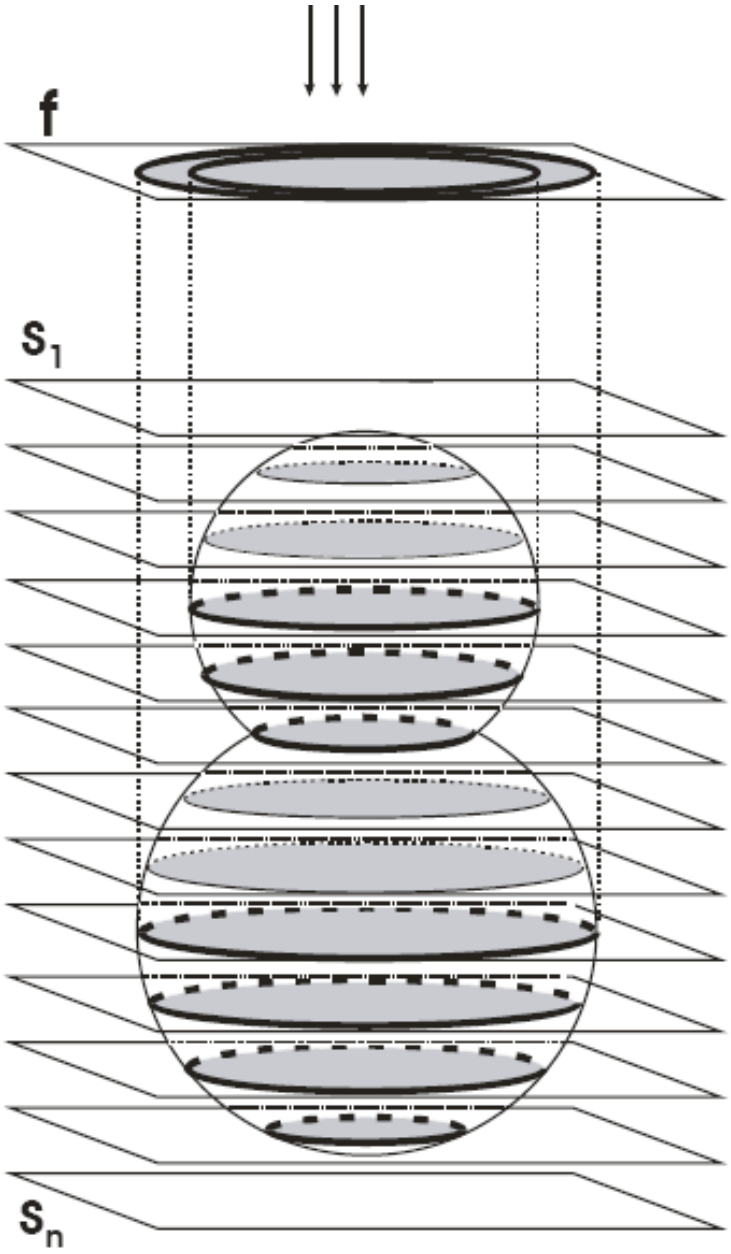

Figura 4.4: Esquema general de la propagación de fronteras con el método de Nagy.

píxeles de la misma textura usando el operador $\odot$. Los píxeles silueta se denotan como ■ y los píxel contorno como ■.

La idea que hay detrás de la tabla que propone Nagy es sencilla (ver figura 4.5):

- En la primera columna: los píxeles vacíos en s_i son transparentes, siempre son sobreescritos por los píxeles en s_{i+1}.

- En la segunda columna: los píxeles en s_i son opacos, nunca se sobreescriben por los píxeles de s_{i+1}.

- En la tercera columna: aquí es donde realmente se detectan las siluetas. En la primera fila, se detecta, en la segunda no hay píxel silueta, en la tercera, se propaga el píxel contorno.

- En la cuarta columna: los píxeles silueta se determinan en s_i son incondicionalmente propagados hacia el resto de las capas.

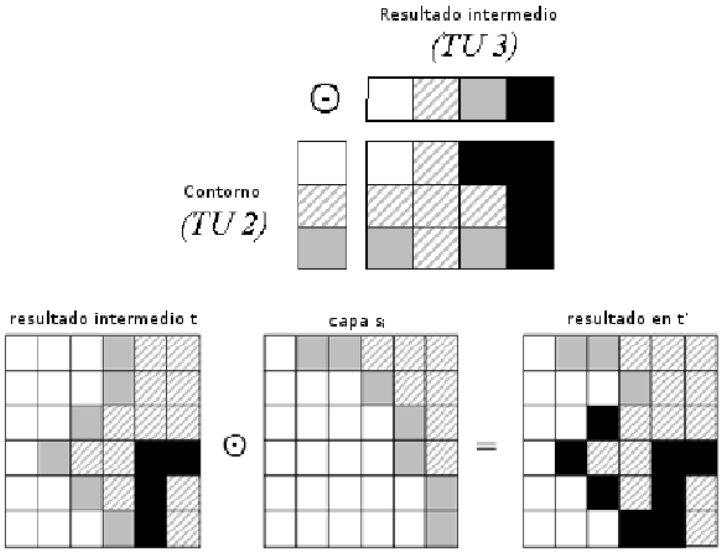

Figura 4.5: Arriba: tabla de decisión que define como se combinan los valores TU 2 y TU 3 en uno nuevo; Abajo: un ejemplo de combinación para obtener siluetas.

Si bien es cierto que los resultados obtenidos por Nagy y Kindlmann son bastante buenos, también es cierto que ambos necesitan definir un umbral manual sobre una superficie y que realmente adaptan el método poligonal de Martín sobre las isosuperficies obtenidas del volumen, por ello no trabaja directamente con los datos volumétricos sino con un preproceso de los mismos. Además el método de Nagy no permite tampoco la obtención de siluetas en el interior del modelo.

4.3. Detección de siluetas basada en operadores y capas

4.3.1. Introducción

La desventaja de los métodos anteriores es que no permiten detectar las siluetas en el interior de los objetos ya que trabajan solamente con una superficie, realmente no es una detección de fronteras a nivel de volumen. En este trabajo proponemos un nuevo algoritmo aplicado a la visualización de volúmenes para la detección de siluetas. Es aconsejable sin embargo, para obtener una buena clasificación de fronteras, tener un tipo de iluminación que no modifique el valor del tono de los datos en un paso previo a esta detección de siluetas. A continuación detallamos el trabajo de forma exhaustiva.

4.3.2. Aproximación al método

Este método altera el cauce normal en un trazado de rayos [71], incluyendo además un paso previo de clasificación. Es necesario que el modelo de iluminación empleado no cambie el tono ya que el método se basa en esta característica para diferenciar las zonas de interés.

Los pasos generales del algoritmo se muestran en la figura 4.6, indicando las zonas en gris la alteración de un cauce normal en el proceso de construcción y visualización del modelo, y las de color morado pasos que, si bien modifican el cauce normal del algoritmo, no son pasos imprescindibles para el buen funcionamiento del mismo ya que son intrínsecamente dependientes de la elección de la característica del tono como elemento separador entre zonas de interés.

4.3.3. Clasificación y cálculo de iluminación

Debido a que la mayoría de datos que se obtienen en modelos volumétricos vienen dados como una serie continua de imágenes en escala de grises [99, 10], el método de clasificación empleado se basa en tonalidades de gris. En los cuales, a partir de ciertos umbrales, aparecen zonas de interés en el modelo.

El sistema de clasificación que se propone es utilizar un rango de tonos para cada una de las zonas de interés, de tal forma que vayan resaltadas con respecto al resto del modelo. Este proceso puede ser fácilmente implementado para que aparezca de forma casi automática dicha clasificación puesto que a partir de un umbral se puede asignar un valor de tono asociado a un color en el espacio HSV [29, 87].

El resultado de esta clasificación es un tono asociado a cada dato. Los datos que pertenecen a las mismas zonas de interés variarán su tono levemente unas respecto a otras, mientras que las que pertenecen a distintas zonas de interés tendrán tonos muy diferentes entre sí. En cuanto al método de iluminación, es de suma importancia que el modelo elegido en el proceso de visualización no modifique el parámetro de clasificación por lo que ya hemos comentado.

Una forma de solucionar este problema fácilmente es hacer independiente el proceso de visualización con respecto al de detección de siluetas, llevando un sistema de color para el proceso de iluminación y otro distinto para el proceso de clasificación, de tal forma que en los sucesivos pasos el proceso de visualización aparezca diferenciado con respecto a la detección de siluetas tal y como se muestra en la figura 4.7.

El modelo de iluminación que se ha usado es el siguiente:

Sea I_j la fuente de luz con dirección $\vec{L}_j$ que ilumina el punto $\vec{P}_i$ con la normal asociada $\vec{N}_i$, la cámara C está orientada hacia la dirección dada por $\vec{D}$. De tal forma que los valores HSV

Figura 4.6: Cauce general de procesamiento del algoritmo.

del punto $\vec{P_i}$ vienen dados por las siguientes ecuaciones:

$$H = TonoAsociado(\vec{P_i})$$

$$S = \vec{L_j} \cdot \vec{N_i}$$

$$V = \vec{D} \cdot \vec{N_i}$$

La saturación (S) sigue un modelo inspirado en el modelo Lambertiano clásico y el efecto producido es parecido, es decir, el sombreado de los objetos. Mientras que el valor (V) al

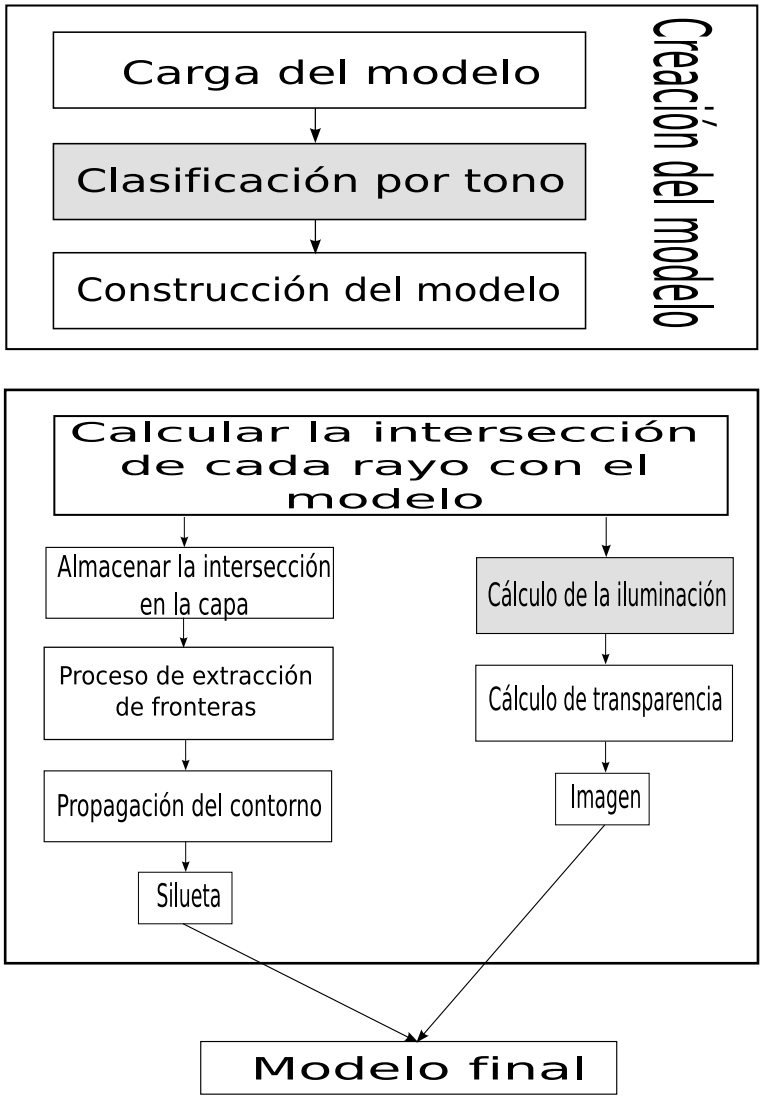

Figura 4.7: Cauce general de procesamiento del algoritmo modificado para hacer independiente el proceso de visualización.

depender de la dirección del observador, el efecto que produce es oscurecer las fronteras de los objetos desde ese punto de vista.

4.3.4. Ajuste de capas

El algoritmo de detección de siluetas va a operar sobre un *buffer* que recogerá al final del algoritmo las siluetas detectadas. Para ello se va a operar el citado *buffer* acumulador con otros *búfferes* paralelos a él y extraídos del volumen (rebanadas del mismo). En esta sección

mostramos cómo obtener esas rebanadas del volumen, que denominamos capas-imagen. En la siguiente mostramos como operar las capas-imagen para calcular las siluetas.

La construcción de las capas-imagen la realizamos durante el proceso de trazado de rayos, mientras se van lanzando los rayos desde el observador pasando por los píxeles vamos almacenando para cada píxel una lista de intersecciones [54].

Cada intersección viene dada por una característica c_{xyz} del dato intersectado por el rayo r_{xy} y una distancia desde el origen del rayo d_{xyz}. En nuestro caso la característica será el color con un valor de transparencia u opacidad (RGBA).

Según la notación tomada x e y indican respectivamente la columna y fila del píxel tomado, mientras que z indica la posición en la lista de intersecciones asociadas al punto de coordenadas (x, y) (ver figura 4.8).

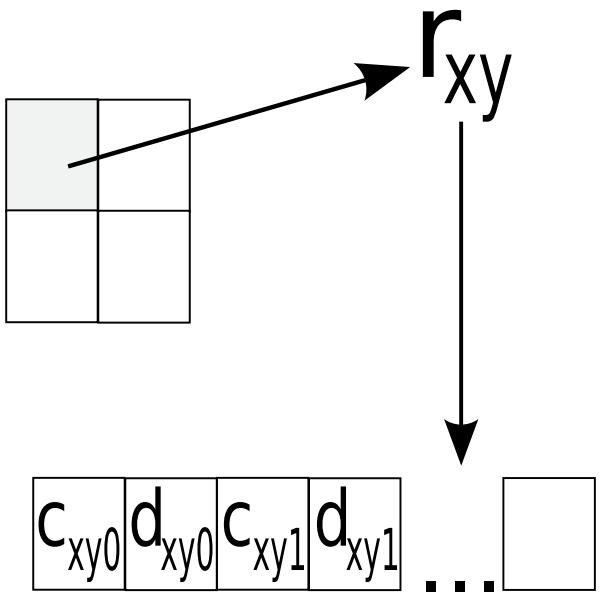

Figura 4.8: Para cada píxel (celda gris) hay un rayo asociado, que a su vez tiene asociado una serie de intersecciones dadas por características c_{xyz} y distancias d_{xyz} desde el origen del rayo

Cada una de estas listas de intersecciones y características está asociada, por tanto, a un píxel. Todas estas características conviene tenerlas en unos *búfferes* independientes de tal forma que las intersecciones queden discretizadas en capas para un proceso posterior de detección de fronteras. Cada capa-imagen, por tanto, tendrá los datos del modelo tal y como si fuera hecho rebanadas siguiendo el plano de la cámara.

La inserción de una característica en un dato viene determinado por el número de capas-imagen que tengamos, éste es un parámetro definido por el usuario pero que es fácilmente calculable si el proceso de inserción se realiza después de tener cada dato en la capa-imagen en lugar de realizarlo al mismo tiempo (ya que se puede calcular el número de capas-imagen en función del número de intersecciones máximo entre todos los rayos lanzados).

Sea r_z el número de capas-imagen elegido, sea z_{min} el mínimo de todas las distancias de

todas las listas, y z_{max} el máximo de las mismas distancias. Sea $inc = \frac{(z_{max} - z_{min})}{r_z}$, definimos el operador $\div$ como el operador de división entera truncando por defecto. Sea r_x la resolución de las capas-imagen en el eje x (todas tienen la misma resolución), r_y la resolución de las capas-imagen en el eje y, s el número de listas, y n_i el número de elementos de la lista l_i. Cada una de las intersecciones en la lista l_i viene representada por tij. Como hemos comentado anteriormente, x e y indican respectivamente la columna y fila del píxel tomado.

El algoritmo para introducir los elementos es el siguiente:

- Para todo x desde 0 hasta $r_x - 1$:
  - Para todo y desde 0 hasta $r_y - 1$:
    - $i = x + y \cdot r_x$
    - Para todo j desde 0 hasta n_i:
      - $Introduce(t_{ij}, c_{xy})$

donde la función *Introduce* es una función que asigna el valor c_{xy} a la celda (x,y,z), y donde z ha sido previamente calculada como $z = (t_{ij} \div inc) - (z_{min} \div inc)$. Obviamente i es la posición en el *array* de listas y j es la posición en la lista l_i.

Con este algoritmo cada intersección se ajusta a la capa-imagen más cercana. Sin embargo, varias intersecciones pueden coincidir en una misma celda de una misma capa-imagen, en ese caso se podrían tomar varias aproximaciones:

1. Tomar uno de ellos de forma indiferente.
2. Tomar el más cercano a la capa-imagen.
3. Tomar la media u otro tipo de interpolación entre los datos.

A priori podría parecer que la mejor aproximación sería la tercera opción, pero hay que tener en cuenta que una media o interpolación en los valores RGBA modifica el tono del color resultante, siendo inútil la clasificación previa. Por ello hay que adoptar un criterio que no modifique la característica que diferencia las zonas de interés, normalmente, para un número más o menos elevado de capas-imagen, puede adoptarse la opción 1 o la 2 indistintamente sin observar ningún cambio apreciable.

Por último, aquellas celdas en las que no se introdujo ningún dato de color se fijan al color y transparencia del fondo. Hay que tener en cuenta que el algoritmo separa las capas-imagen de manera uniforme, sin embargo podría modificarse fácilmente para adaptar la separación de las mismas si se tiene un conocimiento previo de la geometría del modelo.

Una vez obtenidas las capas-imagen tenemos el modelo cortado en rebanadas pero desde la dirección y posición de la cámara, de tal forma que el cálculo de estas rebanadas se realiza al mismo tiempo que se visualiza el modelo, y el resto del algoritmo no se ve afectado en rendimiento. Todas las capas-imagen se encuentran ordenadas en profundidad, desde la más cercana a la más lejana.

4.3.5. Extracción de fronteras

Para la obtención de fronteras se define una serie de *buffers* auxiliares (llamados capas-acumulador) basados en el trabajo de Nagy, cada una de estas capas-acumulador puede tener tres tipos de datos: *vacío*, *lleno* o *frontera*. Además el número de estas capas-acumulador

es el mismo que el de las capas-imagen, y su resolución también es la misma. Cada capa-acumulador está asociada a una capa-imagen correspondiente.

El valor vacío es marcado cuando el valor alfa en una celda de la capa-acumulador es cero, y es lleno en cualquier otro caso. De esta forma todo lo que es semi-opaco es marcado a lleno.

Con esta aproximación el resultado obtenido es similar al de la figura 4.9. Mientras que en el artículo de Nagy las fronteras eran calculadas con la condición siguiente: *Un valor lleno es modificado a frontera si no todos los ocho vecinos son llenos*. En el método que se propone aquí el cambio a frontera se produce con la siguiente condición: *Un valor lleno es modificado si alguno de los ocho vecinos es vacío o el tono del valor lleno difiere con el tono de alguno de sus vecinos llenos en un error* δ. Este error δ suele coincidir con la diferencia entre los tonos realizados en la primera clasificación para las distintas zonas de interés, o bien, puede ser definido manualmente. De esta forma todos los valores llenos que se encuentren limitando un tono o vacío son marcados como frontera. Hay que tener en cuenta que este proceso de extracción se efectúa para cada capa-acumulador de forma independiente.

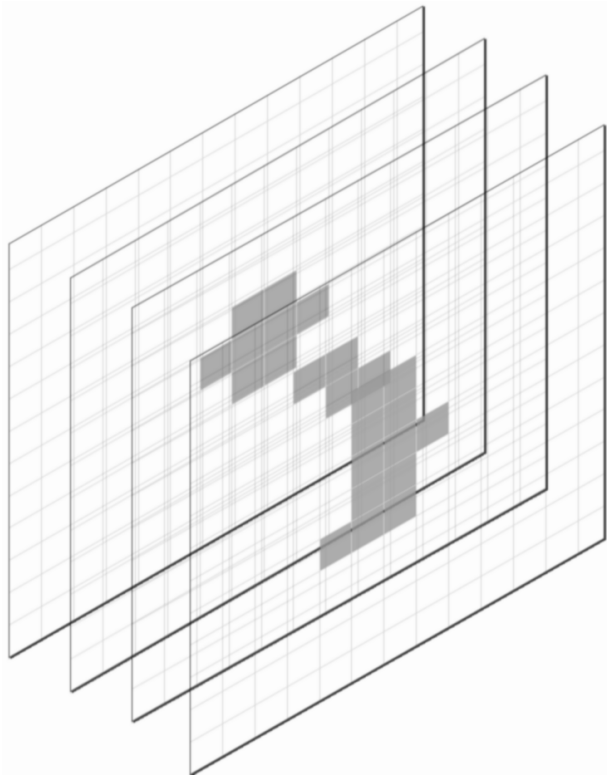

Figura 4.9: Después de aplicar el test del canal de transparencia, el resultado queda marcado en lleno (gris) para todos los datos que eran semitransparentes, y vacíos (blancos) para todo lo que tenía un nivel de transparencia máximo

4.3.6. Extracción de siluetas

Para obtener la silueta Nagy define un acumulador y un operador, en este trabajo presentamos dos acumuladores y dos tipos de operadores ya que se desea obtener silueta, no sólo del objeto que tenga frontera con una zona vacía, sino también entre elementos que tengan un tono distinto. De esta forma uno de los acumuladores (lo llamaremos A_0) tendrá almacenado cuatro tipos de valores: *vacío*, *lleno*, *frontera* o *silueta*. Mientras que el otro acumulador (lo llamaremos A_1) tendrá almacenado un valor entre 0 y 360 correspondiente al tono de un color en el espacio HSV.

Partimos con los acumuladores con un valor por defecto y vamos actualizándolos al ir operándolos con las capas (tanto las capas-imagen como las capas-acumulador) desde la más cercana a la más lejana. Los operadores propuestos en este capítulo ($\oplus$ y $\otimes$) se muestran en la figura 4.10 y 4.11 respectivamente. Y la justificación del resultado de las operaciones es la siguiente:

En el operador $\oplus$ tenemos los casos triviales de operación indicados con color negro, en los que si en una celda del acumulador A_0 tiene el valor vacío, se le asigna lo que haya en la capa-acumulador que se está procesando; cuando está lleno se mantiene el valor que había (salvo en el Caso A que explicaremos más adelante). Para el caso en el que lo que hay en la celda del acumulador A_0 sea frontera, si en la capa-acumulador que se está procesado hay un vacío significa que siguiendo el rayo visual del observador se ha pasado de frontera a vacío, lo cual significa que ese punto debe ser silueta. Cuando una celda del acumulador A_0 se ha definido como silueta simplemente se mantiene.

Los casos especiales A y B están situados en la tabla cuando se pasa de frontera a lleno o de lleno a frontera, en cuyo caso hay que examinar la característica asociada (en nuestro caso el tono) al acumulador y a la capa-imagen en ese lugar, de modo que se pueda discernir si está habiendo un cambio de tono, en cuyo caso estamos ante un cambio de objeto y debe considerarse una frontera o una silueta; o por el contrario si estamos dentro de la misma tonalidad, estamos ante el mismo objeto y por tanto el operador dará como resultado la etiqueta *lleno*. Así los casos se definen de la siguiente forma:

- Caso A: Si $Tono(A_0) = Tono(capa - imagen)$ entonces marcar el acumulador como *lleno*, en otro caso marcarlo como *frontera*.
- Caso B: Si $Tono(A_0) = Tono(capa - imagen)$ entonces marcar el acumulador como *lleno*, en otro caso marcarlo como *silueta*.

Debido a estos dos casos especiales en los que hay que hacer una comparación entre tonos, el acumulador A_1 y el operador $\otimes$ se hacen necesarios ya que se necesita almacenar los tonos de los últimos valores que modificaron el acumulador A_0. El operador $\otimes$ se define según se muestra en la figura 4.11 con dicho objetivo de que cada celda almacene el tono que provocó un cambio en su correspondiente celda del acumulador A_0. De igual modo, también tenemos dos casos especiales A y B que definimos a continuación:

- Caso A: Si $Tono(A_0) = Tono(capa - imagen)$ entonces no tocar A_1, en otro caso igualarlo al tono de la capa-imagen.
- Caso B: Si $Tono(A_0) = Tono(capa - imagen)$ entonces igualar A_1 al tono de la capa-imagen, en otro caso no tocar A_1.

El operador booleano de igualdad que se usa en los casos A y B de los operadores $\oplus$ y $\otimes$ está sujeta al mismo error δ que se mencionaba en la extracción de fronteras.

⊕	Vacío	Lleno	Frontera	Silueta
Vacío	Vacío	Lleno	Silueta	Silueta
Lleno	Lleno	Lleno	Caso B	Silueta
Frontera	Frontera	Caso A	Frontera	Silueta

Figura 4.10: Operador $\oplus$, el resultado indica el estado del acumulador A_0 tras operarlo con una capa-acumulador. Los posibles valores del operando capa-acumulador están representados en la columna de la izquierda, mientras que los del operando acumulador A_0 (antes de operarlo) están mostrados en la parte superior

Ajuste del parámetro δ

El problema que se origina con el parámetro δ es que para el sistema de visión humano hay una clara distinción entre un tono azul y uno rojo, sin embargo el concepto de asignar un número o un intervalo es bastante más abstracto y difícil de discernir. De esta forma podemos diferenciar en el círculo cromático de la figura 4.12 ciertos intervalos de color que el ojo humano diferencia mejor que otros, definiendo una serie de ”colores” para cada zona. Si $\delta = 1$ cada cambio de tono será marcado como límite entre dos zonas de interés y provocaría siluetas en el interior de los objetos debido a esos matices de tonalidad. Es por ello que cada color se introduce en un grupo y se define δ de acuerdo con estos grupos. Además el sistema de visión humano aprecia mejor la diferencia entre los tonos azules y rojos que los naranjas y rojos, por ello δ debe quedar afectada por la distancia de estos grupos en el círculo cromático.

Sea $\delta_2 = d \cdot \delta$, donde d es la menor distancia entre los grupos de los dos tonos que se están comparando y δ_2 el nuevo error tomado para la comparación de tonos. Por tanto, δ_2 sustituye a δ en todas las partes del algoritmo mostrado hasta este punto. Este proceso es prácticamente automático ya que sólo hay que definir una paleta de tonos en el círculo cromático.

4.4. Detección de fronteras mediante dos pasadas

Se puede apreciar en el método anterior que al ejecutar el algoritmo se produce un solapado de las fronteras en las partes internas del modelo debido a las numerosas concavidades y convexidades de los modelos médicos reales. El motivo de este ruido viene dado por la forma en que se generan las capas del modelo (cuando un rayo interseca sobre el modelo

⊗	Vacío	Lleno	Frontera	Silueta
Vacío	Color acum.	Color acum.	Color acum.	Color acum.
Lleno	Color capa	Color capa	Caso B	Color acum.
Frontera	Color capa	Caso A	Color capa	Color acum.

Figura 4.11: Operador $\otimes$, el resultado indica el tono a almacenar en el acumulador A_1 tras operar el acumulador A_0 con una capa-acumulador. Los posibles valores del operando capa-acumulador están representados en la columna de la izquierda, mientras que los del operando acumulador A_0 (antes de operarlo) están mostrados en la parte superior

volumétrico la intersección se discretiza en una de las capas cercanas) y en que los operadores tan sólo almacenan la última operación posible, lo que no posibilita que el método pueda implementarse con texturas volumétricas.

Además tenemos el problema de que el método anterior únicamente se puede aplicar a una visualización de trazado de rayos, pero no a una técnica de visualización mediante texturas o *splatting*.

La solución a este problema se encuentra en redefinir el algoritmo en base a las siguientes suposiciones:

- Cada capa debe contener los datos exactos del volumen en la sección que cortan, es decir el volumen debe estar construido a base de rebanadas y cada rebanada constituirá una capa-imagen (esto se puede cumplir en trazado de rayos y en cualquier otra técnica).
- Existen distintos grupos definidos en base a un criterio inicial (puede realizarse después de segmentarla automáticamente o a mano).
- Es silueta todo aquello que hace de límite entre un grupo y otro y no está oculto por elementos del mismo grupo ni hacia el observador, ni en sentido contrario (aunque se puede rebajar la condición para que en sentido contrario, si son permitidos los elementos del mismo grupo en la misma línea visual).

Se definen dos nuevos *buffers*, en el primero de ellos cada elemento del mismo es una lista de fronteras (lo llamaremos B_f), mientras que en el segundo cada elemento es una lista de grupos (lo llamaremos B_g), además seguimos teniendo en el modelo una serie de capas

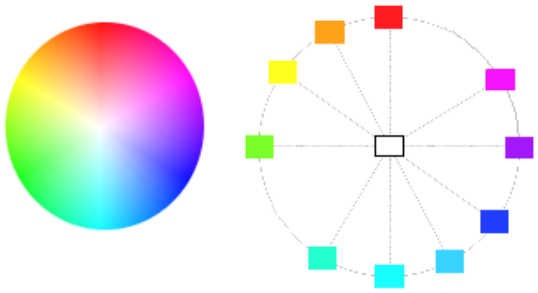

Figura 4.12: Círculo cromático y clasificación de intervalos de tonos más o menos distinguibles por el ser humano

denominadas capas-imagen. Cada material se puede definir como perteneciente a un grupo, incluido el vacío, el cual representaría otro grupo. Hay que tener en cuenta que no hacemos uso de ningún operador mencionado en las secciones anteriores, solo los definidos en esta sección.

Además, la resolución de ambos es la misma que la de las capas-imagen, las listas que poseen ambos *buffers* como elementos tienen el mismo tamaño en ambos y sus elementos están relacionados por sus posiciones en la misma. De forma que si B_f tiene un tamaño de $w \times h$ píxeles, B_g también tiene dicha resolución, además si la lista que se encuentra en la posición (x,y) de B_f tiene n elementos, B_g también tendrá una lista de n elementos en la posición (x,y), los elementos de ambas listas también estarán relacionados, de forma que si el primer elemento de una de ellas es borrado, su análogo también será borrado (ver figura 4.13).

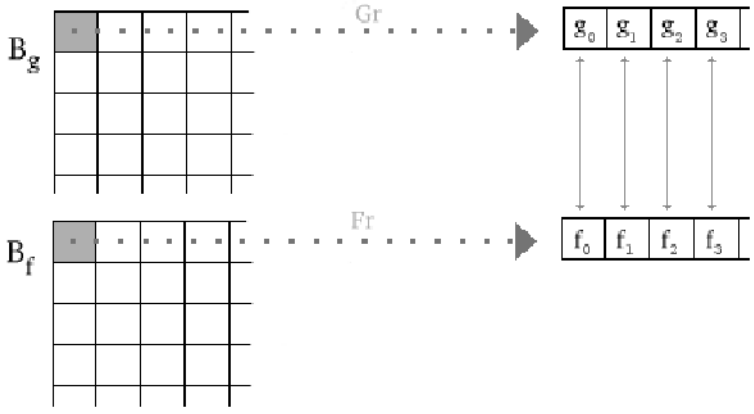

Figura 4.13: Como se puede observar el buffer de listas de grupos (B_g) y el buffer de listas de fronteras (B_f) contienen listas como elementos, cada una de estas listas tienen a su vez elementos relacionados, de forma que si en la lista Gr eliminamos g_1, también será eliminado f_1 en la lista Fr y viceversa.

Una vez hecho un paso previo de clasificación, se realiza el siguiente algoritmo:

```
01. Sea Bg el buffer de listas de grupos
02. Sea Bf el buffer de listas de fronteras
03. Para cada capa-imagen:
04.    Para cada elemento t de la capa-imagen con grupo g:
05.       Sea p la posición de t en la capa-imagen
06.       Sea Fr la lista correspondiente a Bf en la posición p
07.       Sea Gr la lista correspondiente a Bg en la posición p
08.       Si t está rodeado por elementos de otro grupo:
09.          Sea f el valor de profundidad de la capa-imagen de t en p
10.          f es introducido en Fr
11.          g es incluido en Gr
12.       En otro caso:
13.          Si G(t) es igual que alguno de los valores de la lista de grupos:
14.          Sea f el valor de profundidad relacionado con g en Fr
15.          g es eliminado de Gr
16.          f es eliminado de Fr
17. Es silueta toda lista no vacía de Bf
```

A partir de la línea 4 tenemos que cada *buffer* con las fronteras y los grupos funcionan de forma conjunta, y para cada capa imagen se realiza una operación bidimensional. Básicamente en la línea 8 se detecta si el elemento está rodeado por otros elementos de distinto grupo, en caso contrario (línea 12) significa que no tenemos una frontera.

En el caso de que hayamos detectado un candidato a frontera (línea 9) se introducen los valores de la frontera en las listas correspondientes de ambos *búfferes* . En el caso en el que claramente no haya un candidato lo sacamos de la lista (línea 15) de forma que al final los *búfferes* contendrán únicamente las siluetas finales del modelo.

Tal y como se presenta el algoritmo presenta dos problemas: el problema del "cono" y el del "muñeco de nieve". En el primer caso, todo lo que tenga forma de cono visto desde el pico sería marcado como un círculo relleno en lugar de representar la circunferencia. Mientras que en el segundo caso, un muñeco de nieve visto desde la cabeza presentaría siluetas tanto en la cabeza como en el cuerpo, mientras que al dar la vuelta y verlo desde abajo tan sólo presentaría la silueta del cuerpo, incluso si todo perteneciera a un mismo grupo.

Para evitar el primer problema se puede marcar la frontera teniendo en cuenta la siguiente condición en lugar de la que hemos mostrado anteriormente para marcar frontera:

```
01.     .
02.     .
03.     .
04. Si t está rodeado por elementos de otro grupo y
05.    El elemento de la capa anterior en su misma posición
06.    No es del mismo grupo:
07.    Sea f el valor de profundidad de la capa-imagen de t en p
08.    f es introducido en Fr
09.    g es incluido en Gr
10.     .
11.     .
12.     .
```

Para evitar el "problema del muñeco de nieve" podemos realizar el mismo algoritmo inicial (no es necesaria la condición definida para evitar el "problema cono", ya que también evita este problema) dos veces, una desde el observador al modelo y otra en sentido contrario. El resultado sería aquellos elementos que estuvieran en ambos *buffer* de listas de fronteras.

La ventaja principal de este algoritmo es que se puede conocer no sólo la posición de la silueta en el *buffer* sino también a que elemento pertenece, de tal forma que con el grupo se tiene coherencia entre el conjunto de puntos que conforman la silueta.

Si la profundidad en las capas-imagen no es relevante se puede eliminar el *buffer* de fronteras quedándonos con los grupos únicamente (igual que antes se marcaría silueta en todas las listas no vacías, esta vez en el *buffer* de listas de grupos). De igual forma podríamos considerar que el "problema del muñeco de nieve" no es un problema, lo único que habría que hacer es pasar el algoritmo una sola vez con la condición antes mencionada para evitar el "problema del cono".

El resultado mostrado en la figura 4.14 muestra un torso en el que se han aplicado tanto la técnica de las siluetas que se ha descrito aquí (utilizando una doble pasada para eliminar el "problema del muñeco de nieve"y evitar que obtengamos ruido en las siluetas), con una iluminación como la descrita en el capítulo 3.

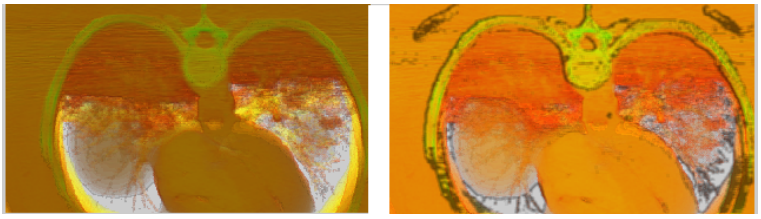

Figura 4.14: A la izquierda modelo difuso de iluminación clásico aplicado con un torso, mientras que a la derecha tenemos la detección de siluetas mediante dos pasadas y la iluminación usada en este capítulo.

4.4.1. Resultados

El método propuesto ha sido probado sobre un trazado de rayos optimizado con un árbol octal (*octree*), para el cálculo de normales ha sido implementado una convolución con tres máscaras de Sobel, una para cada coordenada 3D [42]. Todos los datos tienen asociados un valor de transparencia que se ajusta manualmente en el mismo proceso de clasificación. Se ha utilizado un procesador AMD Athlon 3500 MHz, con 1 GB de memoria RAM, con una tarjeta gráfica GForce5700 y un sistema operativo Linux (Gentoo) con kernel versión 2.6.

Todas las imágenes de esta sección tienen tres zonas de interés, los huesos marcados en verde, los órganos en rojo y los demás tejidos en naranjas, la distancia en el círculo cromático entre los verdes y naranjas es de entre 30 y 40 tonos, y la de los rojos y naranjas de 20 tonos. La resolución del modelo es de 128x200x300.

El número de capas (acumulador o imagen) utilizadas (r_z) puede parecer una limitación a priori, sin embargo, al depender de la resolución del modelo puede ser fácilmente calculable, siendo r_z mayor o igual a la resolución en profundidad del modelo.

El parámetro δ tampoco es una limitación ya que puede calcularse de forma automática a partir de la clasificación inicial de tonos, aunque puede ajustarse a mano para tener un mayor control sobre las siluetas detectadas.

En todas las imágenes se ha supuesto que el problema del muñeco de nieve no es un problema, de forma que en el interior de las siluetas aparecerán también las líneas de forma.

En la figura 4.15 se ha calculado δ automáticamente resultando el valor 35 como media de los valores 30, 25 y 40 que, como hemos comentado, es la distancia en el círculo cromático entre los tonos verdes, rojos y naranjas que usamos. Se observa como se obtiene un resultado

aceptable. r_z también a sido calculado automáticamente según el número de datos en profundidad. Podemos observar como a diferencia del método de Ebert, la detección de fronteras en el modelo no es un simple filtro sobre el modelo final, ya que la imagen f) no detecta las siluetas de cada elemento de interés, además nuestro método detecta fronteras en el interior del modelo (como pueden ser las costillas que apenas son visibles en el modelo original).

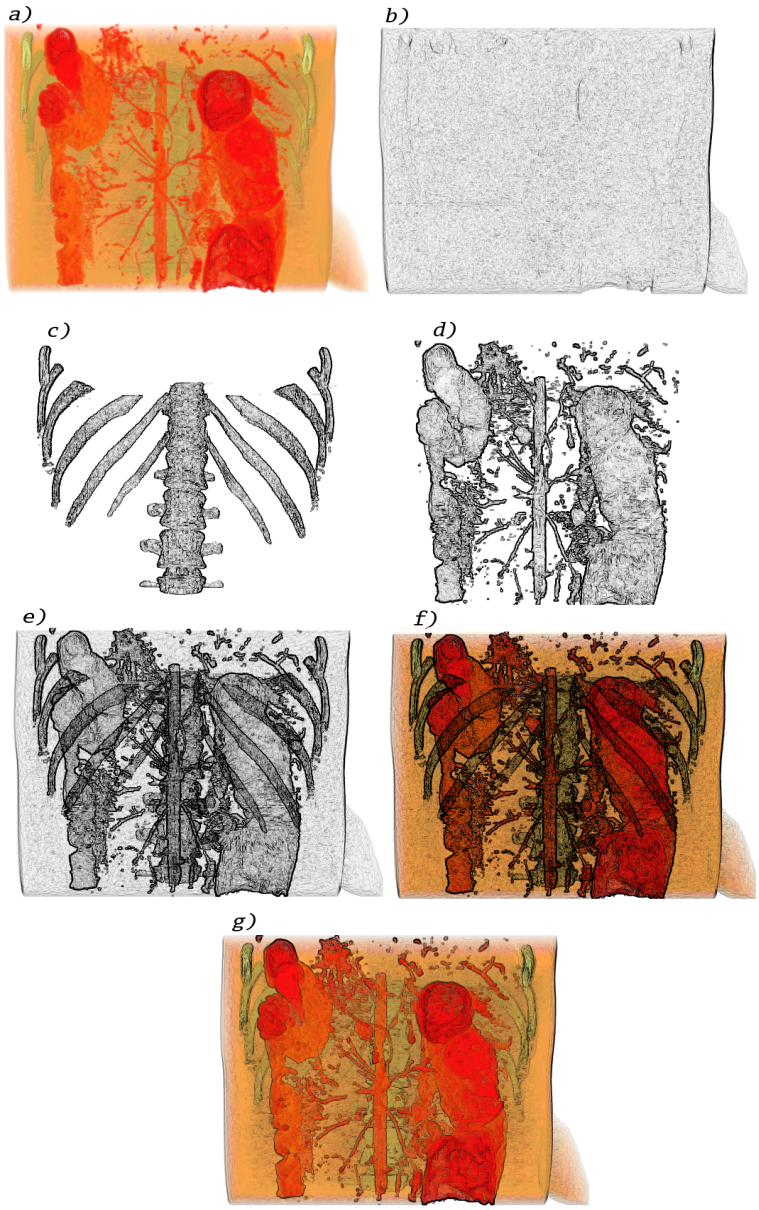

Figura 4.15: a) Imagen iluminada mediante un modelo Lambert + Phong; b), c) y d) Siluetas de los distintos grupos obtenidas a partir de a); e) Siluetas superpuestas; f) Siluetas superpuestas a la imagen a); g) Siluetas obtenidas mediante un filtro de Sobel sobre a).

Capítulo 5

Ilustración a partir de volúmenes

En este capítulo se realizará una revisión rápida de los trabajos más importantes en cuanto a la creación de ilustraciones a partir de la visualización directa de volúmenes y algunos conceptos previos sobre luces virtuales. Después veremos cómo una modificación en la ecuación de la fuente de luz convierte una fuente de luz en virtual y como se puede aplicar para simular distintos tipos de ilustraciones a partir de datos volumétricos.

Introducción

En los anteriores capítulos hemos visto como es necesario definir un método de mejora a las siluetas para poder visualizar mejor distintos detalles. Sin embargo las siluetas no son el único método a aplicar. Veíamos en el capítulo 2.2 como era muy útil una simplificación de los tonos o de los detalles del modelo, más aún en volúmenes donde la cantidad de datos es abrumante.

En este capítulo se mostrarán las técnicas más usadas y los trabajos desarrollados basándonos en la detección de siluetas y en la simplificación de datos mediante luces virtuales.

5.1. Luces virtuales en modelos poligonales

Con la primera aproximación se define una fuente de luz virtual que "ilumina" al modelo, normalmente detectando tanto líneas de forma como siluetas [64].

Martín define la luz virtual como un punto con una posición:

Definición 4. *Una **luz virtual** es un punto con coordenadas homogéneas* (x,y,z,w)*. Si* $w = 0$ *se dice que la luz está en el infinito y es denominada luz virtual no-local. Si* $w \neq 0$ *la luz se encuentra cercana a la escena, y es llamada luz virtual local.*

Esta definición es válida también en el ámbito de los volúmenes (aunque como veremos, puede ser extendida).

La idea básica de las luces virtuales es que una luz puede producir cambios, no solamente en el color o las sombras, si no también a las propiedades de obtención de siluetas y líneas de forma. De este modo, una luz virtual tiene asociada una serie de elementos externos (o parámetros) similares a los que una luz clásica tiene asociados.

En el caso de las luces reales, las luces virtuales se usan para calcular la intensidad en cada silueta. En el trabajo de Martín, lo que se usa es un modelo de iluminación clásico sencillo: $I = I_{ambiente} + I_{difuso} + I_{especular}$.

En el modelo de luces virtuales, las luces pueden ser clasificadas en función de la clase de reflexión que se usa para calcular las siluetas. La luz virtual ambiental no se incluye ya que no es un modelo de reflexión del que puedan obtenerse siluetas. También pueden clasificarse según el efecto que produce, relacionado a esos componentes de un modelo de iluminación sencillo.

Puesto que Martín define las luces virtuales como puntos en el espacio con coordenadas homogéneas (x,y,z,w), la luz posee únicamente atributos geométricos y la característica

de ser difusa o especular. Martín utiliza una condición de curvatura sobre cada uno de los polígonos para detectar las siluetas a partir del vector dado por la posición de la luz y la de la cara.

La gran ventaja de las fuentes de luz virtuales sobre otros métodos de detección de siluetas es que se separa la posición de la cámara de la detección de siluetas y por tanto podemos detectar siluetas desde cualquier posición y se pueden introducir tantas luces como se quieran, teniendo un conjunto de "sombreados" distintos para un mismo objeto de forma intuitiva para el usuario (ver figura 5.1). Además se permite mantener las siluetas independientemente de la posición de la cámara, tal y como muestra la figura 5.2.

Figura 5.1: Las sombras del objeto más a la izquierda están obtenidas con una luz virtual en distintas posiciones diferentes a la de la cámara.

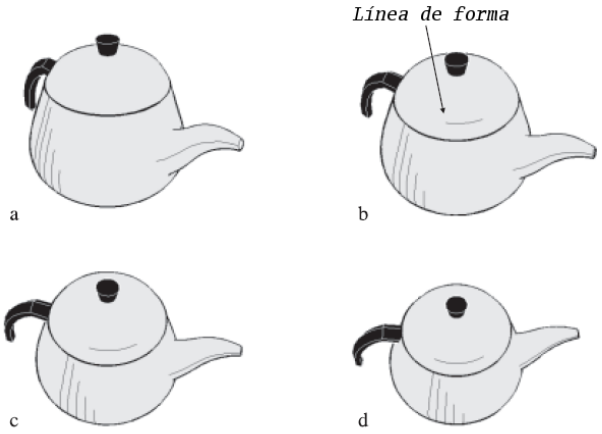

Figura 5.2: Aplicación de una luz virtual especular a una tetera. En b) aparecen las líneas de forma. En c) y d) las líneas cambian según la posición del observador.

5.2. Ilustración en volúmenes

En volúmenes tenemos pocas técnicas que implementen ilustración, los trabajos más relevantes son los de Stompel, Lum y Lu [91, 59, 57]. En los trabajos de Stompel y Lum se realiza una doble pasada con texturas de colores modificados según el gradiente. Realizando

un sombreado parecido al que realizan Gooch o Ebert en sus artículos a las isosuperficies extraidas del modelo, obteniendo una ilustración más parecida a una ilustración técnica que a una empleada en medicina.

El artículo de Lu es más interesante ya que propone un método para realizar punteado sobre el modelo volumétrico en visualización directa. Los pasos seguidos por Lu son los siguientes:

- Preprocesado: aquí se genera un número de puntos para cada volumen basándose en algunas propiedades extraidas de la imagen.
- Procesado del gradiente: en esta etapa se calculan la dirección del gradiente y la magnitud del mismo.
- Ajuste de la resolución: se realiza un aumento de la resolución de 5x5 píxeles.
- Generación de puntos aleatorios: en este paso se realiza un muestreo de puntos sobre el volumen.
- Fronteras y siluetas: se realzan las fronteras a posteriori lanzando puntos masivamente en el lugar donde deberían ir las fronteras. Para determinar las fronteras se hace uso de los gradientes.

Puesto que los primeros pasos ya han sido comentados ampliamente en los capítulos anteriores, nos centraremos en la parte de generación de puntos aleatorios y en el criterio de dispersión sobre el volumen.

Tenemos que tener en cuenta que los resultados obtenidos por Lu distan mucho de los resultados obtenidos en punteado a partir de modelos poligonales (ver figura 5.3). Esto es debido a la complejidad de los modelos volumétricos y a la dificultad añadida de tener elementos internos superpuestos a las partes exteriores del modelo[58].

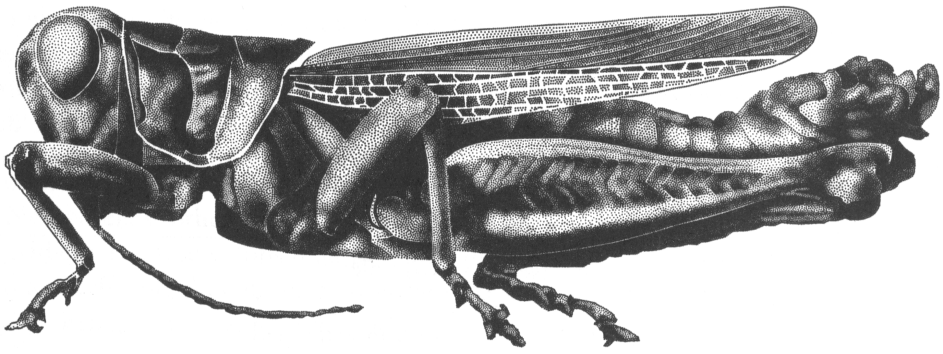

Figura 5.3: Ejemplo de punteado basado en un modelo poligonal.

5.2.1. Punteado sobre el volumen

La generación de puntos inicial se realiza mediante una distribución de Poisson, que se inicializa al número inicial de puntos. Los puntos se generan cerca de plano gradiente según

los datos estadísticos de la distribución de la magnitud del gradiente. Se colocan los puntos de forma aleatoria, al rededor del *voxel*, entre dos planos, $p0$ y $p1$, ambos separados por una distancia elegida por el usuario.

Después se ajustan las localizaciones de este subvolumen aproximando los puntos según la distribución mencionada. Después este proceso se mejora ajustando el número de puntos a dibujar en cada *voxel* con su respectivo tamaño.

Cada *voxel* almacena las siguientes características:

- Número de puntos
- Gradiente
- Valor escalar del *voxel*
- Tamaño del punto
- Lista de puntos conteniendo la localización x, y, z del mismo

Para obtener el número de puntos a lanzar por *voxel* se aplica la siguiente expresión:

$$N_t = N_{max} \times T \tag{5.1}$$

donde N_{max} es el número máximo de puntos por *voxel*, la resolución de pantalla viene dada por T, que se obtiene a partir de la siguiente expresión:

$$T = T_f \times T_s \times T_r \times T_d \times T_t \times T_l \tag{5.2}$$

donde T_f, T_s, T_r, T_d, T_t y T_l son respectivamente las fronteras, siluetas, resolución, distancia, transparencia interior y factores de iluminación. Cada factor se encuentra normalizado entre 0 y 1.

El tamaño base de un punto se calcula mediante la expresión:

$$S_t = ||\nabla \vec{V}_t|| \times S_{max} \tag{5.3}$$

El resultado final de esta técnica puede apreciarse en la figura 5.4.

5.3. Ilustración de volúmenes mediante luces virtuales

5.3.1. Introducción

La mayoría de las técnicas de visualización directa de volúmenes están orientadas a producir imágenes realistas, es por ello que usan los modelos de iluminación basados en Lambert, así como cambios en funciones de transferencia principalmente. Sin embargo el grado de realismo de las imágenes obtenidas dificulta su percepción para ciertos colectivos, es por ello que se suele recurrir al nivel de abstracción que proporciona una ilustración y que facilita la percepción de las mismas. En ilustración es común reducir la gama tonal, a menudo llegando a tener imágenes en escalas de grises o simplemente de punteado. También es típico el variar el grosor de las siluetas simulando diferentes sombras en el modelo. Esto es especialmente cierto en ilustraciones de tipo más realista, como el mostrado en la figura 5.5.

El método presentado en este capítulo tiene dos pasos: la creación y la visualización del modelo.

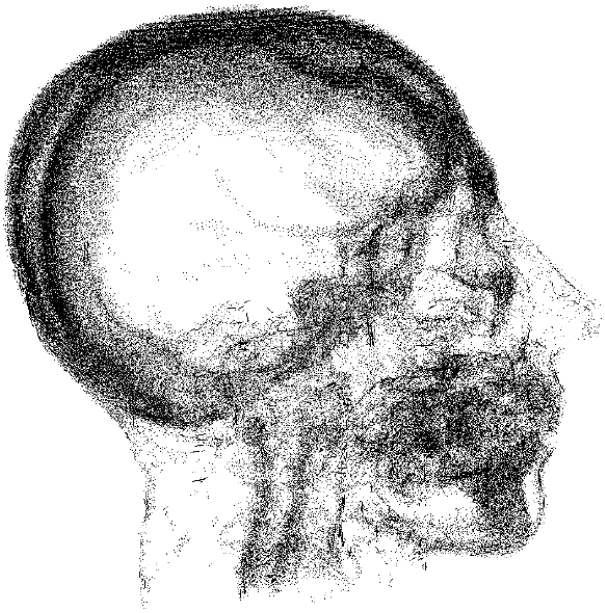

Figura 5.4: Volumen con realce en siluetas, después de aplicar el método de Lu con los parámetros ajustados manualmente.

En el paso de creación se produce una carga de los datos del modelo, y se realiza una clasificación por tono, la cual será distinta si se parte de un modelo en escalas de grises (o un modelo escalar en general), o si se parte de un modelo de color (que normalmente suele ser un modelo volumétrico RGB). Si bien pueda parecer a priori que esta clasificación es un paso de segmentación, en realidad no es un paso que sustituya a una segmentación sino que únicamente se realiza una asociación entre cada uno de los valores de entrada con cada uno de los valores de la paleta de materiales definida, independientemente de que se haya realizado previamente una segmentación o no. De tal forma que cada uno de los valores de entrada del volumen quedará asociado a un tono determinado. Además, en este primer paso se calcularán las normales a partir del gradiente de los datos de entrada.

El propósito del primer paso es que el usuario pueda elegir fácilmente los materiales asociados a cada uno de los datos del modelo original, sin que tenga que reiterar en un método de prueba y error continuo. Para ello, se tendrán en cuenta algunas de las características humanas de la percepción de colores, siempre basados en un sistema de color cercano al usuario, como puede ser el modelo HSV.

En el segundo paso, se aplica un modelo de color no realista que permite el ajuste del brillo y saturación de los colores, además de una herramienta de control de líneas de forma, sombras y siluetas mediante el modelo de luces virtuales.

El cauce general del método es mostrado en la figura 5.6. El paso de visualización del modelo es aquel que ha sido modificado con respecto a los métodos anteriores de visualización directa de volúmenes, incluyendo un sistema interactivo de luces virtuales.

Se realiza una descripción más detallada del algoritmo en las siguientes secciones.

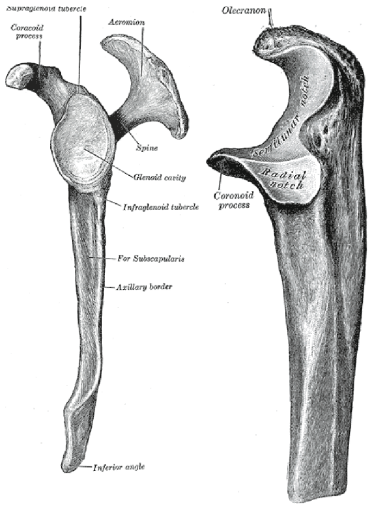

Figura 5.5: Ilustración médica realizada a mano, se puede observar el realismo de las mismas (Imágenes cortesía de la Wikipedia: `http://www.wikipedia.org/`).

5.3.2. Creación del modelo

La clasificación por tono realizada es similar a otras técnicas que utilizan un modelo de color no realista basada en el modelo de color HSV [6] ya comentadas en capítulos anteriores. La idea básica es cambiar la distribución de materiales por un rango tonal parecido al que usan los ilustradores. Este rango tonal es una simplificación del conjunto de tonos disponibles, los cuales son parecidos a los que los artistas utilizan.

Previamente a este paso de selección de tono se ha de calcular las normales a partir de los datos originales, para ello la normal asociada al voxel P_j en la posición (x,y,z) viene dada por el gradiente tridimensional en esa imagen: $\vec{N}_j = \nabla_{(x,y,z)}(P_j)$.

Normalmente suele ser una buena aproximación usar alguna matriz de convolución similar al cálculo del gradiente, como puede ser un núcleo de Sobel o de Roberts en cada una de las 3 dimensiones, aprovechando las propiedades de descomposición de estos núcleos [42].

Este paso de detección de normales es necesario ya que de ellas dependerán los cálculos que se realicen con las luces virtuales. No hay problema en utilizar métodos más avanzados para la obtención de dichas normales con el uso de información procedente de la curvatura, lo que es bueno para un tipo de ilustración menos realista (con menos detalles y sombras) que la que tratamos de realizar en este capítulo [48].

Visualización del modelo

Una vez los tonos han sido calculados para los materiales, el proceso de visualización modifica la saturación y el brillo para obtener diferentes efectos. A diferencia de otros trabajos [6], las fuentes de luz utilizadas no son fuentes de luz convencionales sino que se utilizan fuentes de luz virtuales, las cuales no simulan un modelo de Lambert.

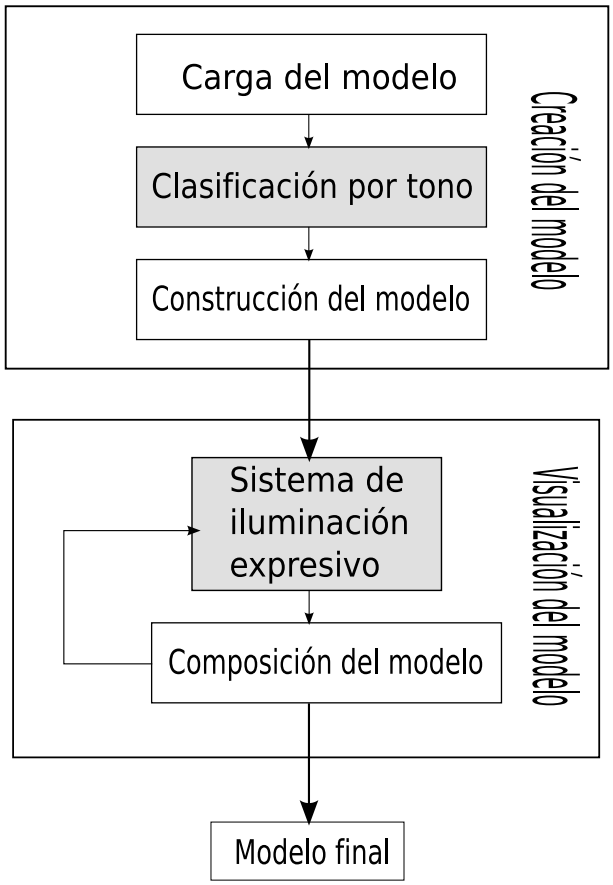

Figura 5.6: Cauce general del método

5.3.3. Modelo volumétrico de luces virtuales

El mayor problema para el cálculo de siluetas clásico es que el usuario no puede elegir donde situarlas ya que siempre suelen ir asociadas a la posición y dirección del observador. Este problema es serio ya que el modelo siempre aparece iluminado desde una posición frontal al observador. Además, en otras técnicas un cambio de cámara puede producir un recálculo de una serie de proyecciones o *búfferes* auxiliares con una orientación arbitraria utilizados para el cálculo de estas siluetas, lo que puede llevar a tener un problema que no es trivial de resolver cuando se utiliza una visualización directa mediante texturas [7, 72].

La solución al primer problema consiste en eliminar el enlace entre la dirección de la cámara y el cálculo de las siluetas, de forma que esas siluetas pasen a formar parte de las líneas de forma del modelo. Así se puede hacer una combinación de líneas de forma y siluetas con varias luces. Este modelo es contemplado por el modelo de luces virtuales [64], las cuales no iluminan por sí mismas sino que permiten una operación de extracción o alteración del modelo.

El usuario puede ajustar estas luces de una forma muy sencilla, ya que tan sólo hay que

modificar el vector de dirección de las mismas. De esta forma, una luz virtual está compuesta por un vector y una serie de parámetros. Estos parámetros irán asociados a la forma de visualizar los materiales que no son silueta o líneas de forma.

Además estas luces virtuales tienen una propiedad y es que no se basan solamente en la extracción de fronteras, sino en la modificación de las propiedades del objeto volumétrico. Por ello, aunque el efecto sea similar al producido en los modelos poligonales, la luz virtual volumétrica es ligeramente distinta a la luz virtual poligonal (tal y como veremos en el capítulo 7).

Definición del modelo de iluminación

Para mejorar las siluetas y visualizar imágenes que parezcan ilustraciones, estilizando las siluetas, las ecuaciones asociadas a S_j y V_j definidas en el capítulo 3 deben ser modificadas en base a una serie de parámetros para que las siluetas queden marcadas de un único color y el resto con otro, tal y como los ilustradores suelen dibujar.

El primer parámetro, k_s, define el nivel de saturación y lo usamos para definir S_j:

$$S_j = k_s \tag{5.4}$$

Si $k_s = 0$ lo que tenemos es una imagen en escala de grises, hay que tener en cuenta que muchas de las ilustraciones médicas utilizan solamente tonalidades en grises. Un parámetro $k_s = 1$ obtiene una imagen muy saturada en color que no suele ser apropiada para ilustraciones.

Esta sencilla ecuación simula el método de aplanamiento de color que se realiza en modelos poligonales mediante la aplicación de texturas unidimensionales. En el caso de los modelos poligonales el color viene definido mediante el valor de la textura, mientras que en nuestro trabajo el color viene definido tanto por el parámetro que se ajusta al tono como a k_s.

Los siguientes 3 parámetros los usaremos para definir el brillo. k_d es un parámetro que afecta a la claridad u oscuridad del sombreado interno y matices de la imagen final. k_a es un parámetro que define la cantidad y el ancho de las siluetas, su valor está entre 0 y 1. $k_a = 0$ implica que no haya siluetas, mientras que $k_a = 1$ significa que todo el modelo es una silueta. k_f es un parámetro que define cuantos detalles son introducidos en la imagen, y depende exclusivamente del histograma de cada volumen. Un buen valor (obtenido experimentalmente) es $k_f = 0{,}8$ (como se muestra en la figura 5.7).

El brillo queda modificado para mostrar simplemente dos tipos de tonos con algunas sombras, y es dado por la ecuación siguiente:

$$V_j = \begin{cases} 0 & \text{si } \vec{N}_j \cdot \vec{C} < k_a \\ X_j & \text{si } \vec{N}_j \cdot \vec{C} \geq k_a \end{cases} \tag{5.5}$$

donde X_j es dado por:

$$X_j = \begin{cases} 1 & \text{si } N_{j_x} > k_f \text{ ó } N_{j_y} > k_f \\ 1 - k_d & \text{en otro caso} \end{cases}$$

Las modificaciones de k_f como k_a se basan en las diferencias del gradiente calculado inicialmente, y que viene normalizado por los valores de la propia normal. El valor N_{j_z} no se usa ya que se supone que es el valor asociado al eje sobre el cual se han extraido los datos, y es

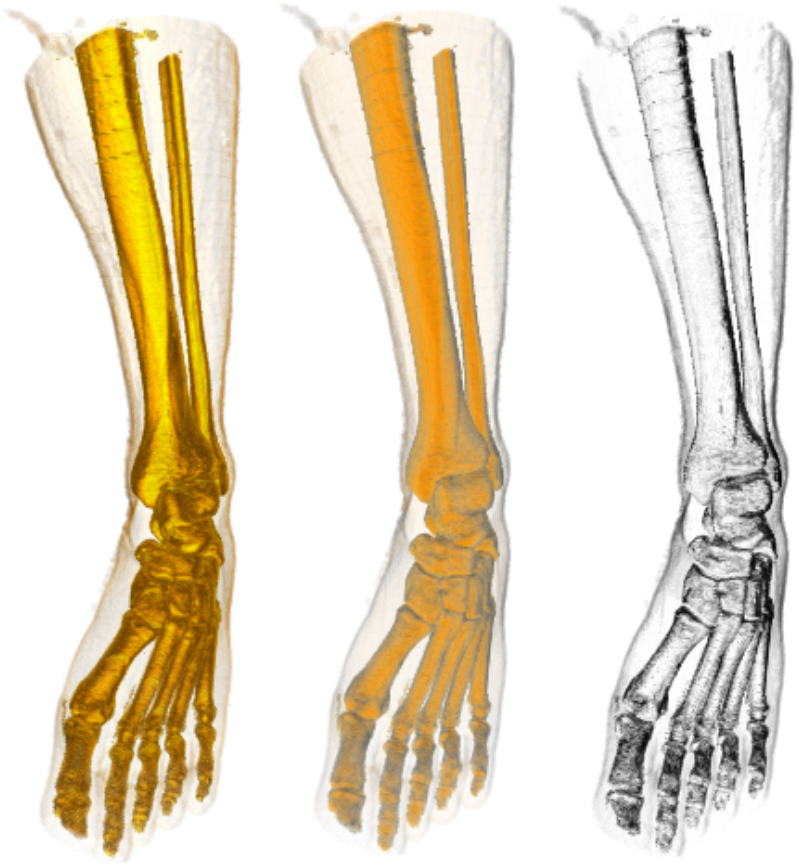

Figura 5.7: La mejora obtenida con el método presentado en este capítulo (a la derecha), con respecto a una iluminación basada en Lambert (a la izquierda) y una iluminación basada en el espacio de color HSV (al centro).

más fiable utilizar los valores de gradiente obtenido de los datos escaneados a más resolución (N_{j_x} y N_{j_y}) [43]. Todos estos parámetros permiten estilizar de formas diferentes las siluetas y el sombreado de los modelos.

Hasta ahora, la luz virtual solamente se ha aplicado cuando se ilumina el objeto, pero como se ha comentado inicialmente, puede ser desvinculada de la dirección del observador para producir líneas de forma. Para ello es necesario modificar las ecuaciones usadas para computar las siluetas.

De esta forma, si se modifica la ecuación 5.5 para usar, no la dirección de la cámara, sino la dirección de la luz virtual, podemos detectar siluetas desde cualquier posición, realzando líneas de forma o sombras adicionales (ver 5.8). Los parámetros usados para todas estas imágenes son descritos más adelante en la tabla de la siguiente sección.

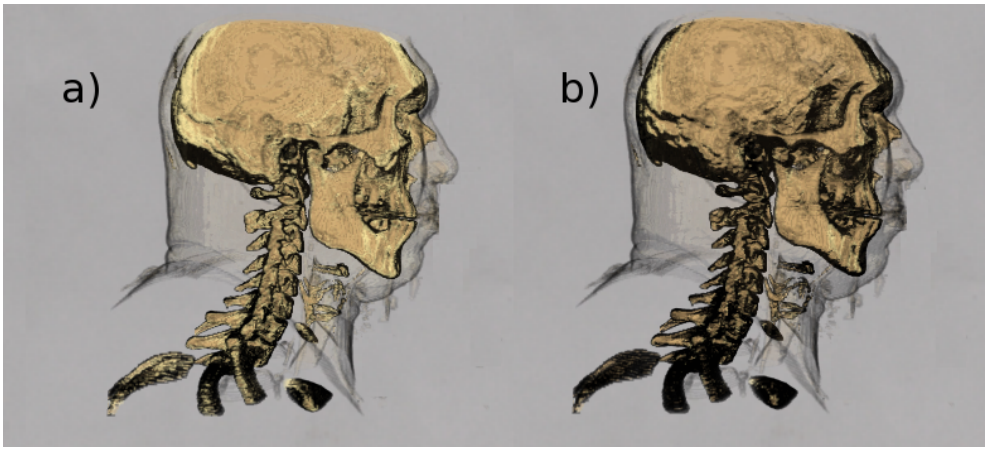

Figura 5.8: En la imagen a) la luz virtual coincide con la posición de la cámara. No ocurre así en la imagen b), en la cual se ha cambiado la luz virtual para que apunte ligeramente desde arriba hacia abajo. Puede verse el sombreado que se produce, realzando la ilustración producida.

La ecuación 5.5 es modificada por:

$$V_j = \begin{cases} 0 & \text{si } \vec{N}_j \cdot \vec{L}_i < k_a \\ X_j & \text{si } \vec{N}_j \cdot \vec{L}_i \geq k_a \end{cases} \tag{5.6}$$

El ilustrador puede modificar las sombras ajustando las luces virtuales, las cuales definen la apariencia de la ilustración final (figura 5.8). Si se sitúa una luz virtual en la dirección de la cámara lo que tenemos son siluetas, en otro caso tendremos sombras (ver figura 5.8).

5.3.4. Resultados

El método descrito ha sido aplicado a modelos con distintas configuraciones de parámetros. Todos los resultados obtenidos se han probado mediante una técnica de visualización mediante texturas 2D, obteniendo resultados similares con técnicas de visualización mediante texturas 3D y trazado de rayos, de forma que el método es independiente del algoritmo utilizado para la visualización del volumen, siempre y cuando se pueda calcular una normal a cada *voxel* y éste tenga una propiedad escalar o un color asociado.

Todos los resultados de este capítulo han sido obtenidos con un PC convencional con las siguientes características: procesador AMD Athlon 800 MHz, 640 MB de memoria RAM, una tarjeta gráfica GForce2 FX, un sistema operativo Linux con kernel version 2.6. Las diferencias de tiempo entre un modelo de Lambert y el modelo propuesto son despreciables.

El método funciona con diferentes elementos en el modelo visualizados al mismo tiempo, tal y como muestra la figura 5.8, que muestra ejemplos con carne (de forma casi totalmente transparente) y huesos (totalmente opacos).

El efecto de los parámetros sobre las imágenes resultantes se pueden observar en la figura 5.9. Dichos parámetros son los siguientes:

Imagen	k_a	k_d	k_s	k_f
a)	-	-	-	-
b)	0.60	0.50	0.60	0.80
c)	0.60	0.70	0.60	0.80
d)	0.60	0.10	0.60	0.80
e)	0.80	0.50	0.60	0.80
f)	0.20	0.50	0.60	0.80
g)	0.60	0.50	0.70	0.80
h)	0.60	0.50	0.10	0.80
i)	0.60	0.50	0.60	1.20
j)	0.60	0.50	0.60	0.10
k)	0.80	0.70	0.01	0.60
l)	0.75	0.20	0.80	0.10

k_a modifica las siluetas, también modifica el grosor de las mismas. k_d afecta a la oscuridad de las sombras internas en la imagen final. Si k_d toma un valor bajo la imagen aparece más plana. k_f solamente modifica la cantidad de detalles cuando k_d es alto, y funciona como un filtro incluyendo o quitando sombras internas en el modelo. Cuando se quiere simular una ilustración en tonos grises es conveniente ajustar los parámetros a un valor alto, salvo *ks*. Tal y como muestra la figura 5.10.

5.3.5. Modificación para sombreado mediante puntos

Como hemos visto en la sección de punteado, los resultados obtenidos hasta el momento en modelos volumétricos no se asemejan en nada a los obtenidos mediante modelos poligonales (ver figura 5.3), resultando un modelo en el cual es muy difícil distinguir formas concretas.

Esto es debido a la dificultad de aumentar la resolución de un modelo volumétrico por la falta de memoria del computador, y dificultad de mezclar elementos translúcidos (de hecho lo habitual es que los artistas únicamente dibujen la superficie de los objetos con esta técnica). Es por ello que hemos decidido realizar una técnica intermedia que, si bien no es punteado, es una técnica que sombrea el modelo mediante la inclusión de puntos aleatorios, y que se puede considerar híbrida puesto que puntea alguna de las zonas del modelo.

Para ello debemos cambiar el algoritmo de iluminación de la siguiente forma:

```
01. Para cada voxel (Vi) del volumen:
02.    N = ObtenerNormalDelGradiente (Vi)
03.    nl = (N · L)
04.    r = ObtenerAleatorioEntre (0, nl)
05.    Si r > t:
06.       cambiarColorDelVoxel (Vf, blanco)
07.    si no:
08.       cambiarColorDelVoxel (Vf, negro)
09.    Devolver (Vf)
```

donde donde *Vi* es el *voxel* de entrada del algoritmo, *N* es la normal que resulta de aplicar la función $ObtenerNormalDelGradiente(Vi)$, la cual calcula la normal a partir de un gradiente tridimensional de los valores de gris del volumen.

La función $ObtenerAleatorioEntre(0,nl)$ lo que hace es obtener una muestra uniforme sobre el rango de valores $(0,nl)$. t es un parámetro dado por el usuario que indicará el número de puntos (mayor cuanto más alto, menor cuanto más bajo).

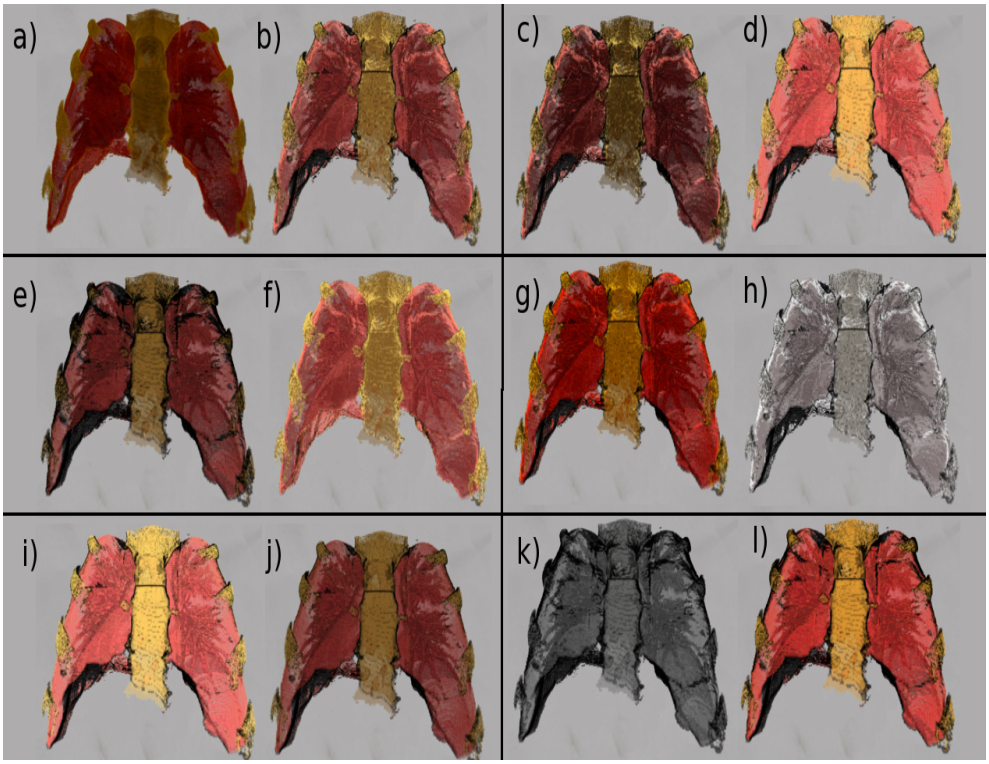

Figura 5.9: De izquierda a derecha, y de arriba a abajo: a) Iluminación realista (Lambert), b) iluminación propuesta con parámetros por defecto, c) y d) modifican la oscuridad y claridad de las sombras interiores (k_d), e) y f) modifican la detección de siluetas (k_a), g) y h) modifican el color de la ilustración (k_s), i) y j) modifican la cantidad de sombras interiores (k_f), k) los parámetros han sido ajustados para ilustración en tonos de grises, l) los parámetros han sido modificados para una ilustración a color

La función $cambiarColorDelVoxel(Vf, negro)$ cambia el *voxel* de salida Vf al color indicado como segundo parámetro. $devolver(Vf)$ simplemente fija el valor del *voxel* de salida.

Podemos aumentar también la resolución del modelo de salida, modificando el tamaño del *buffer* del volumen.

Los resultados obtenidos se pueden comprobar en las imágenes de la figura 5.11.

Como se puede apreciar, el algoritmo genera una distribución de probabilidad según la cercanía a la fuente de luz. Si la fuente de luz es perpendicular al gradiente del volumen obtenemos un valor cercano a 0 y por tanto una alta probabilidad de que exista un punto negro. En otro caso las probabilidades disminuyen. La probabilidad de que un punto con la normal idéntica a la dirección de la fuente de luz sea negro es 0.

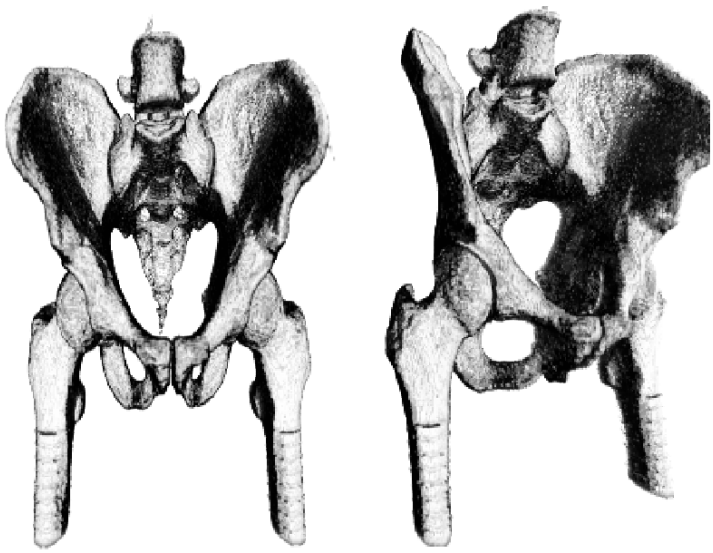

Figura 5.10: Todos estas ilustraciones han sido generadas con unos altos parámetros: $k_a = 0{,}7$, $k_d = 0{,}5$, $k_f = 0{,}9$. El parámetro de saturación es casi cero: $k_s = 0{,}01$. El resultado son ilustraciones realistas similares a las empleadas en medicina.

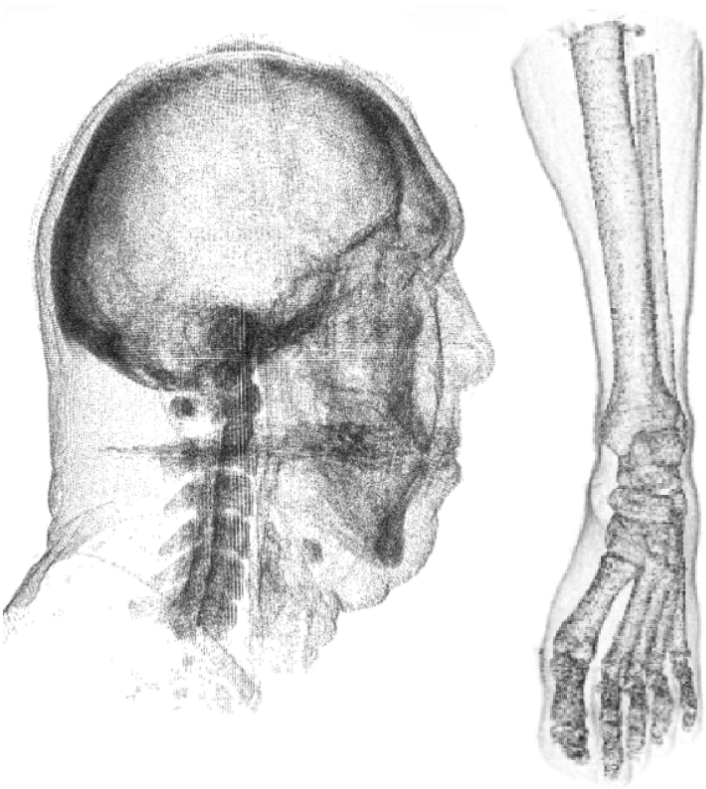

Figura 5.11: Ejemplo de sombreado mediante puntos cambiando la función de ilustración (a la izquierda), a la derecha se mezclan las dos técnicas de este capítulo para obtener una ilustración con siluetas y sombreado ($r = 0{,}45$ para ambas imágenes).

Capítulo 6

Visualización expresiva e ilustración a partir de imágenes

En este capítulo trataremos de realizar una revisión de los trabajos más importantes en cuanto a la creación de ilustraciones, para terminar con los trabajos realizados para simular ilustraciones en tinta y pluma a partir de imágenes bidimensionales y vídeos. Previamente definiremos las técnicas utilizadas por los artistas para crear imágenes no carentes de significado.

Introducción

La visualización expresiva ha sido muy utilizada sobre modelos poligonales, o a lo sumo aplicado al resultado obtenido a partir de estos modelos, ya que es relativamente sencillo obtener elementos de la geometría y no existe ruido en la imagen. En imágenes existen algunos trabajos recientes que analizaremos a continuación. Sin embargo, ninguno de los métodos es capaz de generar imágenes que realmente no parezcan filtros sobre la misma.

6.1. Visualización expresiva a partir de imágenes

En una imagen real, por ejemplo una fotografía, el proceso es mucho más complicado debido principalmente a que lo que tenemos no son objetos si no un conjunto de píxeles que a priori no tienen relación entre sí (suponiendo que hay ruido), además ni los objetos ni sus detalles son fácilmente reconocibles.

Para obtener atributos geométricos de la misma tenemos que aplicar técnicas de análisis de imágenes. Comentaremos algunas de las técnicas más usadas ya que cada autor suele emplear variaciones de técnicas básicas de análisis de imágenes para extraer la información de forma y colores.

6.1.1. Localización de elementos

En cuanto a la localización de fronteras, contornos, y en general de los objetos que conforman la imagen, el trabajo más relevante en cuanto a ilustración es el de Chen [18], en el cual se trata de detectar rasgos en las caras a partir de fotografías humanas.

El procedimiento general es el que se muestra en la figura 6.1, en el cual se utiliza una base de datos de ejemplos que se utiliza para realizar una comparación por los elementos en función de su posición y forma genérica. La imagen muestra los siguientes pasos: a) Fichero de entrada; b) Imagen descompuesta en componentes; c) Emparejamiento con cada componente respecto a los ejemplos de entrenamiento; d) Dibujos correspondientes de los elementos encontrados en la sección anterior; e) Composición mediante dibujado de las partes separadas como dibujo final.

Para poder representar estos dibujos a partir de esta secuencia sencilla de pasos, se realiza un lenguaje interno muy abstracto, realizando una representación del modelo realista a nivel de conceptos sencillos, como por ejemplo, una secuencia de expresiones faciales. Además se realiza una serie de limitaciones, como puede ser la posición de la cara, la iluminación de la

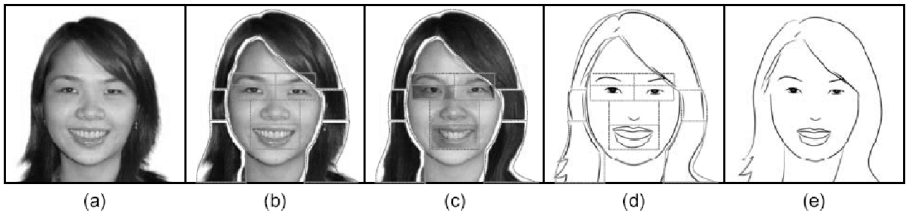

Figura 6.1: Procedimiento general del método de Chen.

misma o el número de expresiones faciales reconocidas. También es bastante problemático establecer una relación entre el conjunto de entrenamiento obtenido de forma empírica y el análisis visual de la cara.

Lo primero es determinar localmente cada uno de los componentes faciales, lo cual es bastante sencillo, teniendo en cuenta que se conoce a priori todas las posiciones aproximadas de dichos componentes.

Puesto que el pelo es más difícil de reconocer, en la primera fase se ignora, pasando el reconocimiento del mismo a ser un segundo paso del algoritmo. Por ello, al final hay que recomponer tanto el pelo como la cara de la persona para crear un dibujo final coherente.

Para cada componente facial se extrae la forma aproximada y la información de su textura asociada usando un modelo de forma activo [21]. Se construye un clasificador diferente para cada tipo de componente y se usa para clasificar los componentes de entrada en el prototipo apropiado. Se usa una interpolación por el *k*-vecino más cercano (kNN) y se usa con el conjunto de prototipos para calcular los parámetros de ajuste. Con esta información se puede sustituir cada uno de los elementos dibujados por los reales, la figura 6.2 muestra los distintos parámetros que se obtienen a partir de este algoritmo.

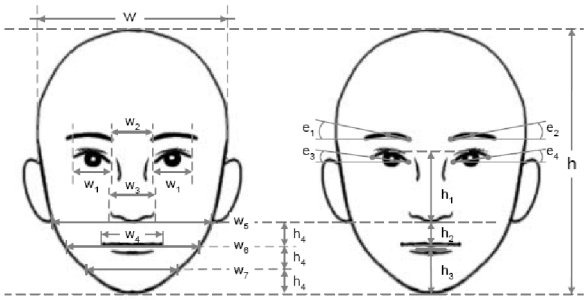

Figura 6.2: Parámetros utilizados para la búsqueda de elementos en el modelo de Chen.

Este método no requiere el cálculo de fronteras salvo para la parte externa de la cara, que es bastante sencilla de reconocer dado el cambio brusco de tonalidad con respecto al fondo, sin embargo para el pelo es más complicado ya que no se conoce la forma concreta del mismo y suele ser irregular.

Para el pelo, la imagen es seccionada en cinco regiones que se denominan componentes estructurales. Cada una de estas componentes están clasificadas en función de su posición con respecto a la cara. De forma que podemos saber si forma parte de terminaciones o no, y

si pertenece a una parte lateral, superior o inferior en el modelo.

Para determinar las características que serán comparadas en el conjunto de prueba del sistema se debe realizar un análisis de la imagen. Para ello se emplean dos técnicas principalmente, la máscara en el canal alfa (*alpha mask*) y los campos de orientación (*orientation fields*).

La máscara en el canal alfa se calcula para separar el pelo del fondo y de la cara. Un píxel usado para el pelo puede estar difuminado con el fondo o la cara debido a las propiedades de la cámara y al efecto producido por la distancia y el foco de la misma (*depth field*). Para ello basta con definir una máscara de un simple *bit* que identifique el tipo de punto en función a la cercanía del pelo (normalmente es sencillo identificar el pelo por el color, ya que en la mayoría de las imágenes de prueba el pelo es moreno, siendo en todos los casos bastante más oscuro que el resto de la imagen).

Sin embargo, detectar la orientación del pelo es algo más complicado. Se utilizan para ello varias máscaras que propagan la información de la orientación de las regiones altas en intensidad a las más débiles, de forma que el ruido es menor, además se requiere que el usuario indique manualmente hacia donde crece el pelo, lo cual hace que el sistema no sea totalmente automático.

Mientras que en el trabajo de Chen [18], el procesamiento en cuanto a análisis de imágenes es sencillo, el trabajo de Hertzmann [40] hace más incapié en el análisis de las imágenes ya que trabaja sobre video, aprovechando la coherencia temporal que ofrecen las imágenes sucesivas en el tiempo.

Este trabajo se basa en la minimización de la energía necesaria para realizar una obra artística a partir de una imagen de colores usando el mínimo número de trazos posibles. Esta función de minimización se utiliza para formular la forma de pintar como un problema de minimización de la energía. De tal forma que, dada una función de energía $E(P)$, se busca una pintura $P*$ (*painting*) con la mínima energía:

$$P* = arg \min_{P \in \rho} E(P) \tag{6.1}$$

A partir de esta expresión podemos definir una pintura $P*$ de la siguiente forma:

Definición 5. *Una pintura es definida como una colección ordenada de trazos de brocha como colores junto con un lienzo de un color o textura fijos.*

La pintura es representada por composición de diferentes trazos en un orden dado sobre el lienzo. Los trazos de brocha se representan como B-splines cúbicos muy gruesos. Los trazos se crean en el orden en el cual se han creado.

Cada función de energía puede crearse a partir de la combinación de varias funciones diferentes, permitiendo expresar una gran variedad de estilos, ya que cada función de energía corresponde a un estilo de pintura diferente.

Además se tiene en cuenta la distancia Euclídea en el espacio RGB para determinar la distancia entre los diferentes colores. Para la detección de los elementos del dibujo se inicializan las variables de la medida de las diferencias con los distintos píxeles de la imagen original G y la pintura creada con los obtenidos por un filtro de Sobel. Esto se realiza para obtener un énfasis adicional a la calidad de las fronteras.

El trabajo de DeCarlo [24] ofrece una alta calidad en cuanto a las imágenes obtenidas, los pasos principales de su algoritmo son:

- Adiestrar a un usuario, mirando la imagen un pequeño intervalo de tiempo, obteniendo el recorrido de su mirada.
- Dividir la imagen en partes usando un análisis visual mediante detección de fronteras y segmentación.
- Visualizar la imagen preservando la forma predicha al aplicar un modelo de percepción humano visual, según los datos del movimiento de los ojos.

Puesto que el primer paso no es posible simularlo, lo que se realiza es una calibración del sistema mediante un usuario utilizando una cámara para visualizar la escena durante un pequeño periodo de tiempo.

Posteriormente se construye una pirámide de imágenes según su resolución, lo que significa que se reduce la resolución de la imagen original hasta tener una imagen de muy poca resolución, usando para ello una proporción en la reducción.

Posteriormente, la segmentación se realiza sobre la imagen de menor resolución y luego se asciende en la pirámide hasta la imagen de mayor resolución y se va realizando el proceso en cada una de las etapas (figura 6.3). Por último se realiza un proceso de emparejamiento de cada una de dichas zonas combinando distintas zonas de la imagen de resolución mayor, para emparejarlas con la de resolución inmediatamente mayor.

Se construye, por tanto, un árbol de elementos segmentados, los nodos contienen propiedades tales como la región, su área, las fronteras y la media del color.

Puesto que el proceso de segmentación se basa en un algoritmo sencillo, es muy dependiente del ruido, debido a esto el proceso no es perfecto y las imágenes a utilizar deben tener una gran resolución y ser imágenes de colores planos.

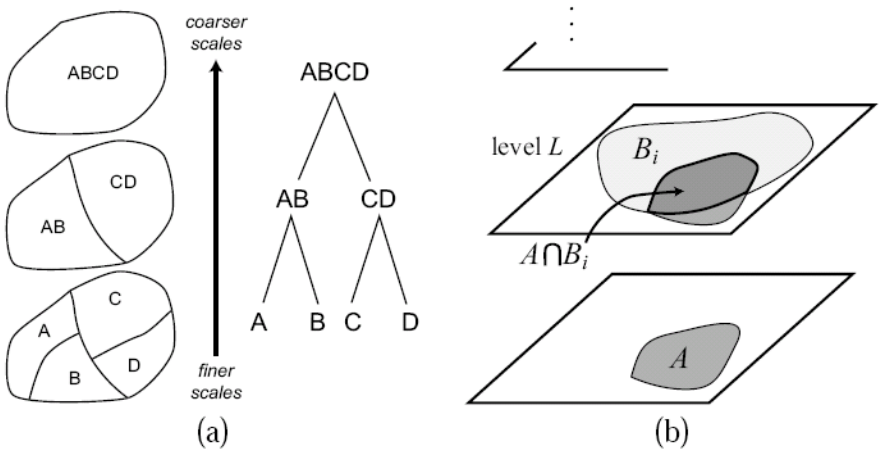

Figura 6.3: a) Segmentación jerárquica y su representación mediante un árbol; b) la regla de solapación usada para inferir esta jerarquía a partir de la pirámide de segmentación

Para el cálculo de cada una de las regiones, se hace uso de la frecuencia mediante el uso de una aproximación a la transformada de Fourier, de forma que la frecuencia es $f = \frac{1}{2D}$, donde D es el diámetro más pequeño del círculo que engloba al área. Una vez realizado el

proceso, se estima el contraste con las regiones vecinas mediante un algoritmo sencillo de simplificación de los tonos.

Para los contornos se utiliza la información del movimiento de los ojos del observador (mediante el uso de una cámara como se ha descrito previamente) para decidir el mejor camino, de esta forma se recorren de forma recursiva los trazos con este criterio hasta que se obtiene un trazo completo.

Se define la función SPLIT(n) para la región n en el paso de segmentación. Esta función será utilizada para detener el algoritmo de detección de contornos y se basa en la diferencia de contraste de las diferentes regiones obtenidas en el paso anterior.

Como desventaja principal de estas técnicas tenemos que necesitan colores para poder analizar la imagen y que muchas de ellas utilizan un sistema demasiado básico para el análisis de la misma, lo que hace que las formas sean difíciles de detectar por el sistema perceptual humano. Si bien es cierto que esto queda mitigado por la utilización de colores.

Además necesitan imágenes de muy alta resolución y son algoritmos muy lentos, lo que hace que sea tedioso el generar vídeos o una elevada cantidad de imágenes.

6.1.2. Estilización de los componentes

Puesto que los artistas no suelen dibujar los trazos con una única línea, es usual que se utilice un patrón a partir del cual se dividen una serie de trazos precalculados siguiendo dicho patrón[60]. En concreto, en el trabajo de Chen [18] se definen una serie de puntos guías a partir de la máscara anteriormente mencionada, y a partir de estos puntos guía se representa un trazo teniendo en cuenta la distancia entre el pelo y un estilo i determinado manualmente por el usuario.

Más simple es el trabajo de DeCarlo [24] en el cual se utiliza un filtro gaussiano con σ alto para emborronar el trazo, y una función que asocia el grosor a un parámetro de excentricidad calculado a partir de la frecuencia.

Sin embargo, en cuanto a la estilización del trazo, el trabajo de Hertzmann [40] es más elaborado. Se basa en la relajación del trazo adaptándolos para la función de energía usada en la pintura. Este método refina la colocación del trazo para elegir un máximo local dentro del mismo, para ello se realiza una búsqueda exhaustiva dentro del conjunto de puntos del trazo, mientras que se fijan el radio y el color, y el resto de la apariencia de la pintura. Además se modifica el algoritmo básico de *snakes* para incluir el número de puntos de control como una variable.

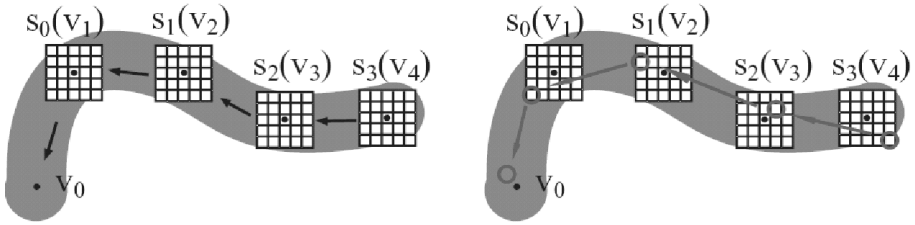

Figura 6.4: A la izquierda figura que muestra la técnica de programación dinámica para computar una forma de trazo óptimo localmente. A la derecha tenemos la energía del trazo óptimo de la longitud n, siempre buscando el camino mínimo.

A un alto nivel el algoritmo completo se basa en una serie de operaciones con los trazos:

- Reactivar el trazo: si un trazo de elimina debido a que se hayan producido modificaciones sobre la pintura, entonces se produce un desajuste en la energía global de la pintura, esta función hace que se recalcule la energía completa de toda la pintura.
- Alargar el trazo: si el radio del trazo está cercano al máximo, el radio debe incrementarse, y por tanto debe además cambiar la energía de la pintura haciendo uso de la función de reactivación.
- Compactar el trazo: si el radio del trazo está cercano al mínimo entonces el radio es decrementado y se reactiva el trazo debido a la pérdida de energía.
- Recolorear: el color de un trazo se activa a la media de los colores de la imagen fuente sobre todos los píxeles del trazo, además se llama a la función de reactivación para recalcular la energía.

Uno de los problemas principales es que el proceso es bastante lento, pudiendo llegar a tardar varias horas para imágenes de baja resolución (320x240 *píxeles*), siendo tedioso el generar vídeos.

6.2. Tinta y plumilla a partir de imágenes en escala de grises

Con este trabajo introducimos un algoritmo de seguimiento y reconstrucción de trazos aplicado como dibujo de tinta sobre imágenes en escala de grises. Para ello definimos un algoritmo basado en parte en un análisis de imágenes y por otra en una simulación del trazo del artista.

6.2.1. Introducción

La ilustración en libros es una técnica compleja y totalmente manual en la actualidad, cada vez más se intenta hacer este tipo de ilustraciones de forma automática a partir de modelos 3D. Sin embargo el proceso manual de modelado sigue estando ahí, y es deseable que fuera del todo automático. Es por ello que hay que intentar que la generación de este tipo de imágenes se pueda realizar a partir de fotografías, de esta forma, no solamente podríamos obtener modelos artísticos de la propia realidad sino también de modelos sintéticos, obteniendo la imagen artística a partir del *framebuffer*.

Como hemos visto en trabajos anteriores existen numerosas dificultades en el reconocimiento de los objetos y los trazos a partir de una imagen. Y en la representación de la misma. En este trabajo proponemos un método bastante rápido y sencillo para poder obtener imágenes de tinta a partir de fotografías. Puesto que el análisis de imágenes "con ruido" son complicadas o imposibles en el caso de no tener una base de conocimiento adicional, nos limitamos a utilizar imágenes o fotografías con el fondo plano, en el cual el objeto es fácilmente reconocible mediante un análisis perceptual descendente.

A partir de una misma idea podemos extender el trabajo para la realización de dibujos a plumilla y brocha o a brocha y esponja con una técnica de aguada.

El algoritmo que proponemos se basa en tres pasos fundamentales:

- Análisis de la imagen, obtención de contornos
- Reconstrucción de trazos a partir de contornos
- Estilización de los trazos

En el primer paso, obtenemos a partir de la imagen una serie de fronteras delgadas que se utilizará como esquema para reconstruir el trazo, sin embargo, antes de realizar un análisis de fronteras conviene alisar la imagen para eliminar el ruido, algo que es imprescindible ya que los filtros de detección de fronteras suelen ser muy dependientes del ruido de la imagen. Una parte fundamental del algoritmo es la reconstrucción de los trazos y el orden que se obtiene del mismo, ya que para los artistas no es indiferente la posición de inicio que la posición final del trazo, sobre todo en técnicas de ilustración con brocha y tinta.

Una vez obtenidos los diferentes trazos y el inicio y el fin de cada uno, se puede alterar dichos trazos mediante una estilización. Las posibilidades son, la modificación de la longitud de los mismos, la modificación del grosor, o la forma de los trazos. Todo ello se podrá realizar mediante una curva aproximativa (NURBs, Bèzier, etc.) o bien, mediante la superposición de diversos tipos de trazos.

Vamos a analizar el algoritmo por detalle en cada una de estas tres fases.

6.2.2. Fase de análisis

El primer paso es suavizar la imagen original mediante un filtro de emborronamiento (simulando la fase de imagen del cerebro). La ecuación que mejor aproxima esta función del cerebro es precisamente la ecuación de la campana de Gauss, que para nuestro caso particular, extendemos a dos dimensiones (debido a la bidimensionalidad de la imagen origen):

$$S(x,y,\sigma) = k \cdot e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{6.2}$$

donde σ es la desviación estándar de la función gaussiana. El término k sirve para que el área total de la función tenga siempre un valor constante de 1. De esta forma tenemos los valores de la función normalizados (y no perdemos información), su valor es $k = \frac{1}{2\pi\sigma^2}$, y se obtiene al despejar k de la ecuación de la campana gaussiana.

Con los valores discretizados realizamos una máscara de convolución y aplicándola a la imagen original O obtenemos la imagen suavizada I (ver figura 6.5.

Una vez hecho esto, pasamos a realizar el algoritmo de Canny. Definimos $G_x(I)$ como gradiente en x de la imagen I y $G_y(I)$ el gradiente en y de la imagen I.

Para calcular estos gradientes realizamos una convolución con los valores discretizados y normalizados de la derivada parcial en x e y de la función gaussiana, utilizamos esta ecuación porque es la que más se asemeja al cálculo realizado por la función del cerebro humano en la fase de esbozo primitivo. Otros filtros como Sobel y Roberts obtienen unos bordes demasiado anchos, y no son necesarios ya que la imagen I resultante de aplicar un filtro gaussiano ya está ausente de ruido blanco (que es el más común en las fotografías).

Las funciones utilizadas serán, por tanto, la siguientes:

$$\frac{\partial S(x,y,\sigma)}{\partial x} = \frac{-x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{6.3}$$

$$\frac{\partial S(x,y,\sigma)}{\partial y} = \frac{-y}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{6.4}$$

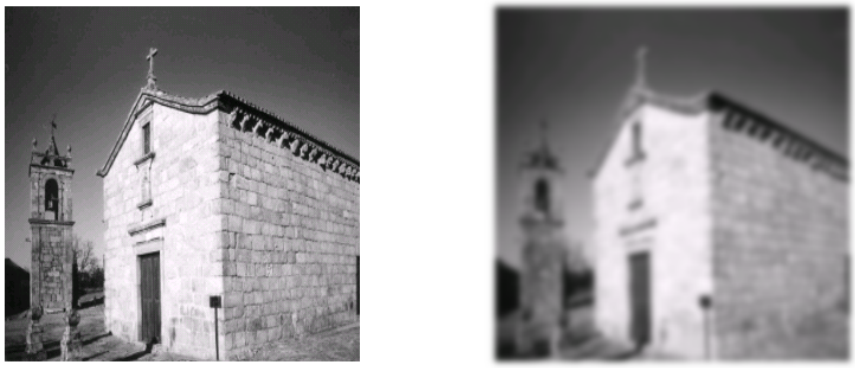

Figura 6.5: A la izquierda imagen original O. A la derecha, imagen I obtenida tras aplicar un filtro con un núcleo gaussiano con $\sigma = 8$.

Aplicamos el algoritmo de Canny, para ello, notamos $M = G_x(I)^2 + G_y(I)^2$, donde M es la imagen magnitud y $D = atan(\frac{G_y(I)}{G_x(I)})$, donde D es la imagen de direcciones. Entonces podemos suprimir de la imagen M los no máximos, para ello definimos dos umbrales tl y th, por tanto, $\forall p_{x,y} \in M$ sólo mantenemos en la nueva imagen de gradientes A los puntos que cumplan $tl < p_{x,y} < th$.

Tras aplicar el algoritmo de Canny (ya comentado en detalle en una sección del capítulo de análisis de imágenes), obtenemos una serie de valores representativos de las fronteras.

El problema principal que tenemos, es que hay partes que no son fronteras debido a que son demasiado pequeñas o aisladas. Para eliminarlas realizamos un proceso de histéresis siguiendo la heurística dada por el siguiente algoritmo (ver figura 6.6):

```
01. Para cada punto P1 dado por las coordenadas (x, y):
02.    Si A(x,y) no ha sido visitado:
03.       Si A(x,y) < tl entonces:
04.          F(x,y) = no-borde
05.          marcar P1 como visitado
06.       Si A(x,y) > th entonces:
07.          F(x,y) = borde
08.          marcar P1 como visitado
09.          seguir la dirección dada por D(x,y) desde
10.          P1 en ambos sentidos, mientras A(i,j) > tl:
11.             marcar el punto P2 dado por (i, j)
12.             marcar P2 como visitado
```

Como comentamos previamente, el problema de este algoritmo es que las fronteras en forma de Y no se detectan correctamente. Esto es corregido en el paso siguiente: la detección de trazos. Podemos apreciar en la figura 6.7 que las fronteras obtenidas con este método son de un solo píxel comparadas con otros filtros muy usados, que pueden parecer más suaves y que detectan mejor pero que únicamente obtienen píxeles cercanos a la frontera que molestan para detectar los trazos.

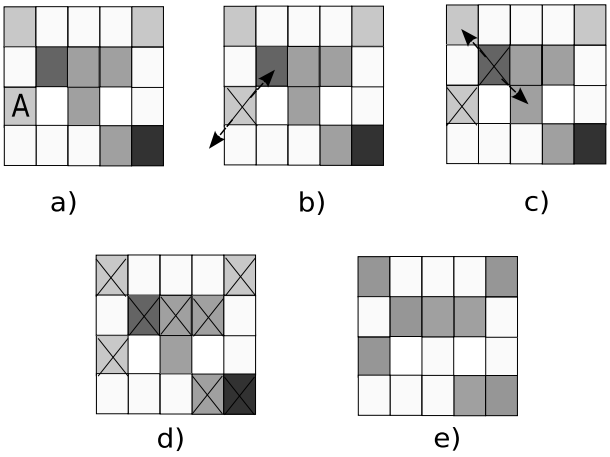

Figura 6.6: Paso de histéresis en el algoritmo de Canny. En esta figura los puntos blancos no cumplen el criterio de ser mayores a t_l a) Imagen de gradientes (magnitudes), comenzamos en el punto A; b) Una vez marcado como frontera seguimos en la dirección del gradiente de direcciones (en ambos sentidos); c) Marcamos el valor como frontera y seguimos la dirección del gradiente; d) Al final de aplicar el algoritmo tenemos marcadas las fronteras; e) Imagen final de fronteras, este es el resultado del algoritmo de Canny.

6.2.3. Detección y reconstrucción de trazos

Debido a que el algoritmo de Canny permite obtener fronteras de un sólo píxel de grosor podemos realizar un algoritmo que permita obtener trazos completos lo más largos posibles.

Nuestro algoritmo para detección de trazos se basa principalmente en una búsqueda de los píxeles de la imagen. Una vez localizado un píxel sin vecinos, o con el menor número de vecinos, introducimos dicho píxel P_i en una lista, y a partir de éste, introducir el píxel más cercano que no esté en esa lista y que esté dentro de una ventana de tamaño t_v.

La condición de parada del algoritmo es que no podamos saltar a ningún píxel que estando dentro de la ventana no halla sido visitado.

De esta forma, todos los píxeles que conforman una línea son tratados como la parte central del trazo y aquellos que estén aislados son el inicio del trazo o el fin del mismo.

El algoritmo es muy sencillo:

```
01. P[0] = Ventana (P, tv)
02. Si P[0] no existe:
03.    Fin
04. Si no:
05.    Obtener el pixel más cercano P[i] que no ha sido
       insertado aún
06.    Si P[i] existe:
07.      Volver al primer paso
08.    Si no:
09.      Insertarlo en la lista
```

tv es el tamaño de la ventana (tv $= t_v$), i es el índice de la lista.

Ventana es una función que determina el píxel no insertado aún que tiene menos vecinos a su alrededor, depende del parámetro t_v que es la distancia máxima de un píxel para que

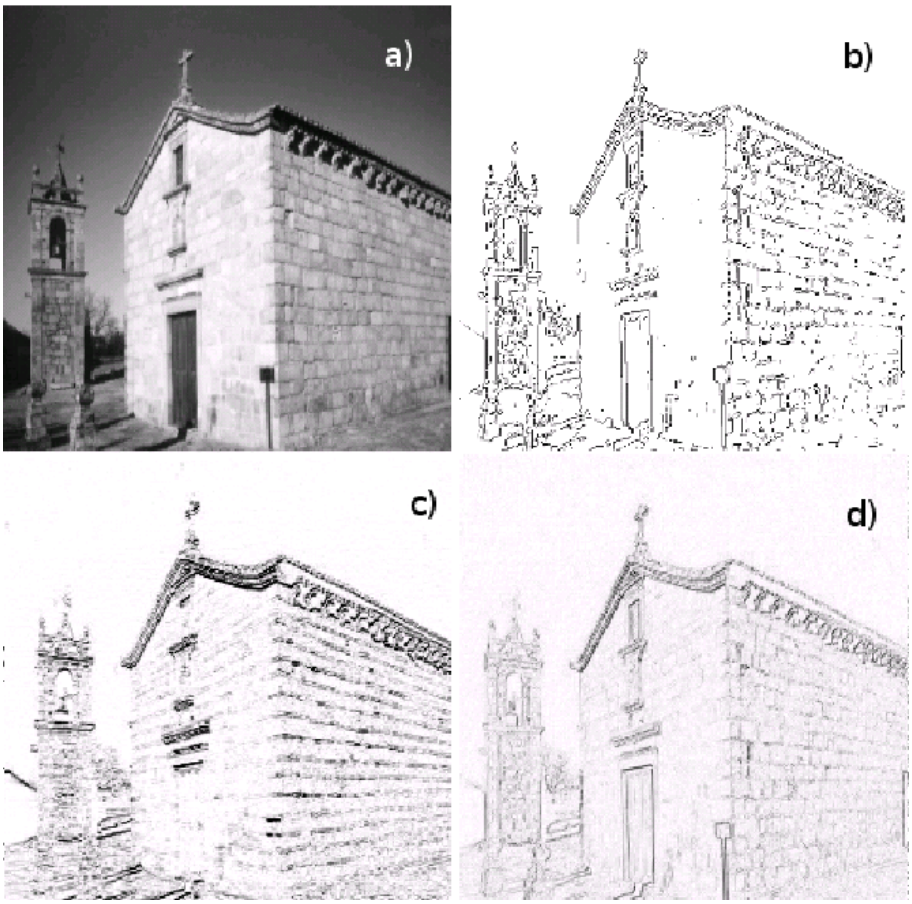

Figura 6.7: a) Imagen original *O*; b) Imagen tras aplicar el algoritmo de Canny con histéresis a *O*; c) Imagen tras aplicar un filtro de Sobel a *O*; d) Imagen obtenida tras aplicar un filtro de Roberts a *O*.

esté unido con el que estamos evaluando.

El problema de este algoritmo viene determinado por la arbitrariedad del punto P0 para cada trazo, de forma que podríamos tener trazos muy pequeños que en realidad eran uno mismo. Para evitarlo, en lugar de finalizar el algoritmo en el paso 2, invertimos la lista donde almacenamos el trazo y realizamos el algoritmo desde el paso 2 con P0 el primer punto del camino.

De esta forma el trazo tiene el orden de los píxeles obtenidos con el algoritmo de Canny. A partir de aquí montamos una lista con los ángulos y distancias de un píxel al siguiente. De tal forma que un píxel depende del anterior.

El algoritmo queda de la siguiente forma:

```
01. P[0] = Ventana (P, tv)
02. Si P[0] no existe:
03.    Invertir lista de pixels
04. Obtener el pixel más cercano P[i] que no ha sido
```

```
    insertado aún
05. Si P[i] existe:
06.    Volver al primer paso
07. Si no:
08.    Insertarlo en la lista
```

En esta lista podemos calcular entre un píxel y el siguiente un ángulo y su distancia en base a su posición en el plano. De esta forma la lista es una unión de píxeles con un determinado orden y unas propiedades de ángulo y distancia asociado a todos ellos (salvo el último), ver figura 6.8.

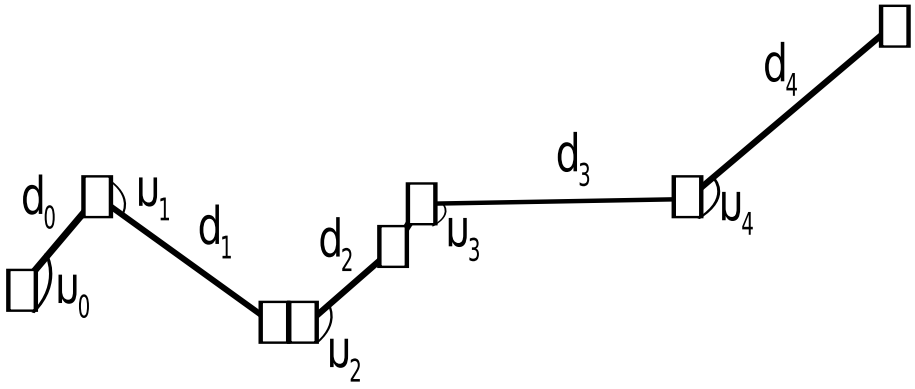

Figura 6.8: Diagrama del resultado tras aplicar el algoritmo de detección de trazos. Tenemos para cada píxel excepto el último, dos atributos asociados, la distancia d_i y el ángulo μ_i. Todos se encuentran en una lista ordenada desde el primero al último.

6.2.4. Estilización sencilla del trazo

Como hemos comentado, el ser humano no realiza trazos perfectos, es por ello que a los trazos que obtenemos con los filtros y la búsqueda de píxeles consecutivos les aplicamos diferentes modificaciones. Antes de entrar en más detalle recordemos las distintas técnicas que utilizan los artistas que dibujan con tinta sobre papel:

1. Cambios de ángulos y tamaños en los trazos
2. Grosores variables de pincel y brocha
3. Diferentes tipos de papel
4. Mayor o menor número de trazos

El cambio de ángulos en los trazos es trivial debido a la estructura que tenemos, ya que modificando los ángulos dados por μ_i (ver figura 6.8) hace que todo el trazo se abra o se cierre desde el origen. La introducción de ruido más o menos compensado en todo el trazo hace que los trazos no sean rectos.

De igual forma, el cambio de tamaño de trazos es también trivial, ya que si disminuimos las distancias dadas por d_i tendremos trazos más cortos. Igual que antes podemos introducir ruido a las distancias para que los trazos tengan tamaños más cortos o largos de forma arbitraria.

Además se nos permite hacer mayores cambios, ya que debido a que cada trazo está en una lista y consiste en una serie de puntos podemos partir estas listas según criterios tales como el ángulo o el tamaño del trazo. Y además podemos modificar los ángulos y la posición de cada uno de estos trazos, de forma que parezcan dos distintos.

Para los grosores del pincel definimos una función w que toma como parámetro el trazo. Esta función depende fundamentalmente del tamaño del trazo, y puede ser cualquiera aunque nosotros recomendamos la función

$$w(\phi) = \frac{\phi}{2} k_{max} \quad \text{si } \phi < \frac{1}{2} \tag{6.5}$$

y

$$w(\phi) = (\frac{\phi}{2} + \phi^2) k_{max} \quad si \ \phi \geq \frac{1}{2} \tag{6.6}$$

donde ϕ es el parámetro que varía entre 0 y 1 y k_{max} es el ancho máximo. Esta función se asemeja al grosor dado por un pincel a lo largo de un trazo, y se ha aproximado en base a distintos grosores de pinceles reales sobre un papel midiendo la distancia y el grosor. Esta función determinará como empieza y termina el trazo.

Para los diferentes tipos de absorción debidos al tipo de papel que se use se puede emplear otra función t, cuyos valores se utilizarán como entrada al canal alfa a la hora de dibujar los diferentes trazos. Igual que antes, t puede tomar cualquier valor, recomendamos $t(\phi) = \frac{acos(\phi)}{\pi} k_{max}$. Como antes, $\phi \in [0,1]$, que obtiene un color muy similar al real.

Definimos g_i como el resultado de aplicar la función de grosor y t_i como el resultado de aplicar la función de transparencia del trazo en el intervalo i. Entonces, para dibujar un trazo de estas características simplemente generamos dos puntos desplazados hasta p_i y p_{i+1} respectivamente y rotados $\frac{\pi}{2}$ radianes con respecto al ángulo μ_i (llamados p_j y p_{j+1} respectivamente), generamos el polígono formado por los puntos punto p_i, p_{i+1}, p_{j+1} y p_j.

Los trazos demasiado pequeños no los evaluamos, tan sólo los que son mayores a un valor umbral l. Para el color de la tinta recomendamos el color en RGBA dado por el vector: $(0{,}2, 0{,}2, 0{,}3, k_r \cdot t_i)$, con $k_r = 0{,}6$ o $k_r = 0{,}8$ y t_i el resultado de evaluar t en el trozo i ya que obtiene un color parecida a la tinta empleada en los bolígrafos (un azul cercano a los violetas).

Sin embargo, después de corregir los ángulos y las distancias aún sigue habiendo características que para un ser humano son difíciles de trazar a mano, debido a que muchos de los ángulos son rectos o forman esquinas.

Para evitarlo definimos los trazos a partir de curvas Bèzier, cuyos puntos de control vienen determinados por los píxeles de las listas de ángulos y distancias que teníamos sobre el punto inicial del trazo.

El resultado es la mezcla semi-transparente de los polígonos generados mediante las listas de trazos (con las modificaciones pertinentes a ángulos y distancias) y las curvas Bèzier (a las cuales se le aplica normalmente una función de grosor, puede estar definida por: $w(\phi) = \frac{acos(\phi)}{\pi} k$). Esta función ha sido aproximada a partir de pinceles reales, aplicando curvas suaves similares a las obtenidas con los polinomios de Bèzier. Una ventaja del uso de este tipo de curvas es que tenemos perfectamente delimitado el trazo, ya que el inicio y el fin coincidirá con el del trazo utilizado.

Al final se realiza una leve corrección (simulando la absorción del papel), dibujando la imagen semi-transparente con un ligero ángulo de rotación (desde $-\beta$ hasta β).

6.2.5. Resultados en tinta y plumilla para imágenes

Los resultados obtenidos con esta técnica varían según la cantidad de detalles en la imagen. Si la imagen no tiene casi detalles, obtenemos una imagen parecida a un esbozo y con unas pocas líneas se reconoce fácilmente el objeto visualizado (ver figuras 6.11 y 6.12). En las imágenes con muchos detalles obtenemos una imagen con muchos trazos que en conjunto forman la imagen a visualizar, como en la figura 6.9 con parámetros: $\sigma = 1$, $t_l = 0$, $t_h = 0{,}2$, $t_v = 4$, $l = 6$, $k_r = 0{,}6$ y como funciones de grosor $f_w(x) = \frac{acos(x)}{\pi k_{max}}$, y de transparencia $f_t(x) = \frac{acos(x)}{\pi k_{max}}$ para un tipo de trazo y $f_t(x) = xk_{max}$ para el otro, o la figura 6.10) donde la primera imagen tiene los parámetros siguientes: $\sigma = 1$, $t_l = 0$, $t_h = 0{,}07$, $t_v = 3$, $l = 4$, $k_r = 0{,}6$ y como funciones de grosor $f_w(x) = \frac{x}{2}k_{max}$ si $x < \frac{1}{2}$, y $f_w(x) = (\frac{x}{2} + x^2)k_{max}$ si $x \geq \frac{1}{2}$, y de transparencia $f_t(x) = \frac{acos(x)}{\pi k_{max}}$ para los dos tipos de trazo. Y la segunda imagen $\sigma = 1$, $t_l = 0$, $t_h = 0{,}1$, $t_v = 3$, $l = 4$, $k_r = 0{,}8$ y como funciones de grosor $f_w(x) = \frac{acos(x)}{\pi k_{max}}$, y de transparencia $f_t(x) = \frac{acos(x)}{\pi k_{max}}$ para un tipo de trazo y $f_t(x) = xk_{max}$ para el otro.

Figura 6.9: Método aplicado a un modelo orgánico.

Los resultados son bastante similares a los dibujados con brocha y plumilla, sin embargo este tipo de dibujos no son usuales debido a que se suele mezclar una técnica de aguada para las sombras, además suele ser complicado el tener que estar dependiendo de tantos parámetros. Lo ideal sería que se pudieran obtener imágenes aceptables con una pequeña cantidad de parámetros, e incluso que estos parámetros fueran opcionales.

Esto se puede solucionar añadiendo información visual, como por ejemplo sombras, a la imagen generada.

6.3. Ampliación para técnica de aguada

El algoritmo anterior tan solo utilizaba un par de trazos para dibujar la imagen, de forma que los trazos eran suaves y anchos. Si modificamos la cantidad de trazos, y en lugar de hacer

Figura 6.10: Diferentes parámetros y funciones de dibujado, los resultados obtenidos son similares con distintos trazos.

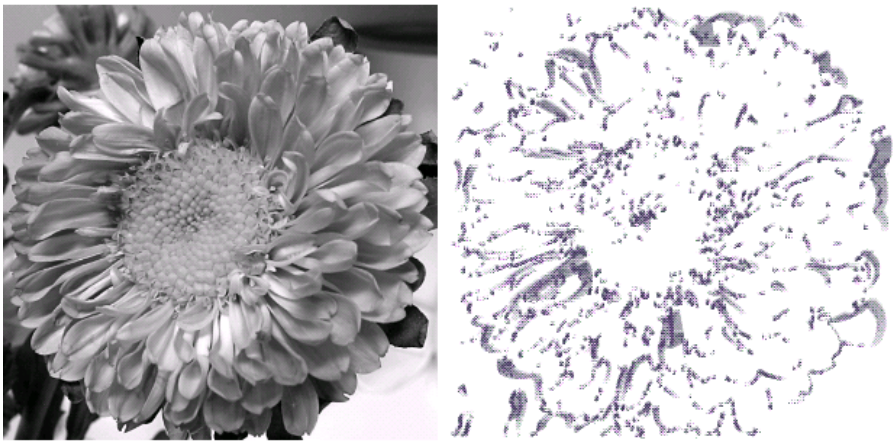

Figura 6.11: Para los elementos de muchos detalles conviene un t_h entre 0.2 y 0.4, para esta imagen hemos utilizado: $\sigma = 1$, $t_l = 0$, $t_h = 0{,}2$, $t_v = 4$, $l = 3$, $k_r = 0{,}2$

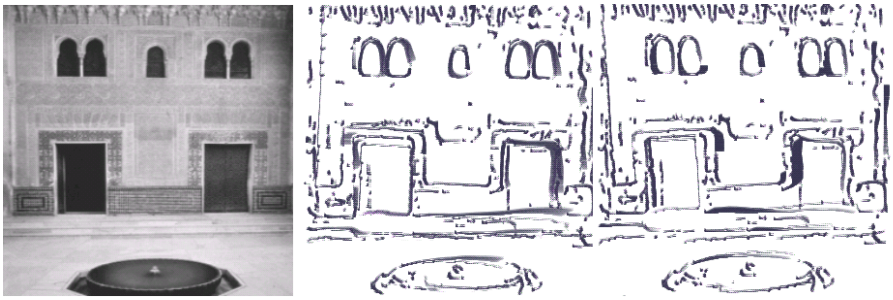

Figura 6.12: Cambio en los parámetros del trazo Bèzier y en los ángulos de la misma. El efecto es similar a un cambio de sombreado en el dibujo. Los parámetros usados en ambas imágenes son: $\sigma = 1$, $t_l = 0$, $t_h = 0{,}1$, $t_v = 4$, $l = 3$, $k_r = 0{,}8$.

dos pasadas, una con Bèzier y otra sin aproximación, lo que hacemos es superponer 8 trazos,

con diferentes funciones de grosor y alteración de la línea, el resultado es más parecido al que un artista suele realizar, ya que para cada trazo el artista no suele dar una única pasada, si no que realiza varias hasta obtener todos y cada uno de los detalles 6.13.

Figura 6.13: Cauce de procesamiento general del algoritmo.

Además podemos manipular cada uno de los trazos como si fueran polígonos en OpenGL, escalándolos, rotándolos o trasladándolos.

De esta forma modificamos el algoritmo global, realizando un bucle con diferentes alteraciones de trazos tal y como muestra el siguiente algoritmo:

```
01. trazo = CrearTrazos ();
02. trazo1 = AlterarTrazos (trazo);
03. Para cada estado i en n:
04.    Si i <> 0:
06.       DestruirTrazos (trazo2);
07.    trazo2 = AlterarTrazos (trazo1);
08.    DestruirTrazos (trazo1);
09.    trazo1 = AlterarTrazos (trazo2);
10.    .
11.    .
12.    .
```

donde n es el número de alteraciones de trazos a realizar ($n = 3$ es un buen compromiso entre velocidad y calidad de imagen). *AlterarTrazos* es una función que modifica el trazo según las siguientes heurísticas sobre los trazos (ver figura 6.8):

- Aumentar cada ángulo de cada trazo: esto permite elevar los trazos
- Disminuir cada ángulo de cada trazo: esto baja los trazos
- Aumentar y disminuir cada ángulo del trazo: esto hace los trazos irregulares
- Aumentar la distancia entre ángulos: aumenta el tamaño del trazo y hace que se pierda ligeramente la forma
- Disminuir la distancia entre ángulos: reduce el tamaño del trazo y aumenta los giros
- Aumentar y disminuir la distancia entre ángulos: esto hace que el trazo pierda la forma original

Mezclando algunas de estas heurísticas podemos realizar alteraciones leves a los trazos para que en un paso posterior podamos representarlos.

Además se modifica la visualización de cada trazo con el siguiente algoritmo:

```
01. Para cada estado (e) de 1 a k:
02.    En caso de que e sea:
03.       0:
04.          DibujaPoligonos (traz, 0, 0, 0.01, 6, 6, transparen);
05.          DibujaBezier (traz, 20, 0.03, 6, 1, transparen);
06.       1:
07.          DibujaPoligonos (traz, 3, 3, 0.01, 6, 4, transparen);
08.          DibujaBezier (traz, 20, 0.04, 0, 0, transparen);
09.    .
10.    .
11.    .
```

donde k es el número de trazos a dibujar con diferente estilo ($k = 8$ es un buen compromiso entre velocidad y calidad de imagen).

DibujaPoligonos es una función que dibuja los polígonos sin utilizar una curva Bézier para ello. Mientras que *DibujaBezier* es el equivalente utilizando los nodos del trazo como puntos de control. Ambas funciones tienen una serie de parámetros que pasamos a describir a continuación:

- Trazos: Los trazos originales sobre los que dibujar los polígonos o tomar los puntos de control.
- Alfa1: Un valor alfa para el color de la tinta en el origen
- Alfa2: Un valor alfa para el color de la tinta en el destino
- Valor: Un valor que se aplica a alguna de las siguientes funciones para el grosor y transparencia del trazo (x representa la longitud del trazo normalizada):

  1. $y = \text{Valor}$
  2. $y = x \times \text{Valor}$
  3. $$y = \begin{cases} x/2 \times \text{Valor} & \text{si} x < 0{,}5 \\ x/2 \times \text{Valor} + x^2 \times \text{Valor} & \text{si} x \geq 0{,}5 \end{cases}$$
  4. $y = \text{Valor} \times \cos(x \times \pi)$

5. $y = \text{Valor} \times \sin(x \times \pi) + \text{Valor} \times \cos(x \times \pi)/2{,}0$
6. $y = (\text{Valor} \times \sin(x \times \pi \times 4) + x \times \text{Valor})/2{,}0$
7. $y = \text{Valor} \times \sin(x \times \pi) + \text{Valor} \times \cos(x \times \pi)/2{,}0$

- FuncionGrosor: Indica el número de la función en la lista anterior a aplicar sobre el grosor del trazo en función de la distancia.
- FuncionTransparencia: Indica el número de la función en la lista anterior a aplicar sobre la transparencia del trazo en función de la distancia.

Estas funciones se estiman a partir de grosores reales de pinceles para tinta. Además añadimos un paso adicional de paletización del fondo para simular la aguada. El algoritmo de paletización del fondo es el siguiente:

```
01. Para cada tono (b[i,j]) de la imagen:
02.    Si b[i,j] < 25:
03.       f[i,j] = 25
04.    Si no:
05.       Si b[i,j] < 50:
06.          f[i,j] = 50
07.       Si no:
08.          Si b[i,j] < 100;
09.             f[i,j] = 100
10.          Si no:
11.             Si b[i,j] < 200:
12.                f[i,j] = 150
13.             Si no:
14.                f[i,j] = 255;
```

donde $f[i, j]$ representa el nivel de intensidad de la imagen de salida. Hay que observar que los tonos se han dividido de forma uniforme, sin utilizar el histograma de la imagen. Esto es debido a que el histograma de las fotografías en escala de grises es muy similar, variando entre zonas muy claras y muy oscuras, siendo innecesario aproximar más la imagen.

Resultados

Podemos observar que los resultados que obtenemos en la figura 6.14 son bastante buenos para unos umbrales estándar (son los mismos que la imagen de la figura 6.9), además podemos variar los colores del fondo o del trazo de la tinta.

El método también funciona con vídeo, de hecho funciona mejor que con imágenes ya que utiliza la robustez del algoritmo de Canny para obtener las posiciones de los trazos, de forma que podemos observar como se aprecian mejor los objetos de las animaciones de las figuras 6.15 y 6.16, sobre todo en escenas de agua.

Como ventaja principal tenemos que, a diferencia del resto de trabajos en visualización expresiva de imágenes, el método aquí propuesto funciona sobre el canal de luminancia únicamente, pudiendo ser mejorado en un futuro utilizando el resto de canales de colores en una imagen en color.

Figura 6.14: Resultados con la modificación introduciendo distintos trazos y paletización del fondo.

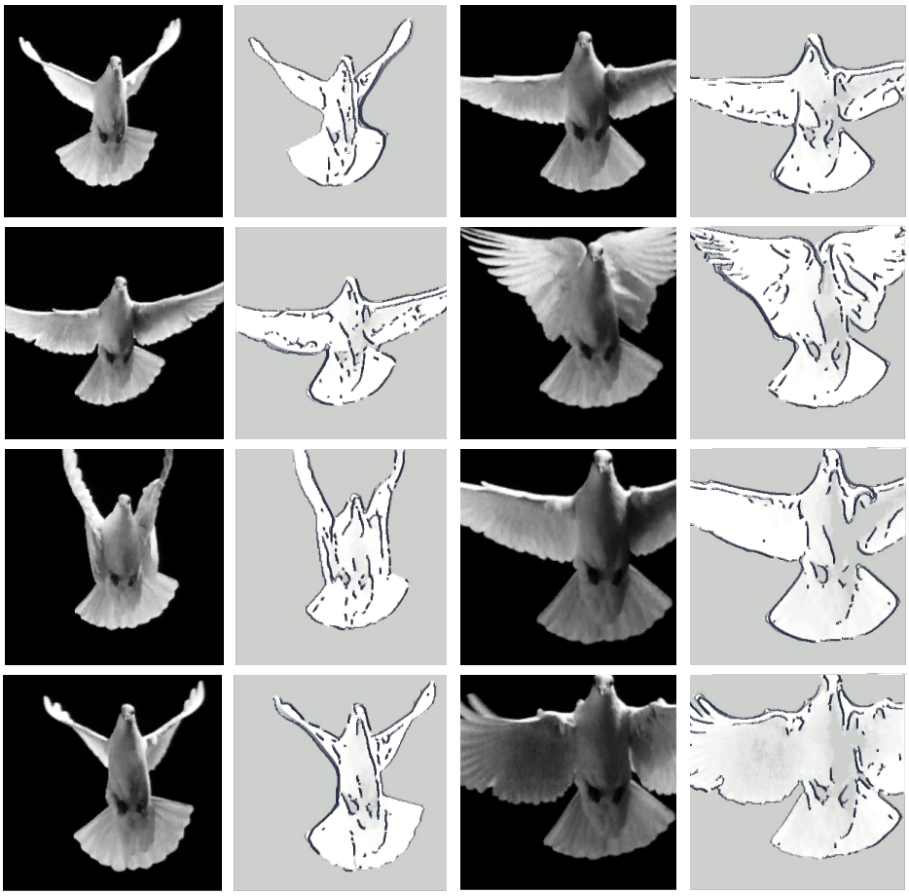

Figura 6.15: Resultados aplicando el método a los cuadros de un vídeo.

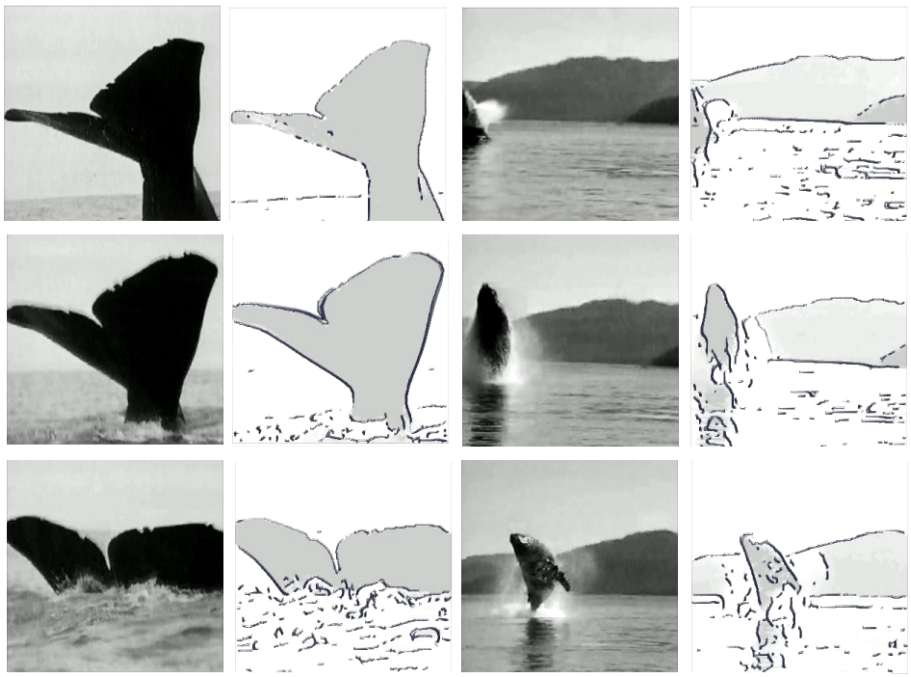

Figura 6.16: Resultados sobre agua, el resultado es muy similar al que los artistas suelen obtener. Los artistas usan este tipo de técnicas con tinta y aguada para dibujar el agua en las ilustraciones.

Capítulo 7

Luces virtuales volumétricas

En este capítulo veremos una introducción al lambda-cálculo y como a partir de esta definición formal de computación podemos definir un modelo de luces virtuales ampliado para volúmenes. Además se comentará la adaptación computacional de dicho modelo de luces virtuales.

Introducción

Como hemos visto en los sucesivos capítulos, unos pequeños cambios en la función de iluminación pueden resultar en una apariencia completamente diferente en la salida de un algoritmo de visualización expresiva.

Sin embargo, es complicado trabajar con algoritmos tan distintos sin tener ninguna base formal sobre la cual trabajar. Lo que se propone en este anexo es una definición de base de las luces virtuales en modelos volumétricos y un método de trabajo y composición con las mismas.

Para ello vamos a realizar primero una introducción al *lambda*-cálculo que es sobre lo que basaremos nuestro sistema y por tanto, nuestras definiciones.

7.1. λ-cálculo

El λ-cálculo es un sistema formal diseñado para investigar la definición de funciones, la aplicación de las mismas y la recursión. El cálculo puede usarse para definir de forma clara cómo de computable es una función. La pregunta de si dos expresiones de λ-cálculo son equivalentes o no, puede ser resuelta por un algoritmo general, y esta fue la primera de las preguntas que se realizó sobre esta técnica [8, 9].

El λ-cálculo ha influenciado a muchos lenguajes de programación tales como Lisp, Haskell, Python, y otros. En general se puede expandir el λ-cálculo para que admita tipos de datos y por tanto, crear un lenguaje de programación funcional (es lo que haremos con nuestras luces virtuales).

7.1.1. Descripción informal del λ-cálculo

En el λ-cálculo cada expresión se basa en una función con un único argumento. El argumento de la función es otra función de un argumento y el valor devuelto de la función es otra función con un argumento.

De esta forma, la función de añadir dos unidades al valor actual de un parámetro ($f(x) = x+2$) sería expresada en λ-cálculo como $\lambda x.x+2$ (o de forma equivalente a $\lambda y.y+2$). Consideremos la función que toma una función como argumento y lo aplica a su argumento, por ejemplo el valor 3, en λ-cálculo se expresaría como: $\lambda x.x\,3$.

De forma que para aplicar $f(3)$ según la función f anterior, escribiríamos: $(\lambda x.x+2)3$. La aplicación de las funciones es asociativa por la izquierda: $f\,x\,y = (f\,x)y$.

Si queremos componer las funciones anteriores para realizar 3 + 2, se escribirían de la siguiente forma: $(\lambda x.x\ 3)(\lambda x.x+2)$ y se resolvería con $(\lambda x.x+2)3$ y $3+2$.

Aunque hemos comentado previamente que una función en *lambda*-cálculo tiene un único parámetro, esto no es totalmete cierto, en realidad se pueden expresar funciones con varios parámetros pero se expresan como una función que devuelve una función como argumento. Por ejemplo, la función $f(x,y) = x - y$ se escribiría como: $\lambda x.\lambda y.x - y$. Las tres expresiones siguientes son equivalentes: $(\lambda x.\lambda y.x - y)\ 7\ \ 2$, $(\lambda y,7 - y)2$ y $7 - 2$.

Además el λ-cálculo se basa en la recursividad, de forma que el intentar desarrollar la expresión $\lambda x.x\ x)(\lambda x.x\ x)$ es recursiva e infinita.

Las λ-expresiones pueden contener variables libres, es decir, variables no ligadas a ningún λ y son parecidas a las variables globales en un programa. Por ejemplo, la variable y es libre en la expresión $(\lambda x.y)$, esta función siempre devuelve y independientemente del x entrante. Puede ser necesario, por tanto, el tener que renombrar las variables ligadas para que no exista confusión con las variables libres:

$$(\lambda x.\lambda y.y\ x)(\lambda x.y) \Longleftrightarrow \lambda z.z(\lambda x.y) \tag{7.1}$$

7.1.2. Definición formal de λ-cálculo

Formalmente comenzamos con una serie de identificadores, en principio, infinito $\{a, b, c, \ldots, x, y, z, \ldots, x_1, x_2, \ldots\}$. El conjunto de todas las λ-expresiones se pueden obtener según se describe en la siguiente gramática libre de contexto expresada en BNF:

1. <expr> ::= <identificador>
2. <expr> ::= (λ <identificador>.<expr>)
3. <expr> ::= (<expr> <expr>)

Las primeras dos reglas generan las funciones, mientras que la tercera describe la aplicación de una función a un argumento. Normalmente se suelen eliminar los paréntesis si no hay ambigüedad, asumiendo que la aplicación de funciones es asociativa por la izquierda. De forma que la expresión $((\lambda x.(x\ x))(\lambda y.y)$ puede escribirse simplemente como $(\lambda x.x\ x)\lambda y.y$.

Las expresiones tales como $\lambda x.(x\ y)$ no definen funciones ya que la variable y es libre (no está ligada a ninguna λ).

Las variables son ligadas según las siguientes reglas:

1. En una expresión de la forma V donde V es una variable, esta V es libre.
2. En una expresión de la forma $\lambda V.E$, las ocurrencias libres son las ocurrencias libres en E excepto V. En este caso las ocurrencias de V en E están ligadas al λ anterior a V.
3. En una expresión de la forma $(E\ F)$ las ocurrencias libres son las ocurrencias libres de E y F.

Para indicar que dos funciones son la misma en λ-cálculo, usamos el símbolo $==$, de forma que: $\lambda x.x == \lambda y.y$ indica que ambas funciones son la misma.

A partir de tres reglas sencillas que expondremos a continuación tendremos definido todo el λ-cálculo.

α-conversión

La regla de α-conversión intenta expresar la idea de que los nombres de las variables ligadas no son importantes. Por ejemplo $\lambda x.x == \lambda y.y$.

La regla de α-conversión obliga a que si V y W son variables, E es una λ-expresión, y $E[V/W]$ significa la expresión E con cada ocurrencia de V en E reemplazando W, entonces:

$$\lambda V.E == \lambda W.E[V/W] \tag{7.2}$$

si W no aparece libre en E y W no está ligada por un λ en E siempre que reemplace a V.

β-reducción

La regla de β-reducción expresa la idea de la aplicación de una función. Esta regla se expresa como:

$$((\lambda V.E)F) == E[V/F] \tag{7.3}$$

si todas las ocurrencias en F son libres en $E[V/F]$. La relación $==$ queda definida como la relación de equivalencia más pequeña que satisface estas dos reglas.

η-conversión

La tercera regla, es la η-conversión. Esta puede añadirse de dos formas como una nueva relación de equidad. La η-conversión expresa la idea de extensionalidad, la cual, en el contexto en el que dos funciones son la misma si y solo si tienen el mismo resultado tras aplicar los argumentos.

Si f y g son extensionalmente equivalentes, por ejemplo si $a == g$ para toda λ-expresión a, entonces en particular, tomando a para ser una variable x no libre en f tenemos $f\,x == g\,x$ y por tanto, $\lambda x.f\,x == \lambda x.g\,x$, y de esta forma $f == g$.

De igual forma podemos obtener esta regla a partir de la β-reducción: $(\lambda x.f\,x)y == f\,y)$, tenemos $(\lambda x.fx == f)$.

A partir del λ-cálculo podemos obtener una aritmética en números naturales e incluso una serie de funciones condicionales y recursión. Aunque todo esto no nos interesa demasiado ya que utilizaremos una extensión del λ-cálculo denominada programación funcional.

7.1.3. Programación funcional

El λ-cálculo como tal tiene el problema de que los datos con los que tratamos son totalmente booleanos o como máximo números naturales, por ejemplo:

$$\text{VERDAD} = \lambda u.\lambda v.u$$

$$\text{FALSO} = \lambda u.\lambda v.v$$

$$0 = \lambda f.\lambda x.x$$

$$1 = \lambda f.\lambda x.f\,x$$

$$2 = \lambda f.\lambda x.f\,(f\,x)$$

$$3 = \lambda f.\lambda x.f\,(f\,(f\,x))$$

y así. Pero no permite trabajar con datos complejos como reales o en nuestro caso, volúmenes o puntos tridimensionales. Para ello se crearon los lenguajes de programación funcionales, los cuales permiten definir tipos sobre las funciones del λ-cálculo, estos lenguajes se basan en la correspondencia de Curry-Howard sobre el λ-cálculo.

La correspondencia de Curry-Howard es una relación entre la definición de lenguajes de programación y la matemática que define el λ-cálculo, y es lo que nos permite extender el λ-cálculo para ser útil desde el punto de vista computacional.

Esta correspondencia es conocida como isomorfismo de Curry-Howard o la correspondencia de la formulación-tipos[8]. La correspondencia puede verse desde dos niveles, primero el que verifica que el valor calculado por una función es análogo al teorema lógico, y segundo que el programa computado es análogo a la comprobación de ese teorema. En concreto, nos interesa el que se puedan definir tipos.

Tipos en λ-cálculo

Siguiendo la notación del λ-cálculo, usaremos $\lambda x.E$ para denotar la función con un parámetro formal x y el cuerpo de la función dado por E. Cuando aplicamos un valor a a un argumento, la función sustituye el valor de x por a. Las expresiones de λ-cálculo válidas tienen alguna de estas formas:

- v: donde v es una variable
- $\lambda v.E$: donde v es una variable y E es una λ-expresión
- $(E\ F)$: donde E y F son expresiones del λ-cálculo

Los tipos dependen de las variables tipo, las cuales serán denotadas por letras minúsculas griegas (α, β, etc.). Denotaremos un tipo por variable, de forma que si x tiene un tipo α y la expresión E tiene un tipo β, entonces $\lambda x.E$ tiene un tipo que se nota como: $\alpha \longrightarrow \beta$, que significa que es una función que espera un tipo α y la expresión completa tiene un tipo β.

Por ejemplo, considerando la función $\mathrm{K} = \lambda a.\lambda b.a$. Supongamos que a tiene tipo α y b tiene tipo β. Entonces $\lambda b.a$ tiene tipo $\beta \longrightarrow \alpha$, y $\lambda a.\lambda b.a$ tiene tipo $\alpha \longrightarrow (\beta \longrightarrow \alpha)$. Puesto que el operador $\longrightarrow$ es asociativo a la derecha, podemos escribir: $\alpha \longrightarrow \beta \longrightarrow \alpha$.

Según esta notación, en un lenguaje funcional tendremos una función definida por una λ-expresión y una expresión de este tipo para determinar el tipo de la misma. Por ejemplo, podemos definir una función que suma dos enteros y devuelve otro como:

$$\text{entero} \longrightarrow \text{entero} \longrightarrow \text{entero}$$
$$\text{SUMA} := \lambda x.\lambda y.x\ y$$

La aplicación de esta función para realizar la suma $3+4$ se puede representar como:

$$(\lambda x.\lambda y.SUMA)\ 3\ 4$$

o como:

$$(\lambda x.\lambda y.x\ y)\ 3\ 4$$

7.2. Luces virtuales en λ-cálculo

Puesto que en informática gráfica trabajamos con un dominio en $\mathbb{R}$, y en espacios compuestos de números reales ($\mathbb{R}^3$), necesitamos que el tipo de dato básico sea un real. Por el isomorfismo de Curry-Howard podemos definir tipos de datos, y puesto que deseamos tener un conjunto de datos reducido, definiremos un tipo de dato base: Real.

7.2.1. Tipos

A partir de este tendremos varios tipos de datos derivados del mismo:

1. Real
2. Booleano
3. Vector3D
4. Color
5. Matriz3x3
6. Matriz4x4
7. Propiedad

Estos tipos se pueden definir como siguen, a partir del tipo base:

Booleano (se puede representar como un subconjunto de los reales: VERDAD $> 0{,}0$ y FALSO $\leq 0{,}0$):

$$\begin{aligned} Real &\longrightarrow Booleano \\ \text{Booleano} &:= \lambda x.x \end{aligned}$$

Vector3D:

$$\begin{aligned} Real \longrightarrow Real \longrightarrow Real &\longrightarrow Vector3D \\ \text{Vector3D} &:= \lambda x.\lambda y.\lambda z.x\ y\ z \end{aligned}$$

Color:

$$\begin{aligned} Real \longrightarrow Real \longrightarrow Real \longrightarrow Real &\longrightarrow Color \\ \text{Color} &= \lambda r.\lambda g.\lambda b.\lambda a.r\ g\ b\ a \end{aligned}$$

Matriz3x3:

$$\begin{aligned} Real \longrightarrow Real \longrightarrow Real &\longrightarrow Real \\ \longrightarrow Real \longrightarrow Real &\longrightarrow Real \\ \longrightarrow Real \longrightarrow Real &\longrightarrow Matriz3x3 \\ \text{Matriz3x3} = \lambda c_{00}.\lambda c_{01}.\lambda c_{02}&. \\ \lambda c_{10}.\lambda c_{11}.\lambda c_{12}&. \\ \lambda c_{20}.\lambda c_{21}.\lambda c_{22}&. \\ c_{00}\ c_{01}\ c_{02}& \\ c_{10}\ c_{11}\ c_{12}& \\ c_{20}\ c_{21}\ c_{22}& \end{aligned}$$

Matriz4x4:

$$\begin{aligned} &Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow \\ &\longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow \\ &\longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow \\ &\longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Matriz4x4 \\ &\text{Matriz4x4} = \lambda c_{00}.\lambda c_{01}.\lambda c_{02}.\lambda c_{03}. \\ &\lambda c_{10}.\lambda c_{11}.\lambda c_{12}.\lambda c_{13}. \\ &\lambda c_{20}.\lambda c_{21}.\lambda c_{22}.\lambda c_{23}. \\ &\lambda c_{30}.\lambda c_{31}.\lambda c_{32}.\lambda c_{33}. \\ &c_{00}\ c_{01}\ c_{02}\ c_{03} \\ &c_{10}\ c_{11}\ c_{12}\ c_{13} \\ &c_{20}\ c_{21}\ c_{22}c_{23} \\ &c_{30}\ c_{31}\ c_{32}c_{33} \end{aligned}$$

La propiedad es algo asociado a un volumen, y trata de ser un valor de las características de cada *voxel* del volumen, el número de parámetros depende de lo que se desee hacer con él, es por ello que aquí definiremos unas propiedades mínimas (valor, color, grupo), aunque es irrelevante desde el punto de vista de la construcción de luces virtuales los valores que tenga añadidos la propiedad de cada *voxel*:

$$\begin{aligned} &Real \longrightarrow Color \longrightarrow Real \longrightarrow Propiedad \\ &\text{Propiedad} := \lambda v.\lambda c.\lambda g.v\ c\ g \end{aligned}$$

La propiedad puede devolver un valor nulo, para indicar que no existe una propiedad asociada a dicho valor. A esta propiedad la denominaremos ε.

Con esto tenemos los tipos más básicos, necesarios para definir un volumen:

$$\begin{aligned} &Vector3D \longrightarrow Propiedad \\ &\text{Volumen} := \lambda v.p \end{aligned}$$

Definición 6. *Un volumen f_{vo} es una función que devuelve la propiedad asociada de un punto cualquiera del espacio.*

Un volumen, por tanto, no es más que una función que toma un vector tridimensional y devuelve un valor de propiedad, concretamente el valor del *voxel* que contiene la posición dada.

Del volumen podemos obtener una definición de imagen como un subconjunto del mismo volumen, es decir, que una imagen no será más que un volumen, con la propiedad de tener todos los puntos en un mismo plano:

$$\begin{aligned} &Vector3D \longrightarrow Propiedad \\ &\text{Imagen} := \lambda v.p \end{aligned}$$

Definición 7. *Una imagen f_{im} es una función que devuelve la propiedad asociada de un punto, perteneciente al conjunto de puntos que forman un mismo plano en el espacio.*

En forma de conjuntos, el rango de la función imagen es un subconjunto del rango de la función volumen. De hecho, el rango de la función imagen y de la función volumen coincide al aplicarla en el subconjunto de puntos pertenecientes al plano que conforman la imagen.

7.2.2. Funciones volumétricas

A partir de estas funciones básicas podemos construir un conjunto al que llamaremos conjunto de funciones volumétricas:

Definición 8. *Una función volumétrica f_v es toda aquella función que toma como argumento una o varias funciones volumen, seguidos de cero o más valores reales o definidos a partir de reales, y que devuelve una función volumen.*

Definición 9. *Al conjunto de todas las funciones volumétricas se le denominará conjunto de funciones volumétricas F_v.*

Expresado según el lenguaje funcional tenemos que es toda aquella función de la forma:

$$f_v := Volumen \longrightarrow \ldots \longrightarrow Volumen \longrightarrow Real \longrightarrow \ldots \longrightarrow Real \longrightarrow Vector3D \longrightarrow \ldots \longrightarrow Vector3D \ldots \longrightarrow Volumen$$

Tenemos algunas funciones volumétricas especiales, que definiremos a continuación:

Definición 10. *Llamamos función volumétrica identidad i_v a la función volumétrica que toma una única función volumen como argumento y que devuelve la misma función volumen.*

Definición 11. *Llamamos función volumen nula 0_v a toda función volumétrica que devuelve una función volumen que únicamente devuelve propiedades vacías* ε.

Un ejemplo de función volumétrica podría ser la siguiente:

$$Volumen \longrightarrow Volumen \longrightarrow Real \longrightarrow Volumen$$
$$SUMA\_VOLUMEN = \lambda v_0.\lambda v_1.\lambda f.v_f$$

Esta función podría sumar los valores de una propiedad de v_0 y v_1 y añadirle el valor de f. El resultado se encontraría en v_f.

Las funciones de transferencia [28, 68] no son más que un subconjunto de las funciones volumétricas. Esto es así debido a que son funciones que cambian una propiedad, al igual que las funciones volumétricas. Sin embargo dentro de las funciones volumétricas también pueden haber transformaciones geométricas como rotación o translación, las cuales no se permiten en las funciones de transferencia.

7.2.3. Luces virtuales volumétricas

Una luz virtual volumétrica se puede definir a partir de un subconjunto de F_v. Mientras que una función volumétrica es una función que se aplica sobre uno o más volúmenes para obtener un volumen, y es irrelevante las operaciones internas que realice la función, en una luz virtual no sucede lo mismo.

Según la definición de luces virtuales de Martín [64], una luz podía situarse en el infinito, lo que es igual a no tener una posición concreta pero sí una dirección. O podía tener una posición concreta pero no tener una dirección.

A diferencia de las luces virtuales clásicas, nuestras luces virtuales volumétricas se definirán como una función con argumentos que indiquen posición y / o dirección, y algunos atributos adicionales:

Definición 12. *Una luz virtual f_l es toda aquella función volumétrica que tenga como mínimo un vector tridimensional como parámetro, significando una posición o una dirección. Y opcionalmente cero o más parámetros reales o definidos a partir de reales.*

Una función volumétrica compuesta por una o más luces virtuales es también una luz virtual.

Definición 13. *Al conjunto de todas las luces virtuales se le denomina conjunto de luces virtuales F_l.*

Expresado según el lenguaje funcional tenemos que es toda aquella función de la forma:

$$f_l := Volumen \longrightarrow \ldots \longrightarrow Volumen \longrightarrow Vector3D \longrightarrow \ldots \longrightarrow Real \longrightarrow Vector3D \longrightarrow \ldots \longrightarrow Vector3D \ldots \longrightarrow Volumen$$

Con esta definición nos aseguramos que a toda luz virtual se le pase una posición o dirección como mínimo en sus argumentos.

Un ejemplo de luz virtual es la siguiente:

$$Volumen \longrightarrow Volumen \longrightarrow Real \longrightarrow Volumen$$
$$LAMBERT = \lambda v_0.\lambda v_1.\lambda f_0.\lambda f_1.\lambda f_2.v_0$$

En la figura 7.1 tenemos un ejemplo de una luz virtual volumétrica f compuesta con varias funciones volumétrica, v_0, v_1 y v_2 son funciones volumen, f es una luz virtual volumétrica, el resto son funciones volumétricas. *Posición* son tres argumentos reales (x, y, z).

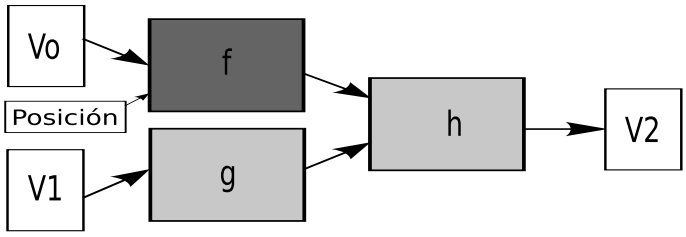

Figura 7.1: Esquema de cauce de funciones compuestas, equivalente a la función: $(\lambda s.h)((\lambda a.\lambda b.\lambda c.f)v_0\ x\ y\ z)((\lambda a.g)\ v_1)$. El resultado de aplicar esta función viene dado por v_2.

Podemos observar que si lo introducimos en una caja negra con los parámetros de entrada y salida, tenemos también una luz virtual volumétrica, lo cual es coherente con la definición dada de luz virtual volumétrica (ver figura 7.2).

7.3. Aplicación computacional

Una vez definidas las luces virtuales y las funciones volumétricas debemos construir un marco de trabajo para la aplicación computacional de estas funciones. Para ello definiremos la máquina virtual que ejecutará cada función hasta obtener el volumen final.

La máquina virtual a construir ejecutará volúmenes y variará en función del cauce de funciones que se construya, por tanto tiene que tener un mecanismo para añadir funciones y

Figura 7.2: Introduciendo la composición de las funciones en una caja negra (función volumétrica *e*) tenemos otra luz virtual volumétrica.

relacionarlas, de forma que forme un cauce de procesamiento para el volumen. Así mismo, hemos definido el volumen como una función, y por tanto también se deberá ejecutar el volumen final para visualizarlo en pantalla.

Además hay que tener en cuenta que una luz virtual volumétrica es una función volumétrica, también una función de transferencia es una función volumétrica. Sin embargo un volumen no es una función volumétrica. Por tanto distinguiremos entre funciones volumétricas y volúmenes, habrá que tener en cuenta que los valores intermedios siempre son volúmenes, por ello habrá que crear una operación para componer los volúmenes intermedios tal y como muestra la figura 7.3.

Tenemos en cuenta que la máquina virtual tiene una limitación y es que no puede devolver más que un volumen como resultado final, esto es coherente con el modelo propuesto, de forma que todo el cauce siempre puede ser una función volumétrica de más alto nivel. Aunque permitiremos que un volumen entre a varias funciones.

Para representar varios volúmenes a los que se les aplica el mismo cauce, se debe ejecutar el cauce para cada uno de ellos, como un cauce normal en el procesamiento de la tarjeta gráfica o el propio procesador (ver figura 7.4). En un único procesador, el primer volumen es procesado antes de que el siguiente volumen entre en el cauce de procesamiento.

También tenemos que tener en cuenta que podría llegar a darse una recursión en el sistema (por ejemplo, el volumen de salida entra de nuevo en la función de entrada), de forma que debe existir un método para evitar la recursión por conectar salidas como entradas en el cauce ejecutado previamente. La recursión la podemos simular mediante el operador Y del λ-cálculo [8], el cual repite una función un número de veces dado. Como este, existen varios λ-operadores que pueden aplicarse al procesamiento de las funciones para obtener bucles, condiciones, etc.

7.3.1. Cauce de procesamiento

La estructura más sencilla para almacenar las funciones y los volúmenes, los cuales supondremos por ahora un tipo de dato propio del sistema, es mediante un par de listas. Uno para los volúmenes y otro para las funciones volumétricas.

Los volúmenes estarán numerados mediante un identificador de tal forma que será sencillo referenciarlos, se pueden sustituir los identificadores por punteros a los volúmenes sin ningún problema, tal y como muestra la figura 7.5.

De esta forma los volúmenes pueden ser fácilmente referenciados mediante su índice en el vector, de la misma forma se almacenan las funciones volumétricas.

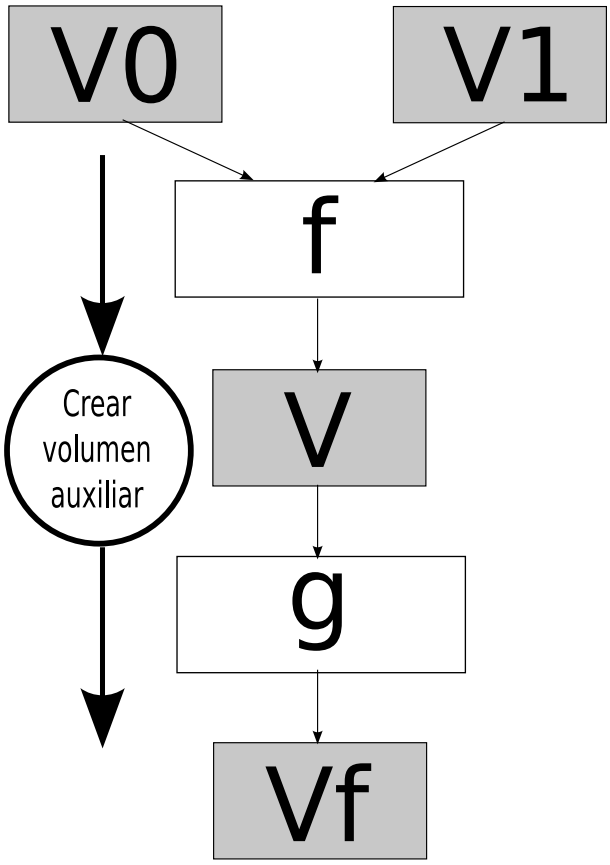

Figura 7.3: Esta máquina virtual ejecuta ambos volúmenes mediante la función f, generando un volumen intermedio auxiliar v que entra a la función g.

Los parámetros de las funciones volumétricas se representan mediante una lista de listas de índices variables en tamaño, tal y como muestra la figura 7.6. Esta lista tendrá el mismo orden que la lista de funciones volumétricas, de tal forma que si en la posición 3 de la lista de funciones volumétricas tenemos una con 3 parámetros, dichos parámetros estarán representados por la lista de la posición 3 de la lista de parámetros.

Para distinguir si a la entrada de la función va un volumen o una función volumétrica, se representan con los índices menores a 0 los volúmenes y con los índices mayores o iguales a 0 las funciones volumétricas. El algoritmo es muy sencillo:

```
01. Si indice < 0:
02.   tipo = volumen
03.   I = abs (indice) - 1
04. Si no:
05.   tipo = función_volumétrica
06.   I = indice
```

I representa la posición en el *array* de volúmenes o funciones volumétricas, según el tipo.

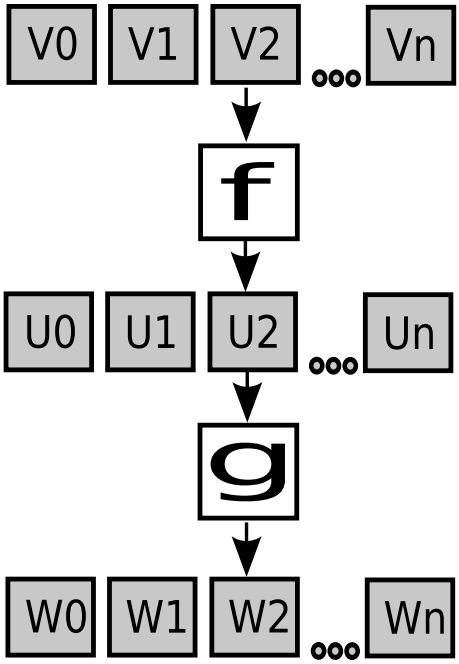

Figura 7.4: Para obtener varios volúmenes se tiene que aplicar el cauce a cada uno de ellos. En este ejemplo se pasan n volúmenes a la función f, cada uno de ellos es procesado y generan n volúmenes auxiliares, que se pasan a su vez a g, el cual genera los volúmenes de salida.

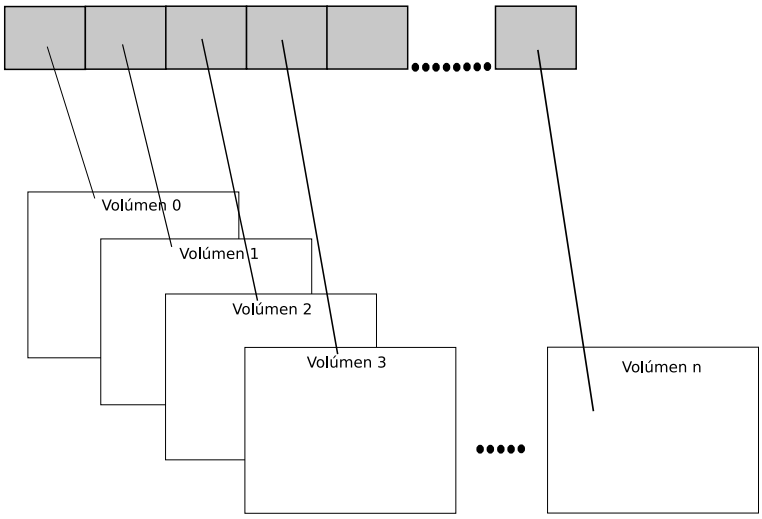

Figura 7.5: Lista de volúmenes.

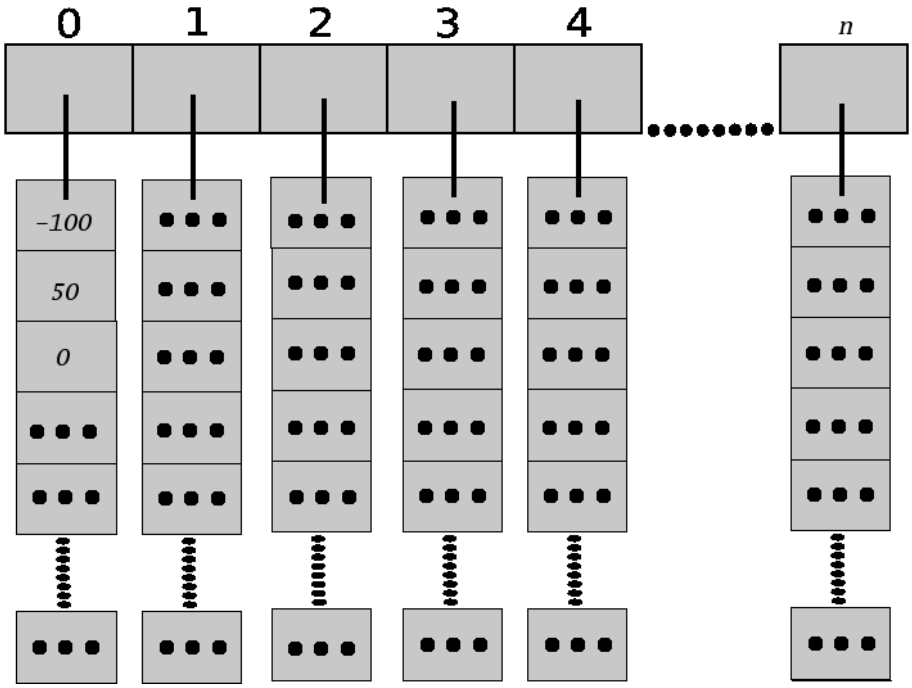

Figura 7.6: Lista de relaciones entre funciones volumétricas, los tres puntos negros indican que hay un número negativo o positivo en el interior de la celda, al igual que las casillas con un número.

7.3.2. Algoritmo de detección de recursión

Como se ha indicado anteriormente, es necesario que no exista recursión entre las entradas y las salidas de las funciones volumétricas. Esto no es trivial ya que podría darse el caso de que una función volumétrica hiciera uso de un volumen generado por ella misma en cualquier paso intermedio y no necesariamente en el paso justamente anterior, tal y como muestra la figura 7.7.

El algoritmo desarrollado para evitar este tipo de situaciones está basado en el algoritmo de eliminación de ciclos en grafos de Dijkstra [26], y es el que sigue:

```
01. la = nueva_lista ()
02. Para cada función volumétrica F:
03.    lp = F->lista_de_parametros()
04.    Para cada parámetro P:
05.       lc = nueva_lista ()
06.       lc->añade (P)
07.       Si lp[P] == F:
08.          Fin del algoritmo (Recursión)
09.       la->añade (lp[P])
10.       Mientras la <> vacío:
11.          vo = la->saca_último()
12.          si vo == función_volumétrica:
13.             si lc->repetidos() == verdad:
14.                Fin del algoritmo (Recursión)
```

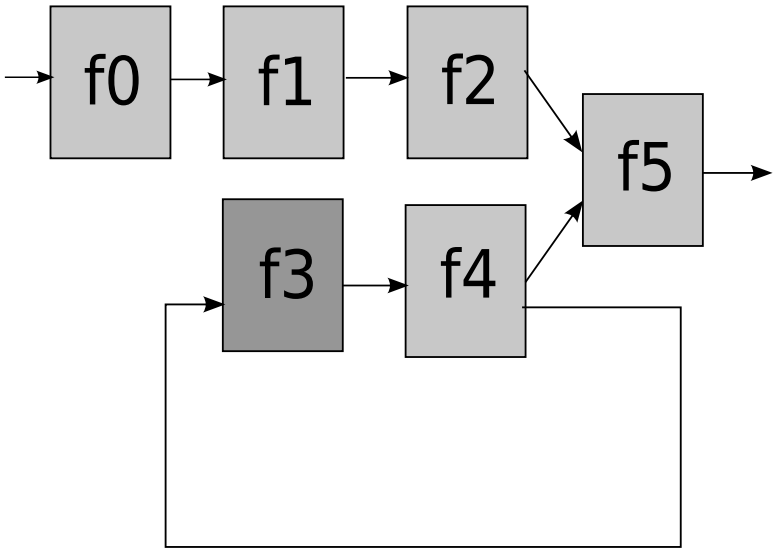

Figura 7.7: En esta función se produce una recursión no permitida en la función $f3$.

```
15.             lc->añade (vo)
16.             fv = función_virtual[vo]
17.             li = vo->lista_de_parametros()
18.             Para cada parámetro k de fv:
19.                Si li[k] == vo:
20.                   Hay recursión
21.                   Fin del algoritmo
22.                borrado = falso
23.                la->añade (li[k])
24.                Si (borrado == falso) y (li[k] == volumen):
25.                   borrado = verdad
26.                   lc->saca_último ()
27. Fin del algoritmo (No recursión)
```

El algoritmo comienza creando la lista *la* que es la lista donde se irán incluyendo los nodos por explorar (llamada *lista de abiertos*).

Para cada una de las funciones volumétricas (F) analizamos su lista de parámetros (a partir de la línea 4), en busca de los argumentos que necesita para ejecutarse. Puesto que cada uno de sus argumentos (P) serán o una función o un volumen, debemos detectar cuando es un volumen (significa que no hay un ciclo), y cuando es función volumétrica (paso 12 del algoritmo). Independientemente de que sea o no un volumen hay que introducirlo en la lista de visitados *lc*.

Los bucles más sencillos se detectan rápidamente si encontramos que el parámetro P se encuentra en la lista de parámetros de la función analizada, entonces tenemos un bucle (es la línea 7).

El resto de bucles se detecta mirando si en la lista de visitados existe algún repetido (es la línea 13, en la cual se examina si el vector *lc* tiene elementos repetidos).

Saltamos a la siguiente función volumétrica que se le pasa por argumento y examinamos sus argumentos (línea 18), a partir de aquí se añade a la lista de abiertos *la* el elemento y se

sacan los elementos que ya hayamos explorado totalmente de la lista de visitados lc.

Si después de ejecutarse todo el algoritmo no nos quedan nodos en abiertos ni funciones volumétricas que explorar entonces es que no existe recursión.

7.3.3. Algoritmo de ejecución

La ejecución del cauce puede realizarse de una forma muy sencilla mediante un algoritmo recursivo a partir de la última celda o función volumétrica.

Detectar esta última celda es trivial ya que no aparecerá como argumento del resto de las funciones, ya que ninguna otra función volumétrica la pasará como su argumento.

De forma práctica, consiste en visitar los parámetros de todas las funciones y analizar cual es la función volumétrica no referenciada.

El algoritmo de ejecución de una función volumétrica es el siguiente:

```
01. Sea lv la lista de volúmenes
02. Sea lf la lista de funciones volumétricas
03. Sea F la funcion volumétrica a ejecutar
04. lp = F->lista_de_parametros()
05. Vaux = nueva_lista ()
06. Para cada parámetro P de lp:
07.    Si P == función_volumétrica:
08.       Vaux->añade (P->ejecuta())
09.    si no:
10.       Vaux->añade (P)
11. V = F->resuelve (Vaux, parámetros_reales)
12. Devuelve V
```

El algoritmo lo único que hace es comprobar si cada uno de los parámetros de la función son volúmenes o no (los parámetros reales se ignoran). En caso de que no sean volúmenes hay que ejecutar previamente una función volumétrica y obtener su volumen para ejecutarla.

Una vez que todos los parámetros son volúmenes se resuelve la función actual ejecutándola realmente con los parámetros reales y los volúmenes.

Sin embargo el problema que tenemos es que para cada función que es llamada varias veces con los mismos parámetros hay que calcular todos los volúmenes de nuevo y ejecutar la misma función cuando el resultado ya lo habíamos calculado previamente.

Para evitarlo basta con añadir dos líneas al algoritmo anterior:

```
11.  V = F->resuelve (Vaux, parámetros_reales)
11a. lv->añade (V)
11b. fv->sustituye (F, V)
11c. lv->limpiaVolúmenes()
12.  Devuelve V
```

La línea 11a lo que hace es añadir a la lista de volúmenes un nuevo volumen resultante de la ejecución de F. La siguiente línea sustituye en los parámetros toda función F por el volumen V, de esta forma ya no es necesario recalcular la función F.

La última línea añadida (11c) es necesaria para liberar de memoria los volúmenes que no van a ser utilizados en todo el cauce. Esta función de limpiar volúmenes será detallada en la siguiente sección.

Si queremos evitar una búsqueda en la lista y el reemplazo (con la función $sustituye()$), también podemos crear un nuevo vector de volúmenes de longitud la misma que las funciones volumétricas. Para cada función volumétrica ejecutada se introduce en esta lista, en la posición de la función volumétrica, el volumen resultante de ejecutar la función. De forma

que en el paso 5 del algoritmo se añadiría una búsqueda sobre este vector y si ya se encuentra un volumen se sustituiría *P* por este mismo volumen.

El algoritmo de ejecución final quedaría por tanto, como:

```
01. Sea lv la lista de volúmenes
02. Sea lf la lista de funciones volumétricas
03. Sea F la funcion volumétrica a ejecutar
04. Sea lu la lista auxiliar de volúmenes
05. lp = F->lista_de_parametros()
06. Vaux = nueva_lista ()
07. Para cada parámetro P de lp:
08.    Si P == función_volumétrica:
09.       Si lu[P] <> vacío:
10.          Vaux->añade (lu[P])
11.       si no:
12.          Vaux->añade (P->ejecuta())
13.    si no:
14.       Vaux->añade (P)
15. V = F->resuelve (Vaux, parámetros_reales)
16. lv->limpiaVolúmenes()
17. Devuelve V
```

Vamos a examinar con detalle el algoritmo de limpieza de volúmenes.

7.3.4. Algoritmo de limpieza de volúmenes

El algoritmo de limpieza de volúmenes es una parte imprescindible del algoritmo de ejecución ya que aumenta la eficiencia en memoria, lo cual suele ser crítico en los sistemas que trabajan con volúmenes por la cantidad de espacio requerido debido a la alta cantidad de datos de los modelos.

El algoritmo que presentamos trata de eliminar todos los volúmenes incluidos los volúmenes originales de entrada por defecto. El algoritmo es el siguiente:

```
01. Sea lv la lista de volúmenes
02. Sea lf la lista de funciones volumétricas
03. Sea lu la lista de volúmenes auxiliares
04. Sea k el índice de la función que se está ejecutando
05. lm0 = nueva_lista ()
06. lm1 = nueva_lista ()
07. Para cada v en lu:
08.    lm0->añade (v)
09. Para cada v en lv:
10.    lm1->añade (v)
11. Para cada v en lu:
12.    Si (v == vacío) y (v->índice() <> k):
13.       lp = lf[v->índice()]->parámetros()
14.       Para cada p en lp:
15.          Si p == función_volumétrica():
16.             Si lu[p->índice()] <> vacío:
17.                lm0->cambia (p, vacío)
18.          si no:
19.             Si p == vacío:
20.                lm1->cambia (p, vacío)
21. Para cada v en lm0:
22.   v->libera()
23.   lu->actualiza()
24. Para cada v en lm1:
```

```
25.   v->libera()
26.   lu->actualiza()
```

Creamos en este algoritmo dos listas, que van a usarse para marcar los volúmenes originales ($lm1$) y los volúmenes auxiliares ($lm0$) que pueden ser liberados. Al principio (pasos 6 - 9 del algoritmo), los vectores se inician para que contengan todos los volúmenes calculados.

En un paso posterior (a partir de 10) se examina la lista de parámetros en cada función volumétrica. Si hace uso de algún parámetro que aún no está calculado, el volumen asociado se pone a *vacío* en la lista de volúmenes auxiliares ($lm0$) si es una función volumétrica (pasos 15 y 16) o en la lista de volúmenes originales ($lm1$) (pasos 18 y 19) en caso de que se trate de un volumen.

Después todos aquellos volúmenes que no se han marcado como vacío son sobrantes (ya que no se necesitan en la llamada de la función actual).

7.4. Descripción del modelo de luces virtuales volumétricas

Puesto que lo que deseamos es tener un modelo que unifique todas las funciones de los capítulos anteriores, pasamos a describir su definición y funcionamiento.

7.4.1. Luces virtuales concretas

Según los capítulos anteriores podemos tener las siguientes funciones volumétricas:

- *CLASIFICA*: Función que toma un volumen origen y modifica la propiedad de grupo en función de tres parámetros reales. Los dos primeros son el intervalo, y el tercero indica el valor de grupo a asignar.
  - $Volumen \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Volumen$
- H: Función que toma un volumen y lo transforma a HSV, quedándose con el valor H intacto y modificando el S y V a 1. Necesita un parámetro que le indique el grupo de *vóxeles* a modificar.
  - $Volumen \longrightarrow Real \longrightarrow Volumen$
- $TRAZA$: Función que toma una imagen y la transforma en tinta, como parámetros tiene los siguientes: parámetro σ (para la convolución Gaussiana), dos umbrales tl y th (para el algoritmo de Canny), una distancia (para la detección de trazos de la ventana) y una longitud máxima de los trazos además de la transparencia.
  - $Imagen \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Imagen$

Como luces virtuales tenemos:

- P: Obtiene un volumen y lo transforma en una imagen, según una posición de la cámara (tres reales) y su dirección (otros tres reales), realizando una proyección en perspectiva de la misma.
  - $Volumen \longrightarrow Vector3D \longrightarrow Vector3D \longrightarrow Imagen$

- *S*: Obtiene un volumen y transforma su color en el espacio HSV, modifica la propiedad de saturación (*S*) y deja las otras dos intactas. Con $S = \vec{L} \cdot \vec{N}$, donde *L* será el primer parámetro (tres reales indicando una dirección) y *N* la normal del *voxel*.
  - $Volumen \longrightarrow Vector3D \longrightarrow Volumen$
- S_j: Obtiene el color del volumen en cada *voxel* y lo transforma al espacio de color HSV, modifica la propiedad de saturación (*S*) y deja las otras dos intactas. La ecuación que lo define es la ecuación 5.4.
  - $Volumen \longrightarrow Real \longrightarrow Volumen$
- *V*: Obtiene un volumen y lo transforma en HSV, modifica la propiedad de intensidad (*V*) y deja las otras dos intactas. Con $V = \vec{L} \cdot \vec{N}$, donde *L* será el primer parámetro (tres reales indicando una dirección) y *N* la normal del *voxel*.
  - $Volumen \longrightarrow Vector3D \longrightarrow Volumen$
- V_j: Obtiene un volumen y lo transforma en HSV, modifica la propiedad de intensidad (*V*) y deja las otras dos intactas. La ecuación que lo define es la ecuación 5.5. Los parámetros corresponden a k_a, k_d, k_s y k_f.
  - $Volumen \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Real \longrightarrow Volumen$
- *SOMBREAR*: Obtiene un volumen y lo transforma en HSV, modifica la propiedad de intensidad (*V*) y deja las otras dos intactas. El algoritmo que lo define se encuentra en el capítulo de sombreado (sección 5.3.5). El parámetro *Real* corresponde al valor *t* del algoritmo.
  - $Volumen \longrightarrow Real \longrightarrow Volumen$
- *RESTA*: Obtiene el resultado de borrar un volumen a otro (el primer parámetro queda sustraído por el segundo).
  - $Volumen \longrightarrow Volumen \longrightarrow Volumen$

7.4.2. Resultados

Con estos algoritmos podemos introducir las funciones volumétricas combinadas con las luces virtuales en el cauce de procesamiento para obtener resultados de forma sencilla para el programador y para el usuario, ya que estas funciones pueden mostrarse como una serie de componentes o cajas que se enlazan entre sí, y que tienen asociados unos parámetros.

Además si implementamos cada uno de estos elementos por *hardware*, la arquitectura queda implementada por el mismo cauce mostrado mediante software.

Estas funciones pueden combinarse para formar los algoritmos mostrados en los capítulos anteriores, como se muestra en la figura 7.8.

El cauce asociado a cada imagen de la figura anterior se muestra en la figura 7.9. En las cajas gris oscuro y letras blancas tenemos los parámetros reales o derivados de reales. En las cajas de fondo blanco tenemos los volúmenes y en las cajas grises con letras negras las funciones volumétricas y luces virtuales.

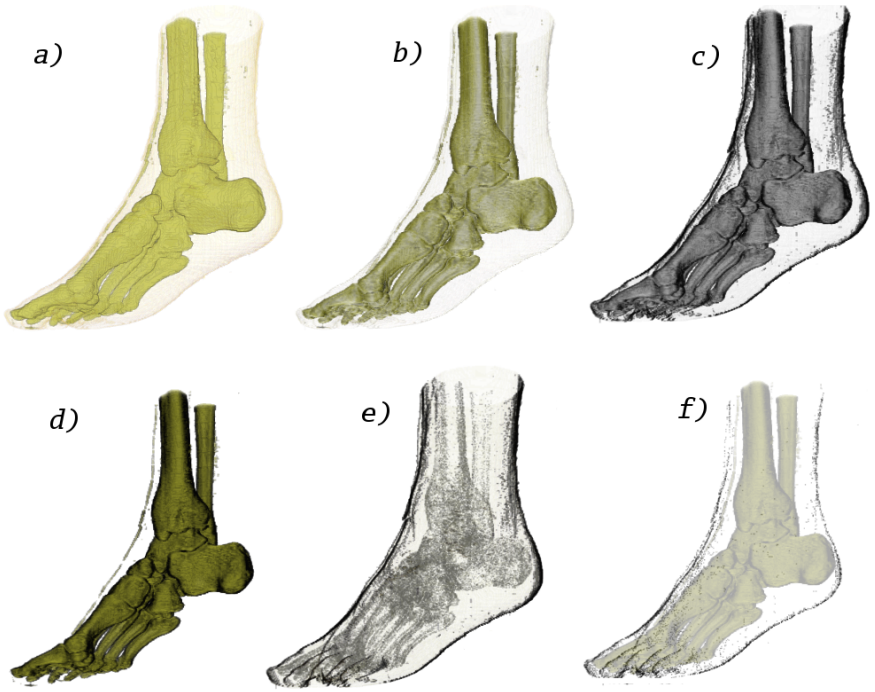

Figura 7.8: Resultados después de aplicar diferentes luces virtuales volumétricas.

Podemos agrupar diferentes funciones volumétricas para hacer funciones volumétricas más complejas, como el caso de la función **CLASIFICA_PIE**, que está formada a partir de distintas funciones de clasificación. Los parámetros asociados a Vj se han obviado para clarificar el esquema, ya que son los mismos que en los indicados en la tabla de la figura 5.9 y han sido explicados en capítulos anteriores. Los parámetros reales en la función de clasificación se han simplificado al grupo (ya que los colores son visibles en la imagen, los valores umbrales son dependientes de los datos de entrada y las luces virtuales están situadas en la misma posición que la cámara).

Podemos observar como los dos últimos cauces generan imágenes mediante la función **P** y quedan mezclados en una imagen final. Podemos por tanto trabajar con volúmenes e imágenes de forma indistinta, aunque por supuesto las funciones que tomen un volumen y se les de una imagen obtendrá un resultado que no será el apropiado, ya que no existirá información tridimensional.

En la figura 7.10 vemos otro ejemplo de mezclar volúmenes con imágenes, mezclando los algoritmos de la sección 2.2 con los de ilustración de volúmenes del capítulo 5.

El cauce utilizado para ambas imágenes se muestra en la figura 7.11, igual que antes hemos simplificado los parámetros del cauce para que se pueda apreciar de forma sencilla.

Hay que tener en cuenta que no todas las funciones son conmutativas, es decir, que no siempre se da que $f(g(x)) = g(f(x))$, un ejemplo lo tenemos en la figura 7.12.

Si observamos el cauce (figura 7.13) podemos apreciar como en la imagen central (b) no se aprecia correctamente el corte producido en la parte parietal del cráneo, no detectando frontera en el corte, es por tanto determinante que se realice la resta del volumen antes de

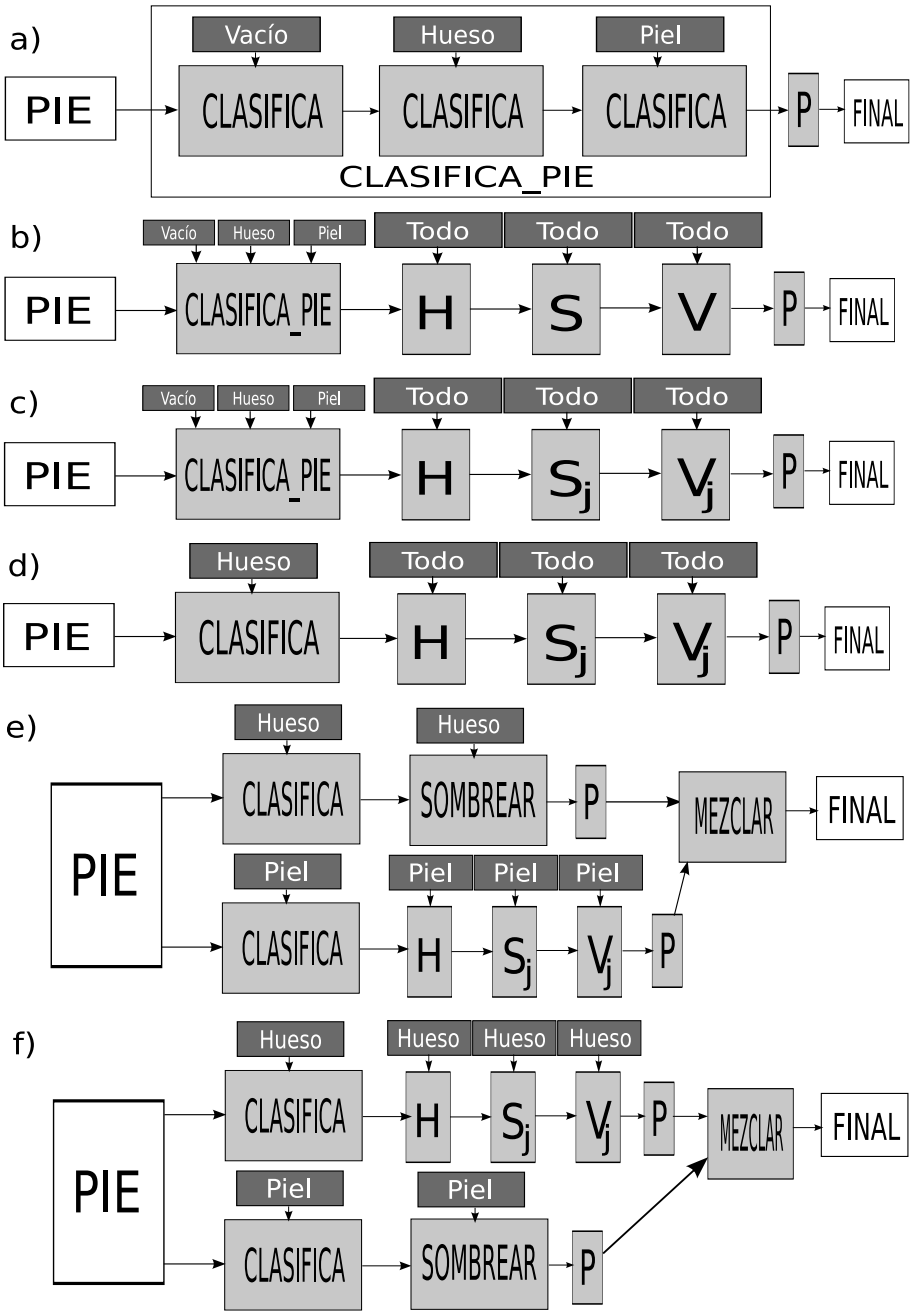

Figura 7.9: Cauces asociados a la figura 7.8.

detectar las fronteras. Si vemos el volumen de la izquierda (a) y de la derecha (b) podemos apreciar que también el sombreado mediante puntos queda alterado en el orden, ya que si

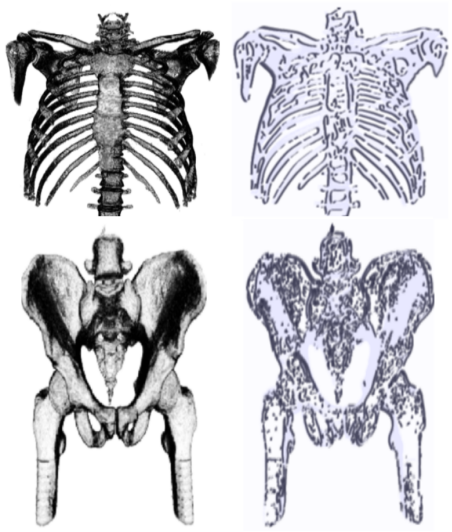

Figura 7.10: Podemos mezclar algoritmos para imágenes con volúmenes 7.10.

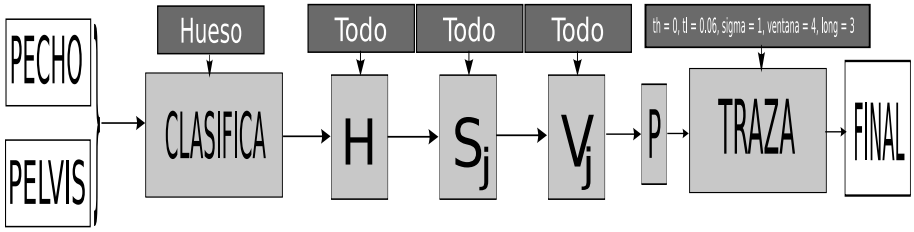

Figura 7.11: El mismo cauce mezcla imágenes con volúmenes.

lo aplicamos antes de detectar las siluetas también se realizará la detección de las mismas después de sombrear, con lo que no nos aparecerán puntos en la parte central del modelo (por ejemplo, el cuello). Aunque las diferencias puedan parecer mínimas, son sumamente importantes dependiendo de la fuente de luz virtual volumétrica que empleemos.

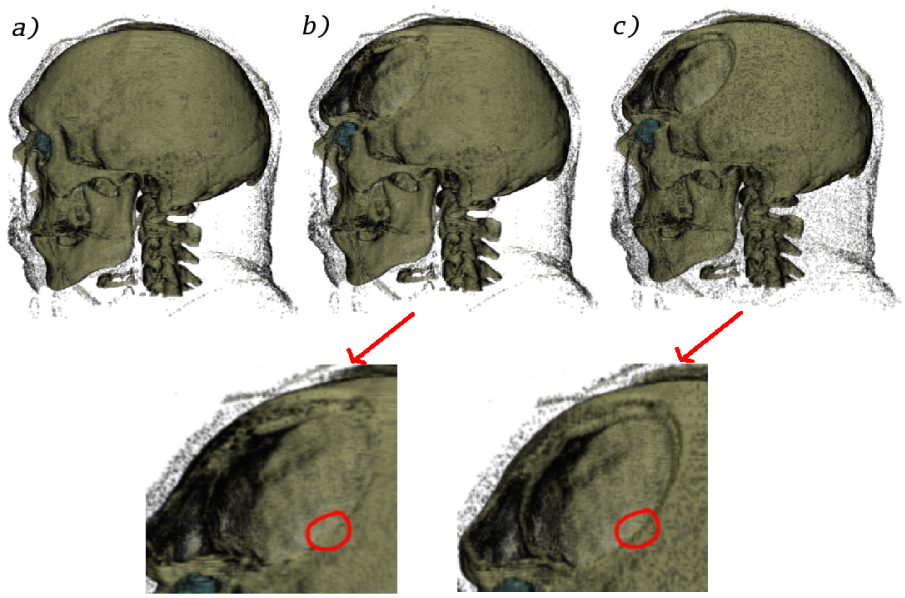

Figura 7.12: A la izquierda imagen sin usar la luz de resta, al centro, la luz de resta se aplica en un orden incorrecto, a la derecha orden correcto de aplicación.

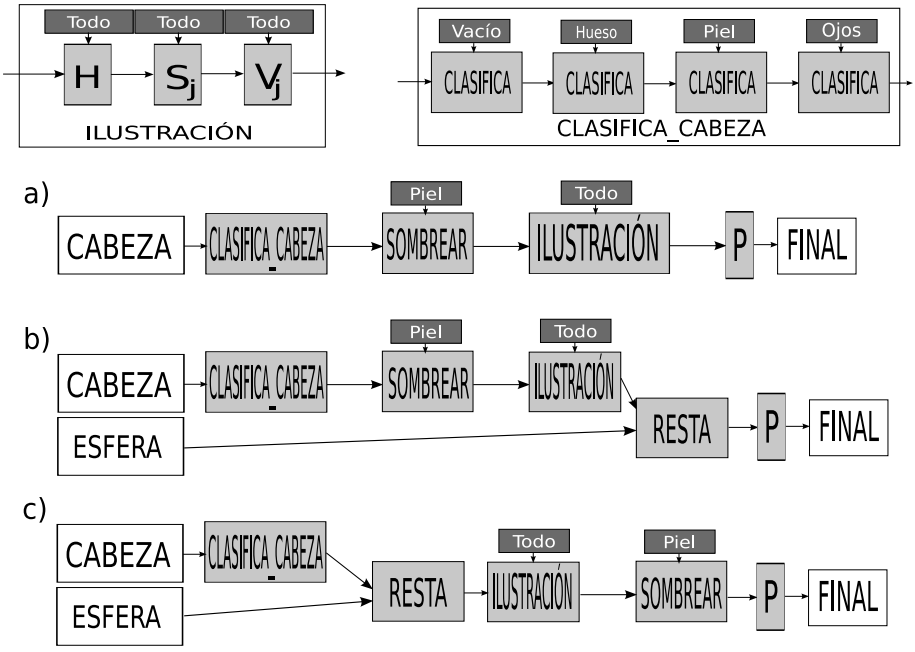

Figura 7.13: Cauces aplicados sobre cada imagen de la figura 7.12, el orden de los parámetros es de arriba (primero) a abajo (último).

Conclusiones y Trabajos futuros

Hemos desarrollado un trabajo que expande las luces virtuales como operadores (funciones) que pueden combinarse para crear nuevas luces virtuales y nuevas imágenes expresivas. Como hemos ido comprobando a lo largo de los sucesivos capítulos, todo el trabajo se ha desarrollado con tecnología puntual, en concreto a base de téxeles o puntos en un espacio, de forma que no es necesario el trabajar con polígonos y además podemos tener modelos que en su interior son sólidos, como es el caso de los modelos volumétricos.

Conclusiones y Trabajos futuros

En este trabajo hemos mostrado que la visualización expresiva tiene tres campos amplios en los cuales puede desarrollarse:

- Para crear elementos expresivos y de animación (capítulo 6).
- Para mejorar la calidad de una imagen desde el punto de vista perceptual humano y poder diagnosticar a partir de ella (capítulos 3 y 4).
- Para generar ilustraciones científicas que ayuden a la adquisición de conocimiento en áreas donde los modelos reales son complicados al sistema perceptual humano. También en campos interdisciplinares donde los que trabajan con los modelos no tienen por qué ser expertos en la materia que están analizando (capítulo 5).

Analizaremos cada uno de los capítulos en particular.

En el capítulo de introducción hemos repasado los trabajos previos generales en la informática gráfica, y en particular en la visualización de volúmenes.

En el capítulo 2 hemos repasado los conceptos básicos de análisis de imágenes junto con la forma de trabajo de la percepción visual humana para introducirnos en los pasos básicos de análisis de los volúmenes en el proceso de filtrado y de cómo obtener información adicional acerca de la propia imagen. Además hemos repasado los conceptos de la percepción visual humana que nos ha servido para poder decidir si una imagen expresiva es válida o no para el sistema perceptual humano, y por tanto, completa para nuestro sistema perceptual. También hemos repasado los conceptos más importantes de la visualización expresiva y las técnicas utilizadas por los artistas para dibujar elementos científicos en ilustración.

En el capítulo 3 hemos presentado un método de iluminación que permite mejorar la visualización de los modelos volumétricos en la visualización directa del mismo. Está basado en el modelo de color HSV mediante el cual se usa el tono para definir el material, la saturación para el sombreado y el valor para resaltar las fronteras. Así mismo, la asignación de tonos a materiales se realiza eligiendo aquellos colores que facilitan la percepción del volumen por parte del usuario.

Si bien existe una parte subjetiva de elección de colores, esta subjetividad pasa a ser un parámetro (prácticamente el único), en un algoritmo completo de visualizado en donde se da un cambio en el tipo de iluminación de los objetos. El proceso no cambia el rendimiento del algoritmo mejorando sustancialmente la apreciación de cambios en las tonalidades de un modelo formado por datos transparentes difícilmente discernibles por el ser humano. Los resultados han sido publicados en [6].

En el capítulo 4 hemos presentado un nuevo método de detección de siluetas específicamente diseñado para volúmenes que contienen zonas de interés en el interior de los mismos. Se basa en un proceso previo de clasificación de la información volumétrica y en un sistema de iluminación expresiva.

Hay que tener en cuenta que las siluetas en un modelo volumétrico no pueden aparecer de la misma forma que en un modelo sólido, debido a que las partes de interés en un volumen pueden encontrarse en cualquier zona del interior del objeto mientras que en los objetos sólidos tan solo pueden aparecer en la superficie del mismo. De esta forma las siluetas en un modelo volumétrico deben estar tanto dentro del objeto como fuera del mismo.

Nuestro método no requiere de la definición previa de un umbral y por tanto puede obtener simultáneamente las siluetas de objetos diferentes en cuanto al material que lo constituyen. Tan solo hay que definir en el proceso de clasificación las tonalidades que se le asignan a las diferentes zonas de interés. Estas tonalidades se eligen teniendo en cuenta la característica del ojo humano de distinguir mejor unos tonos que otros. Parte de los resultados han sido publicados en [5, 7].

En el capítulo 5 hemos presentado un nuevo método de iluminación que permite realizar ilustraciones a partir de datos volumétricos en un cauce de visualización directa de volúmenes. También se han definido parámetros que permiten al usuario modificar la iluminación y los materiales de una forma intuitiva para conseguir imágenes parecidas a las que los ilustradores son capaces de realizar. Nuestra aproximación utiliza un modelo de color cercano al usuario para definir los materiales de una forma rápida e intuitiva, permitiendo un control a posteriori del tipo de ilustración que se desea realizar y evitando tediosos métodos de ensayo y error por parte del usuario. Parte de los resultados han sido publicados en [4].

En el capítulo 6 hemos mostrado como a partir de una serie de imágenes, ya sean estáticas o como elementos en un vídeo (y por tanto, consecutivas en el tiempo) podemos obtener elementos expresivos que pueden llegar a ser considerados ilustraciones parecidas a las realizadas por el ser humano a partir de un nuevo método para ilustración a partir de imágenes en escala de grises. Se han comentado las distintas técnicas que se pueden aplicar tras un proceso sencillo de análisis de imágenes y de abstracción de datos, y se ha aplicado el modelo modificándolo para imágenes continuas en el tiempo. Parte de los resultados han sido publicados en [3].

En el capítulo 7 hemos unificado todos los trabajos expuestos en los diferentes capítulos, creando un nuevo método de trabajo funcional que permite realizar imágenes expresivas de distinta índole mediante la definición sencilla de un cauce de volúmenes. Hemos definido un nuevo concepto: **luz virtual volumétrica** como extensión de las luces virtuales clásicas, permitiendo trabajar de forma muy sencilla con volúmenes tanto para el usuario final como para los programadores intermedios, creando un nuevo paradigma de computación de volúmenes.

Como trabajos futuros tenemos todos un abanico de posibilidades, pero principalmente queremos centrarnos en los siguientes conceptos:

- Aplicar el cauce funcional a una serie de máquinas paralelas para llegar a obtener arquitecturas nuevas y con un mayor rendimiento en tiempos hasta llegar a tiempo interactivo o tiempo real con alta cantidad de datos.
- Mejorar las ilustraciones, no solo para impresión en papel sino también para otro tipo de elementos más modernos, como realidad virtual o libros con capas semitranslúcidas.
- Aplicación de segmentación más compleja en los modelos para obtener elementos de interés con mucha nitidez.

- Modificar la reducción de tonos de los modelos en el dominio de la frecuencia, esto permitirá una mejor calidad del modelo final (con menor ruido).
- Mejorar el algoritmo de detección de siluetas para obtenerlas más rápidamente y de formas más delgadas.
- Mejorar la expresividad de las siluetas, creando caminos alterables de forma similar a como se realiza en las imágenes.
- Utilizar instrucciones específicas (SSE2, 3DNow, etc.) en procesadores específicos de 64 bits pensados para aplicaciones gráficas con valores reales [39].
- Utilizar la aceleradora gráfica para mejorar los tiempos en los algoritmos, paralelizándolos con los procesadores.
- Modificar los algoritmos de iluminación a otros sistemas de color distintos al HSV o al RGB.
- Mejorar los algoritmos de ejecución de volúmenes y limpieza del mismo.

Por último baste decir que este trabajo no es el fin sino el comienzo de nuestra investigación, ya que las posibilidades que se abren ante este nuevo campo es realmente espectacular, ya no solo por las posibilidades de aplicación a distintas áreas ajenas a la informática gráfica (o inclusive a la informática en general), sino también por la posibilidad de la creación de toda una nueva subárea de investigación para obtener imágenes expresivas en un menor tiempo y con una mayor calidad sin coste de complejidad para usuarios o programadores.

Bibliografía

[1] Nvidia opengl extensions especifications. http://www.nvidia.com/page/home, 2004.

[2] Opengl specification, version 2.0. http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf, 2004.

[3] ARROYO, G., AND MARTÍN, D. Artistic representation of ink and paper pictures from gray-scale photographs. In *EUROGRAPHICS'03. Poster Presentations* (Granada, Spain, 2003), pp. 105–108.

[4] ARROYO, G., MARTÍN, D., AND VELASCO, F. Obtención de ilustraciones mediante el uso de luces virtuales aplicadas a la visualización directa de volúmenes. In *CEDI'05* (Granada, Spain, 2005), pp. 269–272.

[5] ARROYO, G., VELASCO, F., AND MARTÍN, D. Detección de siluetas en modelos volumétricos basado en un sistema de iluminación no fotorrealista. In *CEIG'04* (Sevilla, Spain, 2004), pp. 299–312.

[6] ARROYO, G., VELASCO, F., AND MARTÍN, D. Iluminación no-fotorrealista en visualización directa de modelos volumétricos basado en el modelo de color hsv. In *CEIG'04* (Sevilla, Spain, 2004), pp. 285–298.

[7] ARROYO, G., VELASCO, F., AND MARTÍN, D. Silhouette detection in volumetric models based on a non-photorealistic illumination system. *Computer and Graphics 29*, 2 (2005), 209–216.

[8] BARENDREGT, H. P. *The Lambda Calculus. Its Syntax and Semantics*. Elsevier Science Publishers B. V., 1981.

[9] BELL, J., AND MACHOVER, M. *A Course in Mathematical Logic*. Elsevier Sciencie Publisiers B. V., 1993.

[10] BOADA, I. *Towards Multiresolution Integrated Surface Volume Data Representations*. PhD thesis, Technical University of Catalonia, Spain, 2001.

[11] BOURGUIGNON, D., CANI, M.-P., AND DRETTAKIS, G. Drawing for illustration and annotation in 3d. In *Eurographics 2001* (Manchester, UK, 2001), pp. 755–762.

[12] BRODLIE, K., AND WOOD, J. Recent advances in volume visualization. *Computer Graphics forum 20*, 2 (2001), 125–148.

[13] BRUNET, P., NAVAZO, I., AND VINACUA, A. Octree detection of closed compartments. In *ACM: International Journal of Computatinal Geometry and Applications* (Texas, USA, 1991), pp. 263–280.

[14] BUCHANAN, J. W., AND SOUSA, M. C. The edge buffer: a data structure for easy silhouette rendering. In *Proceedings of NPAR* (Annecy, France, June 2000), pp. 39–42.

[15] CANNY, J. A. A computational approach to edge detection. *EEE Trans. Pattern Analysis and Machine Intelligence 8*, 6 (1986), 679–698.

[16] CANO, P. *Representación Jerárquica de Sólidos Poliédricos*. PhD thesis, Dpto. Lenguajes y Sistemas Informáticos, Universidad de Granada, Spain, 2004.

[17] CANO, P., VELASCO, F., AND TORRES, J. C. Modelado 2d mediane jerarquía de quadtrees en distintos sistemas de coordenadas. In *II Jornadas de informática* (Almuñecar, Spain, 1996), pp. 153–162.

[18] CHEN, H., LIU, Z., ROSE, C., XU, Y., SHUM, H.-Y., AND SALESIN, D. Non-photorealistic animation and rendering. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering* (Annecy, France, 2004), pp. 95–153.

[19] CHRISTIANSEN, H. N., AND SEDERBERG, T. W. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graphics (ACM Siggraph Proceedings) 12*, 3 (1978), 187–192.

[20] COOK, L., BATNITZKY, S., AND LEE, K. R. A three dimensional display system for diagnostic imaging applications. *IEEE Computer Graphics and Applications 20*, 3 (1983), 13–19.

[21] COOTES, T. F., TAYLOR, C. J., COOPER, D. H., AND GRAHAM, J. Active shape models - thir training and application. *Computer Vision and Image Understanding: CVIU 61*, 1 (1995), 38–59.

[22] CSÉBFALVIY, B., MROZ, L., HAUSERZ, H., KÖNIG, A., AND GRÖLLERX, E. Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum 20*, 3 (2001), 211–218.

[23] DANSKIN, J., AND HANRAHAN, P. Fast algorithms for volume ray tracing. In *Proceedings of the 1992 workshop on Volume visualization* (Boston, USA, 1992), pp. 91 – 98.

[24] DECARLO, D., AND SANTELLA, A. Stylization and abstraction of photographs. In *SIGGRAPH* (San Antonio, Texas, 2002), pp. 769–776.

[25] DIEPSTRATEN, J., WEISKOPF, D., AND ERTL, T. Transparency in interactive technical illustrations. *Computer Graphics Forum 21*, 3 (2002), 523–532.

[26] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 47*, 2 (1959), 269–271.

[27] DODSON, C. T. J., AND HERDMAN, P. Mechanical properties of paper. *Handbook of Paper Science 2*, 3 (1980), 71–126.

[28] EBERT, D., AND RHEINGANS, P. Volume illustration: Non-photorealistic rendering of volume models. In *Proceedings Visualization 2000* (San Francisco, USA, 2000), pp. 195–202.

[29] FOLEY, J., DAM, V., FEINER, S., AND HUGHES, J. *Introduction to Computer Graphics*. Addison-Wesley, 1996.

[30] FRISBY, J. *Exploratory Data Analysis*. Oxford University Press, 1980.

[31] FUCHS, H., KEDEM, Z., AND USELTON, S. Optimal surface reconstruction from planar contours. *Computer Graphics (ACM Siggraph Proceedings) 20*, 10 (1977), 693–702.

[32] GELDER, A. V., AND KIM, K. Direct volume rendering with shading via three-dimensional textures. In *Proceedings of the 1996 symposium on Volume visualization* (San Francisco, USA, 1996), pp. 23–ff.

[33] GOLUB, G. H., AND LOAN, C. F. V. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins, 1996.

[34] GOOCH, A., GOOCH, B., P.SHIRLEY, AND COHEN, E. A non-photorealistic lighting model for automatic technical illustration. *Computer Graphics 32*, 5 (1998), 447–452.

[35] GOOCH, B., AND GOOCH, A. *Non-photorealistic Rendering*. A. K. Peters, July 2001.

[36] GOOCH, B., SLOAN, P., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. Interactive technical illustration. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (Atlanta, USA, 1999), pp. 31–38.

[37] GOODMAN, J. W. *Introduction to Fourier Optics*. McGraw-Hill, 1968.

[38] GRASSMAN, H. G. Teory of compound colours. *Philosophic Magazine 4*, 7 (1954), 254–264.

[39] HENNESSY, J. L., AND PATTERSON, D. A. *Computer Architecture, Third Edition*. Morgan Kaufmann Publishers, 2003.

[40] HERTZMANN, A. Paint by relaxation. In *CGI '01: Computer Graphics International 2001* (Washington, USA, 2001), pp. 47–54.

[41] ISENBERG, T., HALPER, N., AND STROTHOTTE, T. Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. *Computer Graphics* (2002), 249–258.

[42] JAIN, A. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[43] KAJIYA, J., AND HERZEN, B. V. Ray-tracing volume densities. In *SIGGRAPH'84 Proceedings* (July 1984), H. Cristiansen, Ed., vol. 18, pp. 165–174.

[44] KALVIN, A. A survey of algorithms for constructing surfaces from 3d volume data. *IBM Research Report RC 17600 13*, 6 (1992), 453–462.

[45] KANUS, U., WETEKAM, G., AND HIRCHE, J. Voxelcache: A cache-based memory architecture for volume graphics. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (San Diego, USA, 2003), W. M. In M. Doggett, W. Heidrich and A. S. Editors, Eds., pp. 76–83.

[46] KAUFMAN, A. *Trends in volume visualization and volume graphics*. Academic Press Ltd, 1994.

[47] KAUFMAN, A., DACHILLE, F., CHEN, B., BITTER, I., KREEGER, K., ZHANG, N., AND TANG, Q. Real-time volume rendering. *Special Issue on 3D Imaging of the International of Imaging Systems and Technology 15*, 1 (2000), 144–153.

[48] KINDLMANN, G., WHITAKER, R., TASDIZEN, T., AND MÖLLER, T. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization 2003* (Seattle, USA, 2003), pp. 513–520.

[49] KNISS, J., PREMOZE, S., HANSEN, C., AND EBERT, D. Interactive translucent volume rendering and procedural modeling. *IEEE Visualization 2002 7*, 2 (2002), 109–116.

[50] KRUMHAUER, P., TYSGANKOV, M., REICH, C., AND EVGRAFOV, A. Efficient volume rendering using octree space subdivision. *SPIE, Visual Data Exploration and Analysis VI 3643*, 1 (1999), 211–219.

[51] KRÜGER, J., AND WESTERMANN, R. Acceleration techniques for gpu-based volume rendering. In *Proceedings of Visualization 2003* (Budmerice, Slovakia, 2003), pp. 231–238.

[52] LACROUTE, P., AND LEVOY, M. Fast volume rendering using shear-warp factorization. In *IEEE Transactions on Visualization and Computer Graphics* (1996), vol. 2, pp. 218–231.

[53] LEGAULT, R. *The Aliasing Problems in Two Dimensional Sampled Imagery*. New York: Plenum Press, 1973.

[54] LEVOY, M. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37.

[55] LL.D., N. P. D. Webster's revised unabridged dictionary (1913). by the C. & G. Merriam Co. Springfield, Mass., 1998.

[56] LORENSEN, W., AND CLINE, H. Marching cubes a high resolution 3d surface construction algorithm. *ACM Computer Graphics (Proceedings of SIGGRAPH'87) 21*, 4 (1987), 163–170.

[57] LU, A., MORRIS, C., EBERT, D., RHEINGANS, P., AND HANSEN, C. Non-photorealistic volume rendering using stippling techniques. In *IEEE Visualization 2002* (Piscataway,USA, 2002), pp. 211–218.

[58] LU, A., MORRIS, C., TAYLOR, J., EBERT, D., RHEINGANS, P., HANSEN, C., AND HARTNER, M. Illustrative interactive stipple rendering. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 127–138.

[59] LUM, E., AND MA, K.-L. Hardware-accelerated parallel non-photorealistic volume rendering. In *Proceedings of the International Symposium on Non-Photorealistic Rendering and Animation 2002* (Annecy, France, 2002), pp. 67–74.

[60] LUM, E. B., AND MA, K.-L. Non-photorealistic rendering using watercolor inspired textures and illumination. In *Ninth Pacific Conference on Computer Graphics and Applications* (Tokyo, Japan, 2001), pp. 322–331.

[61] MANSSOUR, I. H., FURUIE, S. S., OLABARRIAGA, S. D., AND FREITAS, C. M. Visualizing inner structures in multimodal volume data. In *SIBGRAPI'02* (Brazilia, Brazil, 2002), pp. 51–58.

[62] MARR, D. *Vision*. W. H. Freeman & Co., 1982.

[63] MARTÍN, D., FEKETE, J. D., AND TORRES, J. C. Silhouettes and non-photorealistic rendering: Flattening 3d objects using silhouettes. In *Proceedings of Eurographics 2002* (Saarbrücken, Germany, 2002), vol. 21, pp. 239–248.

[64] MARTÍN, D., AND TORRES, J. C. Rendering silhouettes with virtual lights. *Computer Graphics 20*, 4 (2001), 271–282.

[65] MAX, N. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (1995), 99–108.

[66] MEISSNER, M., HOFFMANN, U., AND STRASSER, W. Enabling classification and shading for 3d texture mapping base volume rendering using opengl and extensions. In *IEEE Visualization 1999* (San Francisco, USA, 1999), pp. 207–214.

[67] MORA, B., AND EBERT, D. S. Instant volumetric understanding with order-independent volume rendering. In *Computer Graphics Forum (Eurographics'04 Proceedings, Grenoble)* (Grenoble, France, 2004), pp. 489–497.

[68] MORRIS, C. J., AND EBERT, D. Direct volume rendering of photographic volumes using multi-dimensional color-based transfer functions. In *Joint EUROGRAPHICS-IEEE TCVG Simposium on Visualization 2002* (Barcelona, Spain, 2002), pp. 115–ff.

[69] MUELLER, K., MOLLER, T., AND CRAWFIS, R. Splating without the blur. In *Proceedings of IEEE Visualization 99* (1999), M. G. In David Ebert and B. H. Editors, Eds., pp. 363–370.

[70] MÄNTYLÄ, M., AND SULONEN, R. An introduction to solid modelling. *Computer Science Press 8*, 1 (1988), 45–60.

[71] NAGY, Z., AND KLEIN, R. Efficient ray-tracing of volume data. *ACM Transactions on Graphics 9*, 3 (1990), 245–261.

[72] NAGY, Z., AND KLEIN, R. High-quality silhouette illustration for texture-based volume rendering. In *Journal of WSCG* (Plzen, Czech Republic, 2004), pp. 301–309.

[73] NAVAZO, I., AYALA, D., AND BRUNET, P. A geometric modeller based on the exact octree representation of polyhedra. *Computer and Graphics Forum 5*, 2 (1986), 91–104.

[74] NAVAZO, I., AYALA, D., AND BRUNET, P. Extended octree representation of general solids with plane faces: Model structure and algorithms. *Computer and Graphics 13*, 1 (1989), 5–16.

[75] NORTHRUP, J., AND MARKOSIAN, L. Artistic silhouettes: A hybrid approach. In *Proceedings of NPAR 2000* (2000), pp. 87–96.

[76] OPPENHEIM, A. V., AND SCHAFER, R. W. *Digital Signal Processing*. Prentice-Hall, 1975.

[77] PFISTER, H., AND KAUFMAN, A. Scalable architechture for real time volume rendering. In *Symposium on Volume Visualization* (San Francisco, USA, 1996), I. R. Crawfis and C. H. editors, Eds., pp. 47–54.

[78] PYO, S., AND SHIN. Fast volume carving. In *Proceedings of Eurographics* (Saarbrücken, Germany, 2002), pp. 105–112.

[79] REZK-SALAMA, C., ENGEL, K., BAUER, M., GREINER, G., AND ERTL, T. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage-rasterization. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '00* (Saarbrücken, Germany, 2000), pp. 109–118,147.

[80] RHEINGANS, P. Expressive volume rendering. *Journal of WSCG 12*, 1-3 (February 2004), 58–65.

[81] SAITO, T., AND TAKAHASHI, T. Comprehensible rendering of 3d shape. *IEEE Transactions on Visualization and Computer Graphics 24*, 4 (1990), 197–206.

[82] SALISBURY, M., ANDERSON, C., LISCHINSKY, D., AND SALESIN, D. Scale-dependent reproduction of pen-and-ink illustrations. *Proceedings of SIGGRAPH 96 24*, 2 (1996), 461–468.

[83] SAMET, H. *Applications of spatial data structures*. Addison-Wesley, 1990.

[84] SAMET, H. *The design and analysis of spatial data structures*. Addison-Wesley, 1990.

[85] SAMMET, H., AND TAMMIEN, M. Bintrees, csg trees and time. *Computer and Graphics of the ACM 19*, 3 (1985), 121–131.

[86] SECORD, A., HEIDRICH, W., AND STREIT, L. Fast primitive distribution for illustration. In *Thirteenth Eurographics Workshop on Rendering (2002)* (New York, USA, 2002), pp. 527–534.

[87] SEEGMILLER, D. *Digital Character Design and Painting*. Graphics Series, 2003.

[88] SOUSA, M. C., AND BUCHANAN, J. Computer-generated graphite pencil rendering of 3d polygonal models. *Computer Graphics 18*, 3 (1999), 195–207.

[89] SOUSA, M. C., FOSTER, K., WYVILL, B., AND SAMAVATI, F. Precise ink drawing of 3d models. *Computer Graphics 22*, 3, 369–379.

[90] STOCKHAM, T. G. Image processing in the context of a visual model. *Proc. IEEE 60*, 7 (1972), 828–842.

[91] STOMPEL, A., LUM, E. B., AND MA, K.-L. Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques. In *10th Pacific Conference on Computer Graphics and Applications* (Tokyo, Japan, 2002), pp. 452–460.

[92] TAKAGI, S., FUJISHIRO, I., AND NAKAJIMA, M. Volumetric modeling of colored pencil drawing. In *Pacific Graphics '99* (Seoul, Korea, 1999), pp. 250–258.

[93] THIBAULT, W. C., AND NAYLOR, B. F. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics (SIGGRAPH '87 Proceedings) 21*, 4 (1987), 153–162.

[94] T.N.CORNSWEET. *Visual Perception*. Academic Press, 1971.

[95] TORRES, J. C., CANO, P., VELASCO, F., AND CONDE, F. Using octrees in non cartesian coordinate systems. Tech. rep., Technical Report LSI-95-2, Departamento de Lenguajes y Sistemas informáticos, Universidad de Granada, 1995.

[96] TREAVETT, S., AND CHEN, M. Pen-and-ink rendering in volume visualization. In *IEEE Visualization 2000* (Salt Lake City, USA, 2000), pp. 203–210.

[97] TUKEY, J. W. *Exploratory Data Analysis*. Addison Wesley, 1971.

[98] UDUPA, J., AND YUNG-KONG, T. A justification of fast surface tracking algorithm. *Graphical models and image processing 54*, 2 (1992), 15–26.

[99] VELASCO, F. *Representación y Visualización de Datos Volumétricos*. PhD thesis, Dpto. Lenguajes y Sistemas Informáticos, Universidad de Granada, Spain, 2002.

[100] WEILER, M., WESTERMANN, R., HANSENY, C., ZIMMERMANY, K., AND ERTL, T. Level-of-detail volume rendering via 3d textures. In *Proceedings of the 2000 IEEE symposium on Volume visualization* (Salt Lake City, USA, 2000), pp. 7 – 13.

[101] WESTOVER, L. Footprint evaluation for volume rendering. *Computer Graphics 24*, 4 (1990), 367–376.

[102] YOUNG, E. A. Octree space partitioning: A raytracing acceleration technique. In *UNF First Annual COCSE Symposium* (North Florida, USA, 2003), pp. 11–20.

[103] YOUNG, I. T., AND VLIET, L. J. V. Recursive implementation of the gaussian filter. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (Los Alamitos, USA, 1995), pp. 139–151.