Introducción a la Computación Gráfica

Claudio Delrieux
Departamento de Ingeniería Eléctrica
Universidad Nacional del Sur
claudio@acm.org

Prefacio

El objetivo de este texto es servir de apoyo al curso introductorio a la Computación Gráfica a dictarse para el Postgrado en Informtica de la UMSS durante Agosto y Septiembre de 2000. El curso busca proveer un fundamento firme en el tema, tanto en la teoría como en los aspectos de implementación de sistemas gráficos. La revolución cultural producida por la aparición de la Mac y la PC ha acelerado enormemente el contacto de estos temas al gran público. En la actualidad es posible encontrar sistemas de Computación Gráfica en aplicaciones de una gran diversidad, todos los cuales tienen una base conceptual común, la cual tratamos de cubrir aquí. También se incluyen algunos temas avanzados para orientar a los lectores inquietos hacia la bibliografía especializada.

Hemos buscado adaptar los temas a un auditorio lo más amplio posible. Cualquier graduado o alumno avanzado de Ingeniería, Ciencias de la Computación o disciplinas afines debería poder abordar este texto sin inconvenientes. En particular, se requiere una base adecuada en programación en algún lenguaje estructurado (hemos elegido Pascal) y conocimientos elementales de Geometría Analítica. Los ejercicios sugeridos al final de cada Capítulo tienen como objetivo orientar en la secuencia de construcción y uso de los distintos componentes de un sistema gráfico. La resolución de cada ejercicio, además, sirve de apoyo a la comprensión de los temas teóricos.

Agradecimientos

Este texto es el resultado del esfuerzo y la colaboración de muchas personas desde que, en 1985, el Profesor Daniel Magni, del Departamento de Ingeniería Eléctrica de la Universidad Nacional del Sur, propone y dicta por primera vez la materia optativa Sistemas de Programación III, con un programa destinado específicamente al dictado de temas de Computación Gráfica. Dicha materia ha sido dictada todos los años desde entonces, y fue por mucho tiempo la única materia de grado de Computación Gráfica dictada en las Universidades de Argentina. En aquel entonces cursé la materia como alumno, al año siguiente colaboré como docente alumno, y desde 1990 estoy a cargo de su dictado. El crecimiento y mejora del grupo docente se debe fundamentalmente al esfuerzo y capacidad de los alumnos, que con su iniciativa y talento obligaron año tras año a una superación constante de quienes estuvimos a cargo del dictado y de los trabajos prácticos. De todos los docentes que aportaron con su paso por SPIII quisiera agradecer especialmente a Daniel Formica, Fernando Caba y Andrés Repetto.

También deseo agradecer a quienes aportaron su colaboración material que ayudó a mejorar la calidad y presentación de este texto, en particular a Miguel Danzi, Silvia Castro, Andrea Silvetti y Gustavo Patow. También se incluyen como ejemplos ilustrativos algunas figuras realizadas a partir de implementaciones de Sebastián Urbicain Mario Carro y Jorge Anchuvidart. Se incluyen grandes partes de las notas de cursos dados en la Primera Escuela Internacional de Computación (durante el CACiC'97, La Plata), la Escuela de Ciencias Informáticas (en la UBA, 1998), y en RIO 1999.

Bahía Blanca, Julio de 2000

Índice

1 Introducción **7**

1.1 Motivación 8

1.2 Aplicaciones de la Computación Gráfica 8

1.3 Desarrollo Histórico de la Computación Gráfica 9

2 Algoritmos y Conceptos Básicos **11**

2.1 Dispositivos Gráficos 12

2.1.1 Dispositivos de vectores 12

2.1.2 Dispositivos de raster 13

2.1.3 Hardware gráfico para monitores 13

2.2 Técnicas de Discretización 16

2.2.1 El sistema de coordenadas físico 18

2.2.2 Primitivas gráficas 19

2.2.3 Especificaciones de una discretización 19

2.2.4 Métodos de discretización 20

2.3 Discretización de Segmentos de Rectas 20

2.3.1 Segmentos de recta DDA 20

2.3.2 Segmentos de rectas por Bressenham 23

2.4 Discretización de Circunferencias 26

2.4.1 Discretización de circunferencias por DDA 26

2.4.2 Discretización de Bressenham para circunferencias 29

2.5 Discretización de Polígonos 30

2.6 Ejercicios 33

2.7 Bibliografía recomendada 34

3 Computación Gráfica en Dos Dimensiones **35**
3.1 Estructuras de Datos y Primitivas . . . 36
3.2 Transformaciones y Coordenadas Homogeneas . . . 37
3.2.1 Transformaciones afines . . . 38
3.2.2 Coordenadas homogéneas . . . 39
3.2.3 Transformaciones revisitadas . . . 42
3.3 Representación Estructurada . . . 42
3.4 Windowing y Clipping . . . 43
3.4.1 Windowing . . . 46
3.4.2 Clipping . . . 47
3.4.3 La "tubería" de procesos gráficos . . . 49
3.5 Implementación de Paquetes Gráficos . . . 51
3.6 Ejercicios . . . 53
3.7 Bibliografía recomendada . . . 54

4 Aproximación e Interpolación de Curvas **55**
4.1 Motivaciones . . . 56
4.1.1 Especificaciones y requisitos . . . 58
4.1.2 La representación paramétrica . . . 60
4.2 Interpolación de Curvas . . . 60
4.2.1 Interpolación de Curvas de Lagrange . . . 60
4.2.2 Interpolación de Curvas de Hermite . . . 63
4.3 Aproximación de Curvas I: de Casteljau, Bernstein y Bézier . . . 67
4.3.1 Construcción de parábolas . . . 67
4.3.2 El algoritmo de de Casteljau . . . 69
4.3.3 La base de Bernstein . . . 71
4.4 Aproximación de Curvas II: B-Splines . . . 75
4.4.1 Parámetro local . . . 76
4.4.2 La base de Splines . . . 78
4.4.3 B-Splines no uniformes . . . 83
4.4.4 B-Splines cúbicos uniformes . . . 86
4.4.5 Evaluación de B-Splines cúbicos . . . 90
4.5 Ejercicios . . . 92
4.6 Bibliografía recomendada . . . 92

5 El Color en Computación Gráfica **93**
5.1 Introducción . . . 94
5.2 Aspectos Físicos del Color . . . 94
5.3 Aspectos Fisiológicos del Color . . . 95
5.4 El Diagrama CIEXY de Cromaticidad . . . 97
5.5 Espacios Cromáticos . . . 102
5.6 Representación de Color en Computación Gráfica . . . 105
5.7 Paletas Estáticas y Dinámicas . . . 107
5.8 Ejercicios . . . 110
5.9 Bibliografía recomendada . . . 110

6 Computación Gráfica en Tres Dimensiones **111**
6.1 Objetivos de la Computación Gráfica Tridimensional . . . 112
6.2 Transformaciones y Coordenadas Homogeneas . . . 113
6.3 Proyecciónes y Perspectiva . . . 115
6.4 Primitivas y Estructuras Gráficas . . . 119
6.5 Clipping y Transformación de Viewing . . . 122
6.6 Cara Oculta . . . 125
6.6.1 Eliminación de líneas ocultas en superficies funcionales . . . 127
6.6.2 Clasificación de los métodos generales . . . 129
6.6.3 El algoritmo SPIII . . . 131
6.7 Ejercicios . . . 135
6.8 Bibliografía recomendada . . . 136

7 Modelos de Iluminación y Sombreado **137**
7.1 El Realismo en Computación Gráfica . . . 138
7.2 Modelos de Iluminación . . . 139
7.3 Sombreado de Polígonos . . . 147
7.4 Ray Tracing . . . 152
7.5 Radiosidad . . . 156
7.6 Comparación de los métodos de rendering . . . 157
7.7 Mapas de atributos . . . 158
7.8 Ejercicios . . . 160
7.9 Bibliografía recomendada . . . 160

8 Aproximación de Superficies con Puntos de Control **161**
8.1 Introducción . . . 162
8.2 Aproximación de Superficies I: Producto Tensorial de Curvas . . . 162
8.2.1 Superficies de Bézier . . . 163
8.2.2 Superficies B-Spline bicúbicas . . . 168
8.3 Aproximación de Superficies II: Dominios Triangulares . . . 169
8.3.1 Algoritmo de de Casteljau bivariado . . . 171
8.3.2 Polinomios de Bernstein multivariados . . . 171
8.4 Ejercicios . . . 174
8.5 Bibliografía recomendada . . . 174

9 Temas Avanzados **175**
9.1 Modelos de refracción . . . 176
9.2 Animación . . . 178
9.3 Visualización Científica . . . 178
9.4 Modelos no Determinísticos y Fractales . . . 183

A Notación **185**

B Elementos Matemáticos de la Computación Gráfica **187**
B.1 Álgebra de Matrices . . . 187
B.2 Álgebra de vectores . . . 188
B.3 Transformaciones Lineales y Afines . . . 189

Referencias **191**

1 Introducción

1.1 Motivación

La Computación Gráfica ha sido una de las ramas de las Ciencias de la Computación de mayor impacto social y acercamiento al público en general. La graficación está casi siempre asociada con la interactividad, por lo que actualmente casi todo sistema o programa interactivo tiene una interfase gráfica. Esto es así no solo en aplicaciones específicas de graficación matemática y estadística, sino también como medio de producir metáforas visuales para otros sistemas informáticos, como ocurre actualmente en los lenguajes de programación visual, las interfases gráficas con sistemas de ventanas y la metáfora del *desktop*, y en los sistemas de diseño asistido. En la actualidad, para cualquier persona, el acceso a Internet sin posibilidades gráficas es considerado una antigüedad. Aún las personas que no utilizan computadoras en su vida diaria encuentran gráficos computacionales en casi todos los aspectos de la cultura.

También se han popularizado los sistemas que traducen modelos abstractos a imágenes visualizables, como sucede con las simulaciones ingenieriles que representan gráficamente los resultados provenientes de modelos matemáticos para auxiliar su comprensión, o con los datos obtenidos por sensores (satélites, tomógrafos o telémetros en general) que se representan visualmente en los boletines meteorológicos o en las pantallas de los médicos.

Todos estos sistemas, utilizados para fines tan diversos, tienen un fundamento subyacente que consiste en una serie de técnicas derivadas de la aplicación computacional de la Geometría Analítica. En este texto presentamos los fundamentos teóricos y los detalles de implementación de los temas que constituyen el núcleo de la Computación Gráfica en tres dimensiones. El estudio de estos temas es indispensable para comprender la idiosincracia de la disciplina, y de todos los sistemas que, como mencionamos, se basan de una u otra forma en la graficación.

1.2 Aplicaciones de la Computación Gráfica

Los gráficos proveen uno de los medios más naturales y potentes de comunicarse. El procesamiento cerebral del aparato visual está altamente desarrollado, y por lo tanto es capaz de reconstruir, procesar, interpetar, recordar y cotejar una enorme cantidad de información en un tiempo apenas perceptible. Por su parte, la evolución tecnológica y la rápida difusión de las computadoras determina que en la actualidad la Computación Gráfica es el medio de producción de imágenes más importante en la actualidad, superando a la fotografía, diseño y artes gráficas en general, y compitiendo con el cine y la televisión. Tiene la ventaja adicional de poder reproducir imágenes "virtuales", que no necesariamente existen o se pueden ver en la realidad.

Entre las numerosas aplicaciones, podemos dar una breve descripción representativa, la cual es sin duda cada día menos completa:

Interfases Actualmente la mayoría de los sistemas operativos, utilitarios, procesadores de texto, etc. tienen una interfase gráfica basada en la metáfora visual del escritorio, con íconos, menúes descolgables, barras deslizantes y muchas otras facilidades. De esa manera, el uso del teclado se vuelve necesario solamente para el ingreso de texto.

Industria del entretenimiento Aquí podemos contar tanto la producción de video-juegos, películas y cortos de dibujos animados, posproducción y efectos especiales de peliculas, publicidad, y también el desarrollo de programas utilitarios destinados a la creación de productos en estos rubros.

Aplicaciones comerciales Son cada vez más comunes los sistemas para elaboración de presentaciones comerciales, incluyendo cartillas gráficas, diagramación automática, y publicación

electrónica en general. Con el advenimiento de Internet y HTML, el desarrollo del comercio y las oficinas virtuales ha sido dramático, y el uso de gráficos es el medio indispensable para agregar distinción y atractivo a los sitios en la WWW.

Diseño asistido El CAD en general, desde el dibujo de planos hasta el desarrollo de chips VLSI pasando por cientos de otras aplicaciones, también tiene un gran auge en la actualidad. En todos los casos la representación gráfica de la información es la clave del funcionamiento.

Aplicaciones Científicas Aquí podemos contar desde los sistemas de simulación (cinemática, por elemento finito, etc.) hasta la visualización de fenómenos abstractos y su representación gráfica por medio de metáforas visuales adecuadas.

Cartografía y GIS Los gráficos por computadora son actualmente utilizados como soporte para los sistemas de información geográfica (GIS) y todas las aplicaciones relacionadas (turismo, geología, minería, clima, urbanismo, etc.).

1.3 Desarrollo Histórico de la Computación Gráfica

Hasta hace aproximadamente 20 años, la Computación Gráfica era un campo pequeño y especializado, dado el costo de los equipos necesarios para la implementación de sistemas gráficos. Para el público en general, el tema consistía en desarrollar aplicaciones en sistemas *mainframe* sobre paqetes gráficos *built in*, los cuales producían una salida visual para programas científicos o comerciales en los que se necesitaban elaborar gráficos estadísticos como histogramas, diagramas o trazado de funciones. El trabajo de investigación en Computación Gráfica, como es entendido en la actualidad, parecía entonces una tarea fútil orientada a objetivos pretensiosos y de escasa utilidad. Desde entonces, la Computación Gráfica experimento dos grandes cambios de paradigma en sus objetivos, pasando de los gráficos estadísticos a la síntesis realista de imágenes tridimensionales (a mediados de los 70) y luego a la representación gráfica de datos u objetos abstractos, en lo que hoy se conoce como *Visualización Científica* (a mediados de los 80). Actualmente es posible afirmar que el campo está atravesando un nuevo cambio paradigmático, producido esencialmente por el desarrollo de la Realidad Virtual, junto con la tecnología de sensores y actuadores cinemáticos, y por supuesto Internet con acceso a alta velocidad. Conectarse desde el hogar a una simulación gráfica del *Pathfinder* y recorrer un Marte virtual es una posibilidad casi rutinaria, aún para el no experto.

Podemos afirmar que la Computación Gráfica quedó definida en el trabajo doctoral de Ivan Sutherland en el MIT en 1963 [80]. Su desarrollo del sistema interactivo de graficación Sketchpad fue el fundamento sobre el cual se apoyaron las ideas de la mayor parte de las ideas de la Computación Gráfica 2D y 3D (por ejemplo, los Capítulos 3 y 6 de este texto). Por su parte, a mediados de la década del '60 comenzaron a desarrollarse en Francia los métodos de diseño asistido de curvas y superficies. En unos pocos años comenzó a existir una disciplina científica, la cual se plasma con la creación de Sociedades Profesionales como el ACM SIGGRAPH en 1969. La década del 70 fue esencialmente de cambios tecnológicos. El descenso del costo de la memoria volátil produjo el reemplazo de los costosos equipos basados en display de vectores, por los dispositivos de la tecnología de *raster* que actualmente se utiliza. Básicamente, cada *"pixel", picture cell* o celda de pantalla, tiene un respaldo en un conjunto de memoria dedicada. Ello motivó el desarrollo de los algoritmos de digitalización de primitivas gráficas como los que se reseñan en el Capítulo 2.

Asociado al cambio tecnológico en los monitores, surgieron los nuevos dispositivos de entrada (mouse, tablas digitalizadoras) y también de impresión en blanco y negro o color de alta calidad y precios razonables. Por lo tanto, los métodos de diseño de curvas y superficies comenzaron a tener una salida visual que antes era imposible con los displays de vectores. La mayoría de los métodos fundamentales para diseño de curvas y superficies, presentados en los Capítulos 4 y 8,

fueron desarrollados y perfeccionados durante esta década. Otra de las posibilidades originada con el cambio tecnológico fue la representación de colores. Esto motivó el desarrollo de modelos y espacios cromáticos adecuados para la relación costo-performance de los monitores (ver Capítulo 5). Con la posiblidad del color, el primer cambio de paradigma, la búsqueda de la síntesis de imágenes con realismo, llegó a su máxima expresión con el desarrollo de modelos empíricos de iluminación y sombreado (ver Capítulo 7). Estos modelos permiten la representación de escenas en las cuales se simulan los fenómenos óptico-físicos con un costo computacional bastante bajo para los resultados obtenidos. También en los 70 comienzan a establecerse estándares, dado que las aplicaciones gráficas se difundieron en medio de una diversidad de cambios y evolución, y una gran cantidad de proovedores, fabricantes y diseñadores.

Sin duda la década de cambios más vertiginosos fue la de 1980. El surgimiento de las máquinas PC, aunque con capacidades gráficas limitadas, permitió la popularización de sistemas y aplicaciones que crearon un mercado exigente y competitivo (por ejemplo con el Autocad). También comenzaron a diseñarse herramientas gráficas de interfase hombre máquina, como por ejemplo el sistema operativo de la Macintosh II, los lenguajes de programación visual y el hipertexto. El rol que no alcanzaron a cumplir los Comités de estandarización (por ejemplo, el GSK fue aprobado recién en 1985, cuando hacía varios años que ya era obsoleto) fue cubierto por las compañías comerciales que al crear una aplicación novedosa se transformaban en estándares de facto en el mercado (por ejemplo el Poscript, el OpenGL y X Windows).

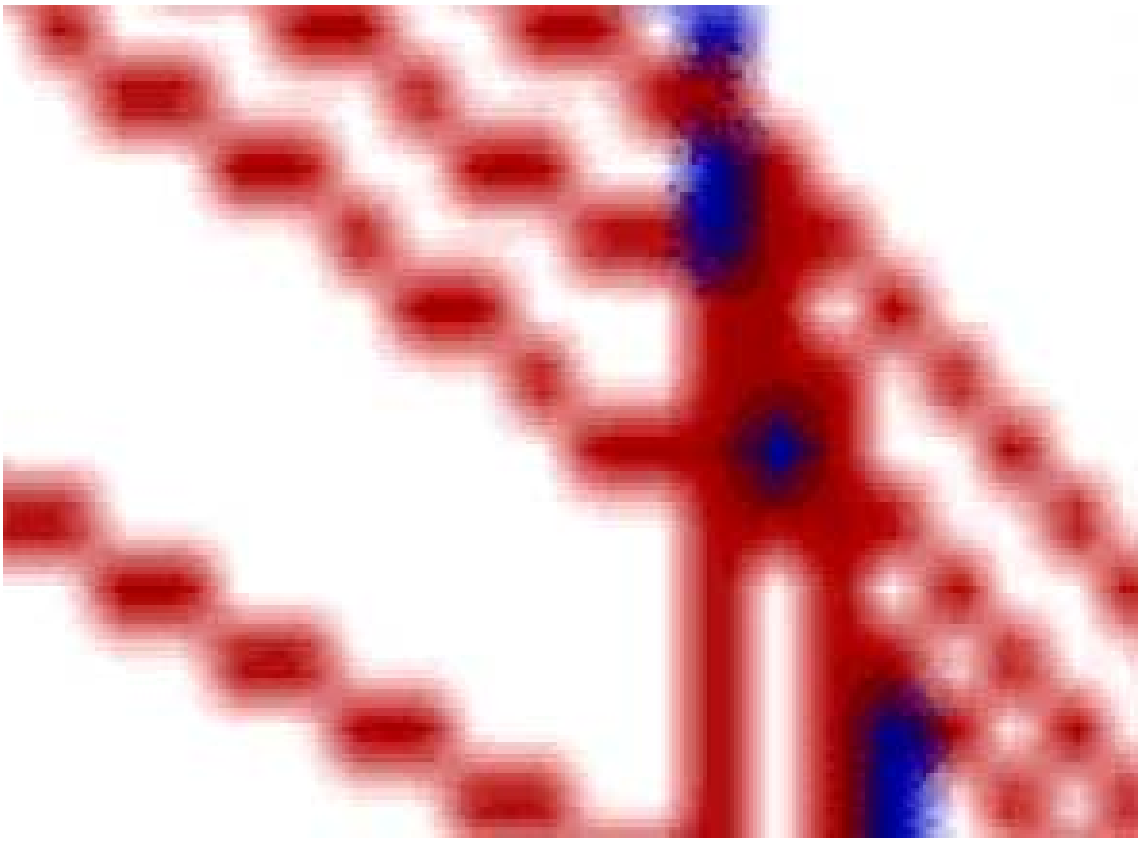
También esta década marcó el segundo cambio de paradigma, porque la evolución de los modelos gráficos, junto con la capacidad de representación de los monitores y la integración de los sistemas gráficos a otro tipo de aplicaciones (simulaciones en ingeniería, sensores remotos, datos de satélites, etc.) permitieron desarrollar herramientas para la representación gráfica de conjuntos enormemente complejos de datos. Estas ideas, que con el tiempo fueron el fundamento de la *Visualización Científica*, apelan a la enorme capacidad de comprensión visual humana (ver Sección 9.3). De esa manera es posible representar, por ejemplo, millones de datos meteorológicos en un único "gráfico" que permite comprender a golpe de vista las características esenciales de una determinada situación climática.

La popularización de la Computación gráfica significó, además, el surgimiento y desarrollo de aplicaciones en las áreas más diversas. Durante los '80 comenzaron a utilizarse herramientas gráficas para el diseño en Ingeniería en todas sus actividades, desde aviones y barcos hasta circuitos integrados. En Arquitectura e Ingeniería Civil se utilizan sistemas para la simulación, el diseño y la elaboración y análisis de modelos. En Medicina podemos mencionar desde el diagnóstico por imágenes hasta la simulación y planeamiento de operaciones quirúrgicas o el desarrollo de implantes. En animación y videojuegos se dio un desarrollo espectacular en la calidad e imaginación con los que surgieron universos de fantasía. Cada uno recordará películas famosas de Hollywood videoclips, publicidades, videojuegos para Sega, etc.

En la actualidad, el desarrollo tecnológico permite integrar multimedios con acceso directo a redes, adquirir datos a través de guantes y sensores y actuadores cinemáticos, y procesar en tiempo real la simulación de un observador sumergido en una realidad virtual. Las posibilidades de los *browsers* para Internet están resultando cada vez más obsoletas, por lo que no es aventurado pensar que la integración de estas tecnologías significará un nuevo cambio de paradigma en un futuro cercano.

Todo este vasto campo de aplicación de la Computación Gráfica es imposible de cubrir en un libro introductorio. Sin embargo, para cualquiera que desee especializarse en algún tema específico, la base común es aproximadamente la presentada aquí. Esperamos cumplir con las expectativas del lector al acercarle los fundamentos de la Computación Gráfica de una manera clara y precisa, con el deseo de brindarle la mayor motivación y entusiasmo para seguir recorriendo este camino.

2 Algoritmos y Conceptos Básicos



2.1 Dispositivos Gráficos

Los resultados gráficos de una aplicación pueden mostrarse en una gran variedad de dispositivos de salida. Normalmente estos dispositivos son o bien de pantalla o bien de impresión. Sin embargo, desde el punto de vista de la Computación Gráfica, es importante otra clasificación, referida al modo en que los mismos son manejados por la computadora. De esa manera, podemos ver que existen dispositivos de los siguientes tipos:

- **Dispositivos de vectores**, los cuales reciben de la computadora la información geométrica de la localización y tamaño de las primitivas que soportan, de las cuales producen una reproducción "caligráfica".
- **Dispositivos de raster**, los cuales reciben de la computadora la información de una serie de pixels, los cuales son posicionados en forma contigua.

Los dispositivos de vectores fueron los primeros en desarrollarse, pero luego del vertiginoso descenso en el costo de la memoria volátil, a partir de la década del 70 se hicieron más baratos los dispositivos de raster. Como veremos en este Capítulo, ésto implica un cambio en la manera de representar las primitivas gráficas (usualmente dichas primitivas son el punto, el segmento de recta y la circunferencia o el círculo).

2.1.1 Dispositivos de vectores

Actualmente estos dispositivos son más caros, pero tienen ciertas ventajas que los hacen únicos. Por ejemplo, tienen mucha mejor resolución y precisión que los dispositivos de raster, y requieren un ancho de banda de comunicación mucho menor dado que no reciben la discretización completa de las primitivas sino solamente su posición.

Plotters: Grafican en una hoja (que en algunos casos puede ser de gran tamaño) sobre la cual se desliza una pluma movida por motores de pasos de gran precisión. En los plotters de tambor, la pluma se desliza en sentido horizontal y el papel en sentido vertical. En los plotters planos (más económicos), el papel está fijo y la pluma realiza todos los movimientos. Son usuales las resoluciones del orden de los 10000×10000. Es posible utilizar colores por medio de varias plumas. Son ideales para la graficación rápida y precisa de planos.

Displays de almacenamiento: Al igual que televisores y monitores, estos dispositivos son pantallas de rayos catódicos, pero difieren en ciertos aspectos tecnológicos. Esencialmente, la pantalla tiene cierta "memoria" electrostática que mantiene visibles los elementos graficados con muy alta precisión y sin la necesidad de refresco. Por lo tanto, una imagen muy compleja a la cual se van agregando elementos en orden es idealmente representada por estos dispositivos. Un elemento se representa "pintándolo" por medio de una serie de recorridas del cañón electrónico. El borrado, sin embargo, no puede hacerse en forma selectiva, por lo que no se puede alterar la posición de un elemento sin tener que borrar y redibujar todos los demás. Sin embargo, su precisión y velocidad sin necesidad de memoria volátil los hace ideales para la representación de imágenes de radares.

2.1.2 Dispositivos de raster

Impresoras de matriz: Era hasta hace poco el dispositivo de impresión más común. Recibe de la computadora la información gráfica como una secuencia de líneas, las cuales va reproduciendo con una cabeza impresora (por medio del golpe de martillos o el rocío de tinta).

Impresoras Laser: Recibe de la computadora la información gráfica como una secuencia de líneas, las cuales almacena en una memoria local. Dicha memoria es utilizada para comandar la intensidad de un haz laser que recorre línea por línea el papel, mientras es expuesto al contacto del toner. Donde el haz incide con gran intensidad, el papel se dilata por el calor y absorbe el toner.

Monitores: Se han popularizado enormemente a partir del descenso en el precio de la memoria volátil y el incremento constante en la calidad de las prestaciones (resolución, color, precisión). Esencialmente se comportan de una manera similar a un receptor de televisión, excepto por el hecho de que reciben la señal de video y sincronismo en forma directa de la computadora y no a través de una portadora de radio. Al igual que con las impresoras de matriz, la imagen se construye línea por línea, en sentido horizontal primero (de izquierda a derecha) y vertical después (de arriba abajo). Debe existir un refresco de la imagen en memoria, la cual es recorrida por la tarjeta gráfica de la computadora para producir las líneas de barrido. Por lo tanto, el elemento esencial en estos dispositivos es la tarjeta gráfica, la cual veremos en detalle más adelante. Los monitores más poulares pueden tener resoluciones de hasta 1200×1024 (aunque este límite avanza día a día), con una cantidad de colores limitada por las prestaciones de la tarjeta gráfica. Esto representa una calidad más que aceptable para la mayor parte de las aplicaciones.

2.1.3 Hardware gráfico para monitores

Los dispositivos de raster requieren un refresco permanente de la discretización de la salida gráfica. En el caso de los monitores, dicho refresco se realiza en un segmento de la memoria volátil de la computadora denominada *frame buffer* o buffer de pantalla, que usualmente se implementa por medio de memoria RAM de alta velocidad localizada dentro de la tarjeta gráfica. El buffer de pantalla es accedido en forma rítmica por el generador de video, que es el encargado de "componer" la señal de video que va hacia el monitor. Al mismo tiempo, al producirse una salida gráfica por parte de la CPU de la computadora, la misma debe ser discretizada y almacenada en el buffer de pantalla. Este acceso debe ser permitido solamente en los momentos en los que el generador de video no está accediendo al buffer, y por lo tanto se requiere el uso de un árbitro que mantenga abierto el acceso al buffer solo en esos casos (ver Figura 2.1).

El temporizado es crítico en el manejo del buffer de pantalla, por lo que se requiere memoria RAM de alta velocidad, mucho mayor que la velocidad requerida para la RAM de la CPU. Por ejemplo, en una norma de video de 1024 pixels por línea, la pantalla es refrescada 35 veces por segundo a una tasa de aproximadamente un millon de pixels por pantalla. Esto significa que en promedio el buffer de pantalla es accedido 35 millones de veces por segundo por el generador de video, lo cual requiere una velocidad de acceso a memoria de aproximadamente 30ns para cumplir solo con el refresco de pantalla. En una situación como ésta, utilizar memoria de 25ns. para el buffer de pantalla permite utilizar solamente un pico de 5 millones de accesos por segundo para la CPU, lo cual en muchos casos es insuficiente si se tiene en cuenta que el acceso entre la CPU y la tarjeta gráfica por el bus ISA debe cumplir cierto protocolo que hace más lenta la comunicación.

Otro esquema posible para manejar la memoria de pantalla es utilizar la tecnología de *bus local* (difundida alrededor de 1993 con las motherboard 486 y tarjetas Vesa Local Bus). Básicamente la idea es evitar el uso del bus de datos ISA para interconectar la tarjeta gráfica con la CPU. De ese

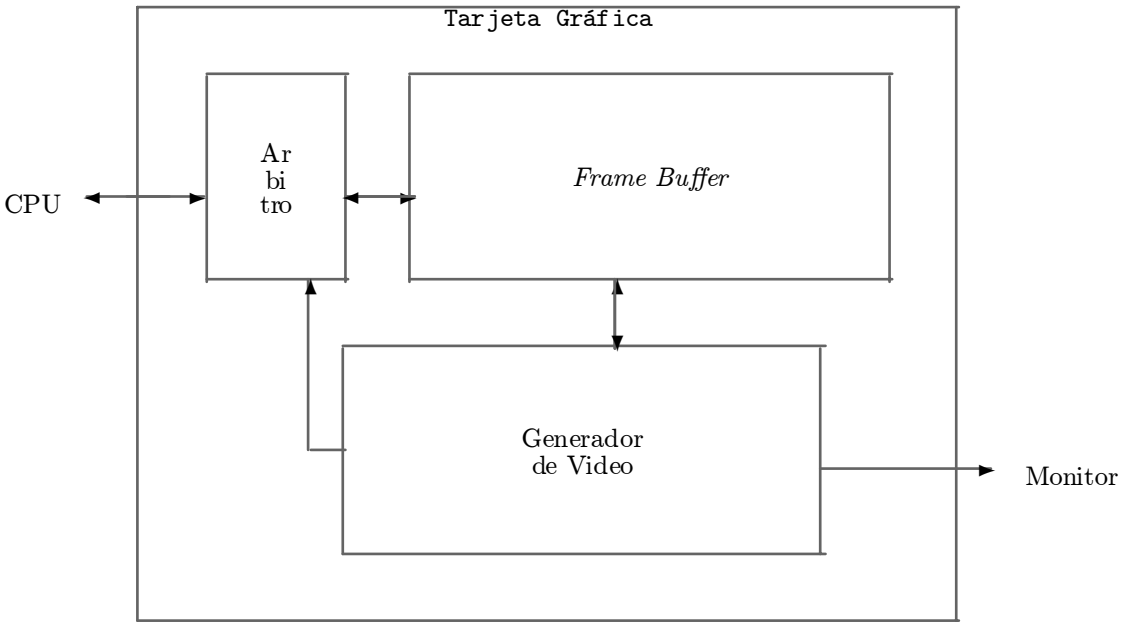


Figura 2.1 *Componentes básicos de una tarjeta gráfica de raster.*

modo se utiliza un segundo bus (llamado bus local), normalmente de 32 bits en vez de 16, con la velocidad del reloj externo del microprocesador (50Mhz. en vez de 8.33) y con capacidad de acceso directo a memoria (ver Figura 2.2). Este tipo de configuraciones permite una mejor utilización del ancho de banda marginal de la memoria del frame buffer, y por lo tanto, en determinadas aplicaciones, como por ejemplo animaciones, la prestación de un mismo hardware aumenta en un orden de magnitud solamente al modificar la configuración de acceso.

La evolución del hardware gráfico en PC siguió esquemas similares, utilizándose primero el port PCI y actualmente el AGP. Las tarjetas gráficas PCI tienen una configuración similar a las tarjetas ISA dado que el port está pensado para dispositivos genéricos. Sin embargo, el acceso es en 32 bits, a frecuencia de reloj externo, y en algunos casos con posiblidades de acceso directo a memoria. El port AGP, por su parte, permite un acceso en 64 bits a frecuencia interna del micro (400Mhz. en algunos casos) directamente al bus interno y al cach/'e. Por lo tanto es esperable un rendimiento cientos de veces mayor que con las tarjetas ISA.

Por otra parte, como veremos a lo largo de los distintos capítulos, muchas de las operaciones matemáticas necesarias dentro de la Computación Gráfica siguen un procesamiento disciplinado que puede en muchos casos implementarse directamente en el hardware de la tarjeta. Es por dicha razón que han surgido tarjetas aceleradoras por hardware (PCI o AGP) que permiten que una aplicación se deslinde del trabajo de efectuar las transformaciones geométricas, pintado de polígonos, el mapeo de texturas, etc.

Recapitulando, la clave del funcionamiento de la tarjeta gráfica no está en los requisitos de memoria, sino en la estructura del generador de video. El generador de video debe recorrer la memoria del buffer de pantalla y entregar las líneas de barrido al monitor dentro de una determinada norma de video. Dicha norma puede ser una norma de televisión (PAL o NTSC) o de monitor

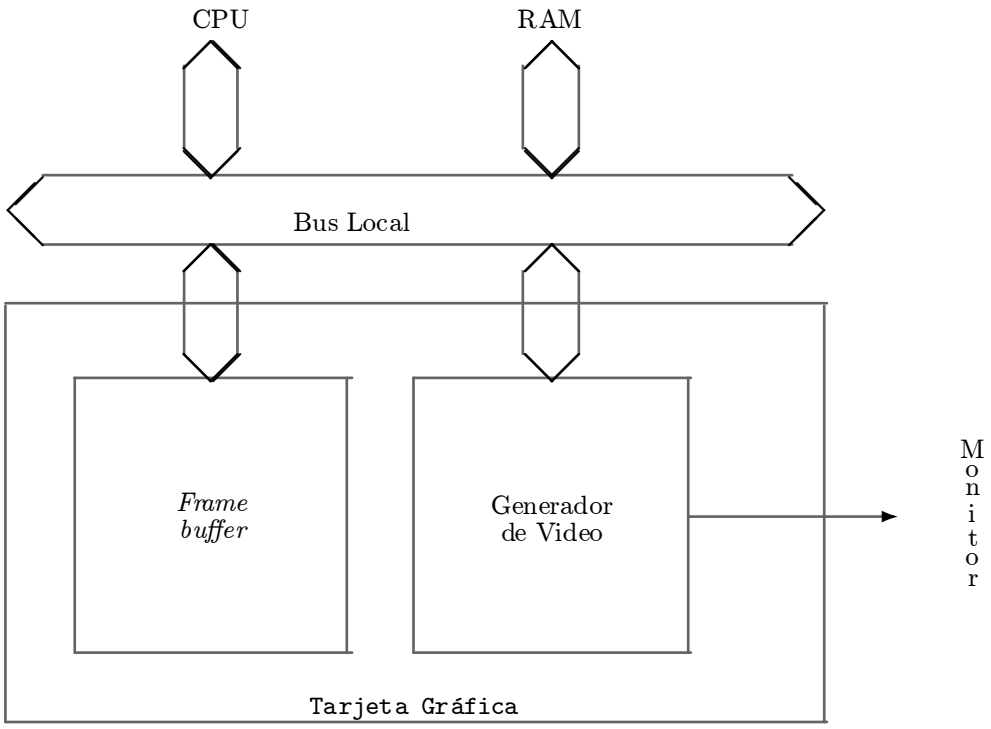


Figura 2.2 *Tarjeta gráfica basada en tecnología* local bus.

(1024×768, 800×600, etc.). Entonces, el barrido es producido por un generador de barrido cuyas frecuencias horizontal y vertical son programables en función de la norma que se utiliza.

Las señales de barrido son enviadas al monitor, pero también se utilizan para encontrar la posición de memoria en la cual está almacenada la información gráfica de cada pixel que constituye una línea de barrido. Esto se realiza por medio de una unidad aritmética que encuentra una dirección lineal a partir de los valores de la señal de barrido horizontal y vertical. La dirección lineal habilita la salida del valor almacenado en un lugar de la memoria del buffer de pantalla. Dicho valor es transformado en información gráfica por medio de una tabla de color (ver Figura 2.3), excepto en el modo *true color* que se explicará más adelante.

En las tarjetas gráficas se representa el color por medio del espacio cromático RGB (ver el Capítulo 5). Esto significa que el color de cada pixel se representa por medio de una terna de valores de las componentes en rojo, verde y azul, respectivamente, que tiene dicho color. Normalmente es innecesario almacenar dicha terna para cada pixel. En cambio se utiliza una tabla de colores predefinidos para lograr un uso más eficaz del buffer de pantalla, dado que en el mismo no se almacena el color del pixel sino el índice al color correspondiente en la tabla de colores. Por ejemplo, si se utilizan solamente 256 colores simultaneos, entonces es necesario almacenar solamente 1 byte de índice para cada pixel (ver Figura 2.4).

En dicho modelo se utiliza la sentencia `putpixel(x,y,c)` para acceder al buffer de pantalla y setear el pixel en la posición `(x, y)` con el índice de color `c`, con `x, y, c` de tipo `word`. Para setear la tabla de colores se utiliza la sentencia `setrgbpalette(c,r,g,b)`, donde `c` es el índice del

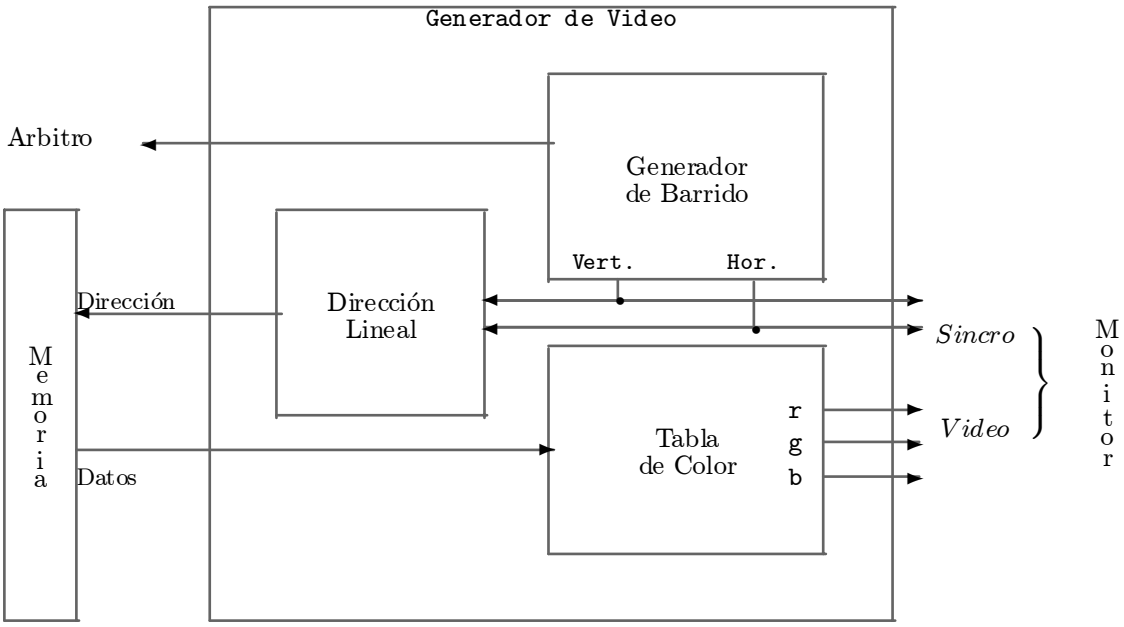


Figura 2.3 *Estructura del generador de video.*

color a setear, y `r, g, b` son las componentes en el modelo RGB del color. En los modos gráficos VGA y super VGA, los parámetros `r, g, b` son de tipo `word`, pero se truncan los dos bit menos significativos, dado que el rango efectivo de cada componente es de 0 a 63. Esto permite definir 256 colores simultáneos de entre 256K colores definibles. Por lo tanto es conveniente utilizar una aritmética dentro de dicho rango para representar los colores, y multiplicar por 4 en el momento de la llamada.

El manejo se simplifica en el modelo *true color*, en el cual cada pixel tiene alocada la memoria para representar las componentes en RGB del color en que está seteado (ver Figura 2.5). Esto representa triplicar el tamaño del buffer de pantalla y el ancho de banda de la memoria asociada, lo cual tiene un costo bastante elevado, pero permite representar 16.7M colores simultáneos Como ya se dijo, en algunos casos puede ser necesario el costo, pero normalmente un modelo como el de la Figura 2.4 puede ser suficiente.

2.2 Técnicas de Discretización

A partir de este punto, y en lo que resta del Capítulo, nos vamos a comprometer con el modelo de dispositivo de raster. Para conseguir independencia de dispositivo, entonces, es necesario adoptar un conjunto de primitivas y establecer una serie de métodos que permitan representar dichas primitivas en nuestro dispositivo de raster satisfaciendo un conjunto de especificaciones.

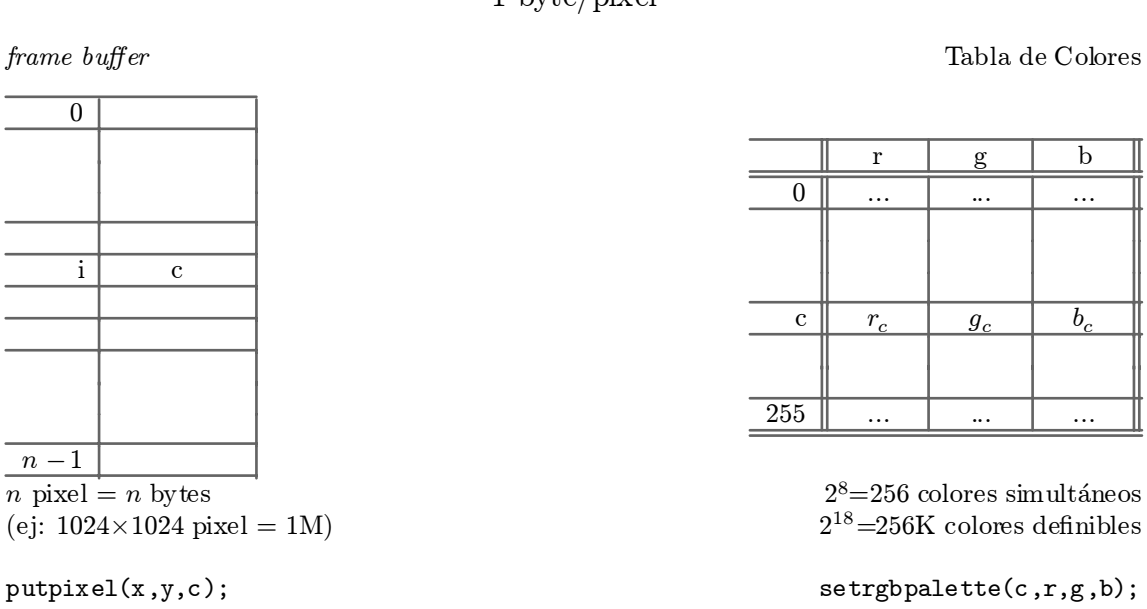


Figura 2.4 *Modelo de memoria y tabla de colores a 1 byte/pixel.*

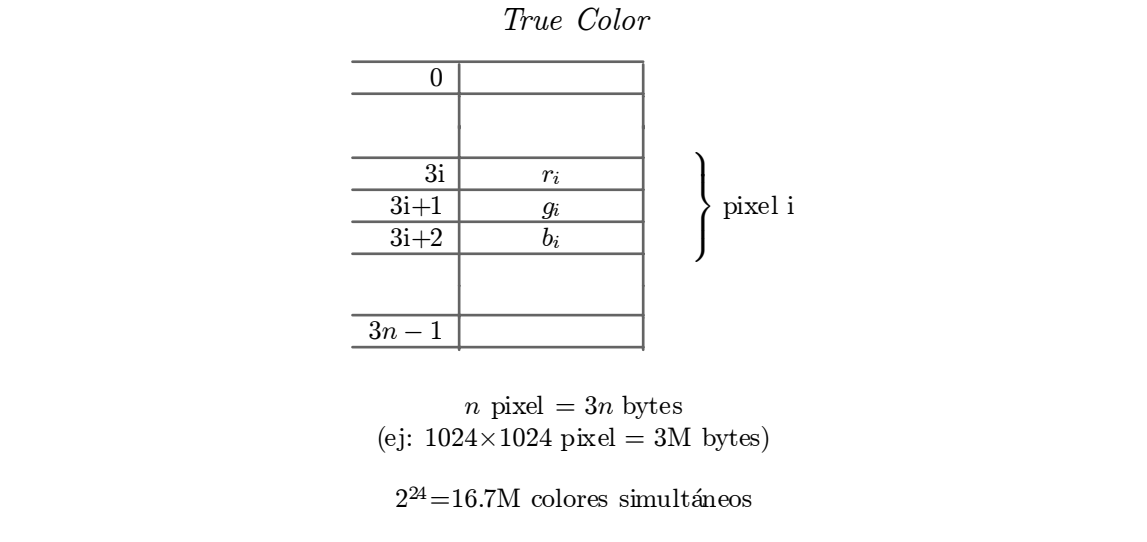


Figura 2.5 *Modelo de memoria* true color.

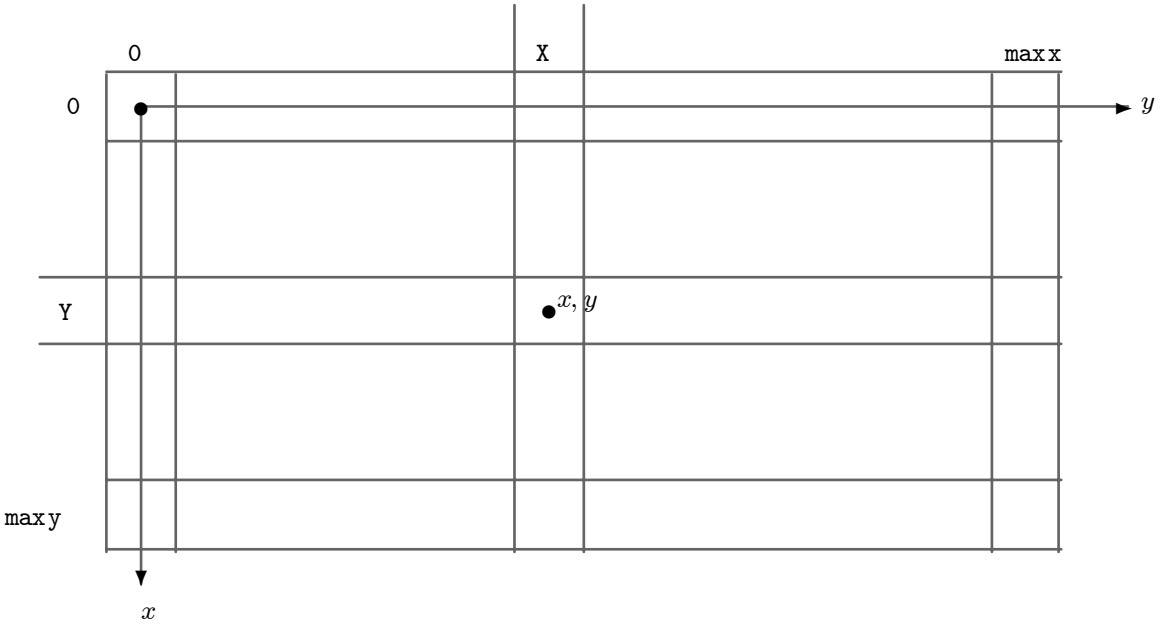


Figura 2.6 *Sistema de coordenadas físico junto al espacio de la escena.*

2.2.1 El sistema de coordenadas físico

El primer paso para conseguir una representación adecuada de las primitivas es caracterizar matemáticamente el medio que nos permite representarlas. Las primitivas gráficas independientes de dispositivo (en la "imagen" mental del usuario) normalmente se representan en un espacio Euclídeo de una determinada dimensión. En dichas condiciones un punto es una entidad matemática $p = (x, y)$, donde $(x, y) \in R^2$.

En el soporte aritmético de la computadora, dicha representación se efectúa con los tipos de datos provistos, que pueden ser números reales con punto flotante de simple o doble precisión. Este espacio se denomina *espacio de la escena* y es uno de los muchos espacios que se utilizarán para factorizar adecuadamente las diversas tareas de un sistema gráfico.

Por último, en el soporte gráfico del buffer de pantalla, un punto se representa con un pixel, y dicha representación se efectúa seteando una posición de memoria con un contenido dado. Este espacio se denomina *espacio de pantalla* y se direcciona a partir del sistema de coordenadas físico, cuyo origen es el vértice superior izquierdo. Es posible encontrar varias correspondencias posibles entre el sistema de coordenadas físico y un sistema de coordenadas arbitrario en el espacio de la escena. En la literatura normalmente se considera que un pixel es un "punto con extensión" en el espacio de la escena, y por lo tanto el origen de dicho espacio coincide con el vértice superior izquierdo del pixel `(0,0)` (ver por ejemplo [33, 40, 66, 84]). Desde nuestro punto de vista, esto no es del todo correcto, y parece más adecuado pensar que el origen del sistema de coordenadas de la escena esta en el *centro* del pixel `(0,0)` (ver Figura 2.6).

De esa manera, el espacio de pantalla es un espacio discreto y acotado `[0 .. maxx]`×`[0 .. maxy]`, con `maxx, maxy`$\in N$, el cual está en correspondencia con el espacio de la escena (euclídeo) $(x, y) \in R^2$ a través de las operaciones `X := round(`x`);` y `Y := round(`y`);` Por dicha razón es que la operación de llevar una primitiva del espacio de la escena al espacio de pantalla se denomina *discretización*.

2.2.2 Primitivas gráficas

Es muy difícil escoger un conjunto de primitivas gráficas que sea adecuado para la representación de todo tipo de entidades gráficas. Sin embargo, el siguiente subconjunto en la práctica resulta suficiente:

Puntos: Se especifican a partir de su localización y color. Su discretización es directa, como se mencionó más arriba.

Segmentos de recta: Son esenciales para la mayor parte de las entidades. Se especifican a partir de un par de puntos que representan sus extremos.

Circunferencias: En algunos casos representar entidades curvadas con segmentos poligonales puede ser indadecuado o costoso, por lo que en la práctica las circunferencias o círculos se adoptan como primitivas. Se especifican con la posición de su centro y su radio.

Polígonos: Son indispensables para representar entidades sólidas. Se representan a partir de la secuencia de puntos que determina la poligonal de su perímetro.

2.2.3 Especificaciones de una discretización

En el momento de escoger un método de discretización para una primitiva gráfica, es indispensable contar con criterios que permitan evaluar y comparar las ventajas y desventajas de las distintas alternativas. Entre todas las especificaciones posibles, podemos mencionar las siguientes:

Apariencia: Es la especificación más obvia, aunque no es fácil describirla en términos formales. Normalmente se espera que un segmento de recta tenga una "apariencia recta" más allá de que se hallan escogido los pixels matemáticamente más adecuados. Tampoco debe tener discontinuidades o puntos espúreos. Debe pasar por la discretización del primer y último punto del segmento. Debe ser uniforme, etc.

Simetría e invariancia geométrica: Esta especificación se refiere a que un método de discretización debe producir resultados equivalentes si se modifican algunas propiedades geométricas de la primitiva que se está discretizando. Por ejemplo, la discretización de un segmento no debe variar si dicho segmento se traslada a otra localización en el espacio, o si es rotado, etc.

Simplicidad y velocidad de cómputo: Como los métodos tradicionales de discretización de primitivas se desarrollaron hace tres décadas, en momentos en que las posibilidades del hardware y software eran muy limitadas, los resultados eran muy sensibles al uso de memoria u operaciones aritméticas complejas. Por lo tanto, los métodos tienden a no depender de estructuras complejas y a ser directamente implementables en hardware específico de baja complejidad.

2.2.4 Métodos de discretización

Dada una primitiva gráfica a discretizar, debemos encontrar los pixels que la representen de la manera más correcta posible. Para ello, lo más adecuado es caracterizar matemáticamente a dicha primitiva, de modo que su discretización pueda efectuarse en forma sencilla. Entre los diversos métodos que pueden plantearse destacamos los dos siguientes:

Evaluar su ecuación diferencial a diferencias finitas: Este método, denominado DDA (discrete diference analyzer) consiste en plantear la ecuación diferencial de la primitiva a discretizar, y luego evaluar dicha expresión a intervalos adecuados.

Análisis del error: Estos métodos fueron desarrollados por Bressenham [18] y se basan en analizar, dado un pixel que pertenece a la discretización de la primitiva, cuál es el próximo pixel que minimiza una determinada expresión que evalúa el error que comete la discretización.

2.3 Discretización de Segmentos de Rectas

El análisis de los métodos de discretización de rectas parte de considerar el comportamiento esperado en determinados casos particulares. Dichos casos surgen de suposiciones específicas que simplifican el problema, pero que al mismo tiempo se pueden generalizar a todos los demás casos por medio de simetrías. Dado un segmento de recta que va de (x_0, y_0) a (x_1, y_1), se supone que

- $\Delta x = (x_1 - x_0) \geq 0$,
- $\Delta y = (y_1 - y_0) \geq 0$, y
- $\Delta x \geq \Delta y$.

Esto equivale a trabajar en el "octavo" del espacio de pantalla sombreado en la Figura 2.7, donde el origen es el pixel que corresponde a la discretización del punto (x_0, y_0) y la zona sombreada a los lugares donde puede ubicarse el punto (x_1, y_1).

2.3.1 Segmentos de recta DDA

Como ya se mencionara, los métodos DDA evalúan la ecuación diferencial de la primitiva a intervalos finitos. En el caso particular de los segmentos de recta, la ecuación diferencial es

$$\frac{\partial y}{\partial x} = \frac{\Delta y}{\Delta x} = m.$$

El método busca encontrar una secuencia de $n + 1$ puntos tales que $(x_0, y_0) = (x^0, y^0), (x^1, y^1), \cdots, (x^n, y^n) = (x_1, y_1)$. La discretización de cada uno de ellos son los pixels de la discretización del segmento. Esta propiedad, si bien es trivial, es de gran importancia porque determina que la discretización de un segmento de recta es invariante frente a transformaciones afines. Esto significa que es equivalente transformar los extremos del segmento y discretizar el segmento transformado, o discretizar primero y transformar cada punto obtenido. Sin embargo, la primera alternativa es mucho más eficiente.

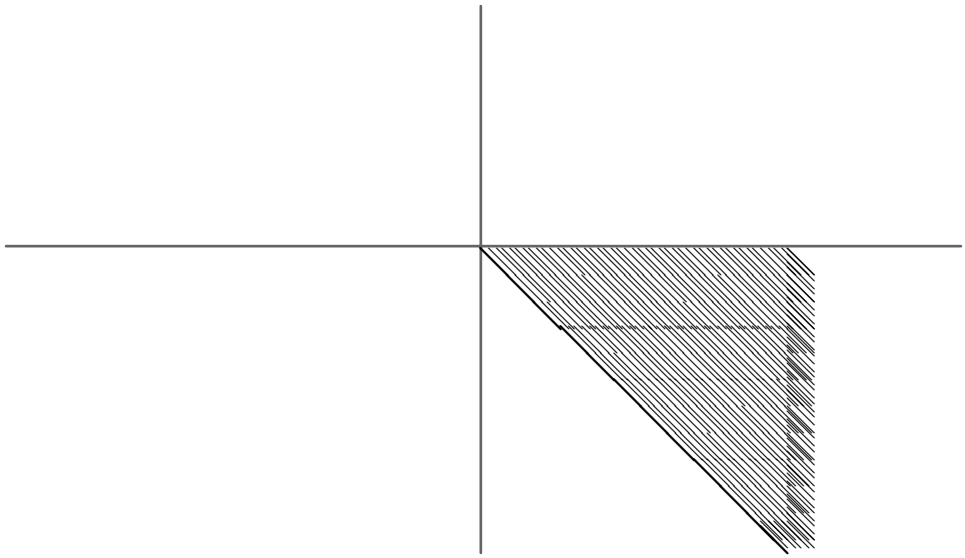


Figura 2.7 *Situación particular para derivar los algoritmos de discretización de segmentos de recta.*

Dada la ecuación diferencial y un incremento finito arbitrario $\varepsilon = \frac{1}{n}$, podemos pasar de un pixel dado de la secuencia al siguiente por medio de la expresión

$$\begin{cases} x^{k+1} &= x^k + \varepsilon \Delta x \\ y^{k+1} &= y^k + \varepsilon \Delta y \end{cases}$$

ε determina la "frecuencia" de muestreo del segmento. Un valor muy pequeño determina que muchas muestras producirán puntos que serán discretizados al mismo pixel. Por el contrario, un valor muy grande determinará que el segmento aparezca "punteado" en vez de ser continuo como corresponde. Un valor práctico es elegir $\varepsilon \Delta x = 1$ y por lo tanto $n = \Delta x$, es decir, la discretización tiene tantos pixels como longitud tiene el segmento en la variable que más varía (más uno, dado que la secuencia tiene $n + 1$ puntos). Al mismo tiempo es fácil ver que

$$\varepsilon \Delta y = \frac{\Delta y}{\Delta x} = m.$$

En la Figura 2.8 es posible ver el resultado de discretizar un segmento particular por el método DDA. Obsérvese que los puntos extremos (x_0, y_0) a (x_1, y_1) son en efecto puntos y por lo tanto están ubicados en cualquier lugar dentro del pixel que corresponde a su discretización. Un algoritmo sencillo que computa la discretización de un segmento de recta por el método DDA se muestra en la Figura 2.9. Obsérvese que es necesario computar las variables en punto flotante, y que además se requiere una división en punto flotante.

Para poder discretizar un segmento de recta en cualquiera de las posibilidades es necesario considerar las simetrías que se aplican. Si por ejemplo no se cumple que $\Delta y = (y_1 - y_0) \geq 0$, entonces hay que considerar pendientes negativas (simetria A), caso que el algoritmo de la Figura 2.9 realiza automáticamente. En cambio, si $\Delta x = (x_1 - x_0) \geq 0$, entonces es necesario *decrementar* a `x` en el ciclo e iterar mientras no sea *menor* que x_1 (simetría B). Por último, si no se cumple que $\Delta x \geq \Delta y$, entonces es necesario intercambiar los roles de las variables `x` e `y` (simetría C). Cualquier combinación de situaciones se puede resolver con combinaciones de simetrías (ver Figura 2.10).

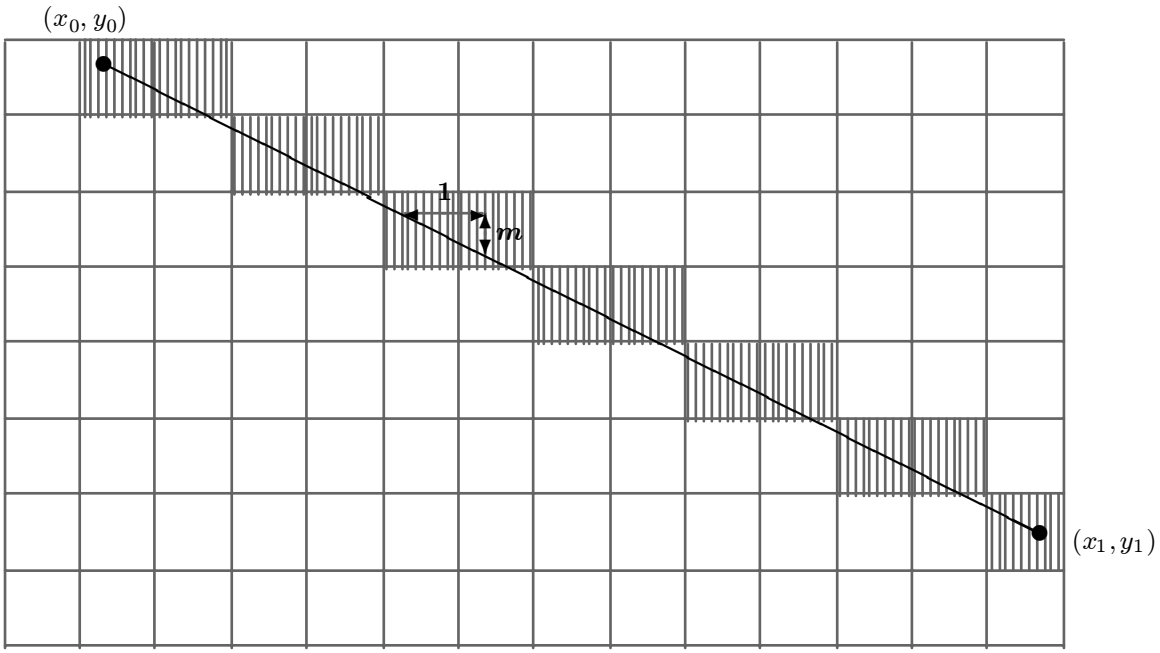


Figura 2.8 *La discretización de un segmento por DDA.*

```
procedure linea(x0,x1,y0,y1:real;col:integer);
var x,y,m:real;
begin
      m := (y1-y0)/(x1-x0);
      x := x0; y := y0;
      while x<=x1 do begin
         putpixel(round(x),round(y),col);
         x := x+1;
         y := y+m;
      end;
end;
```

Figura 2.9 *Algoritmo DDA para segmentos de recta.*

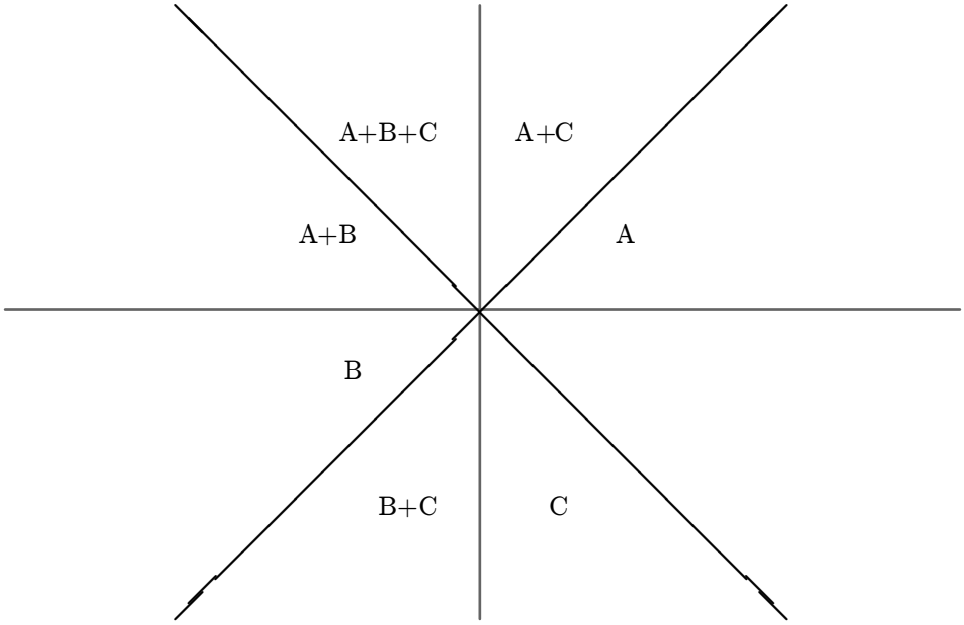


Figura 2.10 *Simetrías para los demás casos.*

2.3.2 Segmentos de rectas por Bressenham

En el algoritmo DDA para segmentos de recta es necesario computar sumas entre las variables en punto flotante, y además se requiere una división en punto flotante para computar la pendiente. El mérito del algoritmo que vamos a presentar consiste en que todas las operaciones se realizan en aritmética entera por medio de operaciones sencillas, y por lo tanto, su ejecución es más rápida y económica, y es de fácil implementación con hardware específico [18].

El punto de partida del análisis es el siguiente. Si la discretización de los puntos extremos del segmento debe pertenecer a la discretización del segmento, entonces es conveniente efectuar la llamada al algoritmo luego de discretizar los extremos. Esto significa que (x_0, y_0) y (x_1, y_1), y por lo tanto Δx y Δy son enteros. Luego, si p es un pixel que pertenece a la discretización del segmento, entonces en las condiciones particulares mencionadas más arriba, el proximo pixel solamente puede ser el ubicado a la derecha (**E** o "hacia el este"), o el ubicado en diagonal hacia la derecha y hacia abajo (**D** o "en diagonal", ver Figura 2.11).

La decisión de ir hacia el paso **E** o **D** se toma en función del error que se comete en cada caso. En este algoritmo se considera que el error es la distancia entre el centro del pixel elegido y el segmento de recta, medida en dirección del eje `Y` positivo del espacio de pantalla (es decir, hacia abajo). Si el error en p fuese cero, entonces al ir hacia **E** el error pasa a ser m (la pendiente del segmento), y en **D** el error pasa a ser $m - 1$ (ver Figura 2.12).

En general, si en p el error es e, la actualización del error es

- Paso **E**
  $e := e + m$

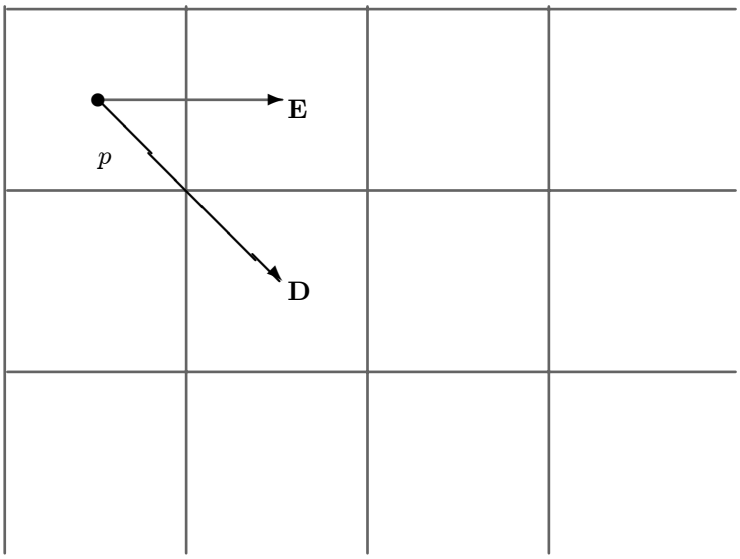


Figura 2.11 *Si p pertenece a la discretización, el próximo pixel solo puede ser* **E** *o* **D**.

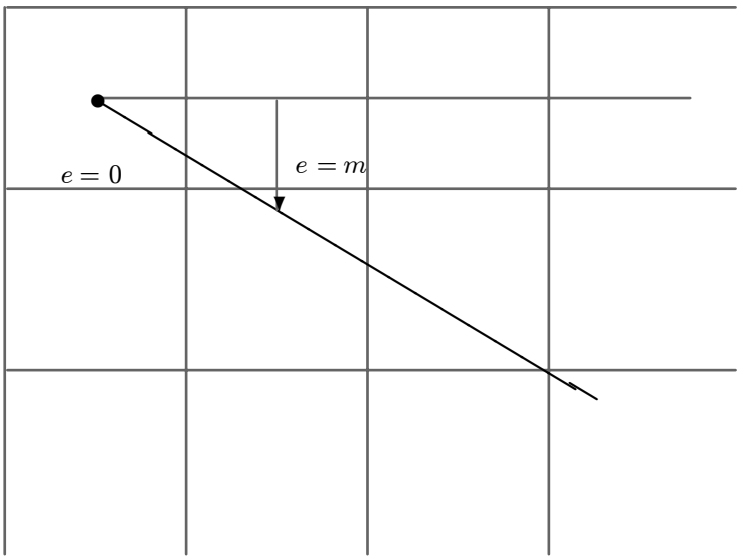


Figura 2.12 *Actualización del error.*

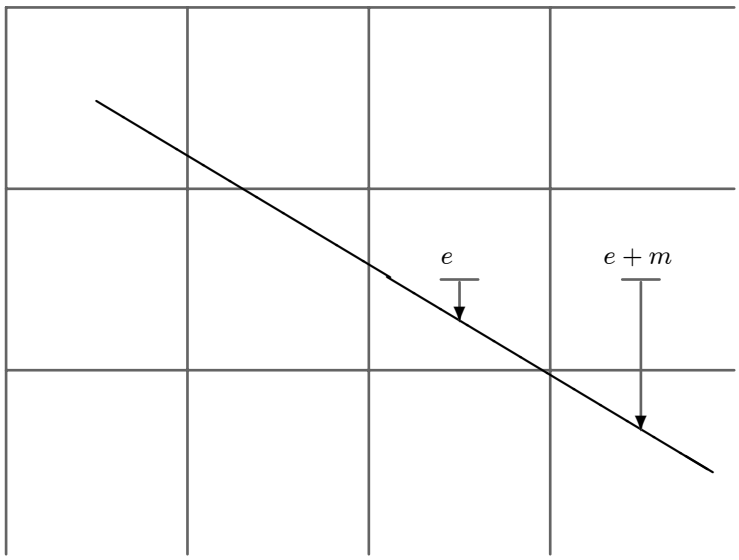


Figura 2.13 *Elección del próximo pixel.*

- Paso **D**
  $e := e + m - 1$

Por lo tanto, la elección del paso **E** o **D** depende de que el valor absoluto de $e+m$ sea o no menor que el valor absoluto de $e + m - 1$. Expresado de otra manera, sea e el error en un determinado pixel. Si $e + m > 0.5$ entonces el segmento de recta pasa más cerca del pixel **D**, y si no, pasa más cerca del pixel **E** (ver Figura 2.13).

Una de las economías de cómputo del método proviene de poder testear el error preguntando por cero. Es fácil ver que si se inicializa el error en

$$e_o = m - 0.5,$$

entonces en cada paso hay que chequear $e > 0$ para elegir **D**. La otra economía proviene de realizar manipulaciones algebraicas para efectuar un cómputo equivalente pero en aritmética entera. Como se testea el error por cero, multiplicar el error por una constante no afecta el resultado. Por lo tanto, multiplicamos el error por $2\Delta x$. A partir de dicho cambio, se constatan las siguientes igualdades:

- $e_o = 2\Delta x(\frac{\Delta y}{\Delta x} - 0.5) = 2\Delta y - \Delta x.$
- Paso **E**
  $e := e + 2\Delta y.$
- Paso **D**
  $e := e + 2(\Delta y - \Delta x).$

Todas las operaciones, entonces, se efectúan en aritmética entera.

```
procedure linea(x0,x1,y0,y1,col:integer);
var x,y,dx,dy,e,ix,iy:integer;
begin
     dx := x1 - x0; dy := y1 - y0;
     ix := 2*dx; iy := 2*dy;
      y := y0; e := iy - dx;
     for x := x0 to x1 do begin
        putpixel(x,y,col);
        e := e+iy;
        if e>0 then do begin
          y := y+1;
          e := e-ix;
        end;
     end;
end;
```

Figura 2.14 *Algoritmo de Bressenham para segmentos de recta.*

La implementación del algoritmo de Bressenham para segmentos de recta se muestra en la Figura 2.14. Teniendo en cuenta que los productos por 2 en aritmética entera se efectúan con un desplazamiento a izquierda, es posible observar que el mismo utiliza operaciones elementales e implementables con hardware específico muy sencillo.

2.4 Discretización de Circunferencias

Como en el caso de los segmentos de recta, en la discretización de circunferencias o círculos trabajaremos solamente en una parte de la misma, y obteniendo las demás partes por simetría. Supongamos que la circunferencia a discretizar está centrada en el origen, y que $p = (x, y)$ es un pixel de su discretización tal que

- $x \geq 0$,
- $y \geq 0$,
- $x \geq y$.

Dicha situación describe un octavo de la circunferencia, pero las partes faltantes pueden encontrarse por medio de las simetrías mostradas en la Figura 2.15.

2.4.1 Discretización de circunferencias por DDA

Sea una circunferencia de radio r centrada en el origen y sea $p = (x, y)$ un punto que pertenece a la misma. Entonces, la ecuación diferencial de la circunferencia en dicho punto (ver Figura 2.16) es

$$\frac{\partial y}{\partial x} = -\frac{x}{y}.$$

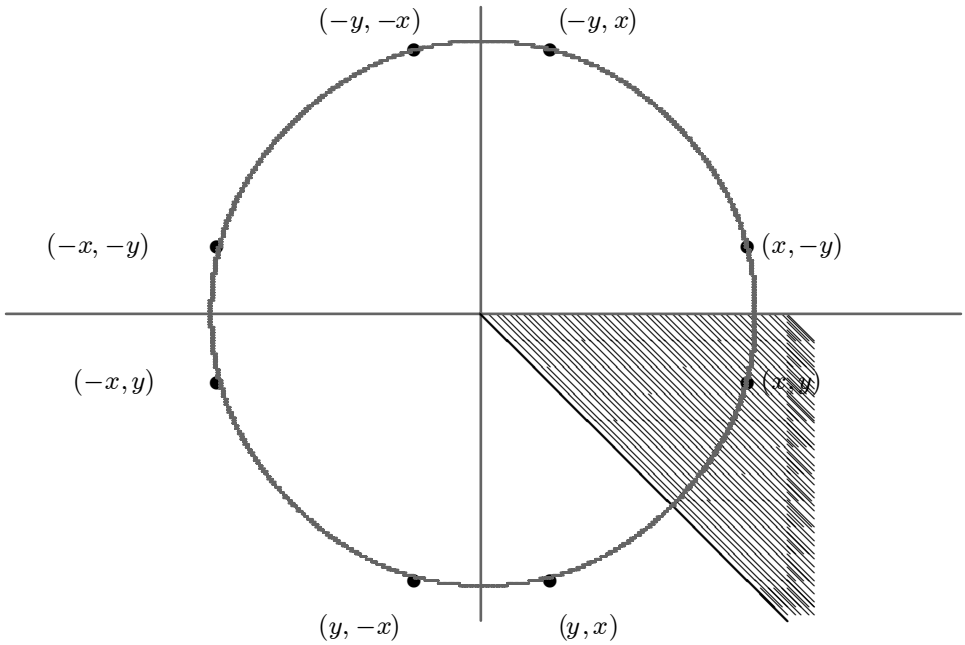


Figura 2.15 *Simetrías para la discretización de circunferencias.*

La evaluación de la ecuación diferencial por diferencias finitas, como en el caso de los segmentos de recta, consiste en encontrar una secuencia de valores, de modo que dado un pixel de la discretización (x_k, y_k), el próximo pixel se encuentra con

$$\begin{cases} x_{k+1} &= x_k - \varepsilon y_k \\ y_{k+1} &= y_k + \varepsilon x_k \end{cases} \tag{2.1}$$

ε determina la "frecuencia" de muestreo de la circunferencia. Valores pequeños determinan un cómputo redundante, y valores muy grandes determinan un cubrimiento desparejo. Un valor práctico es elegir $\varepsilon x = 1$ y por lo tanto $\varepsilon y = \dfrac{y}{x}$ (ver Figura 2.17). El primer pixel de la secuencia es $p_0 = (r, 0)$, el cual pertenece a la discretización de la circunferencia y además se computa en forma directa.

Es importante tener en cuenta que el sistema de ecuaciones 2.1 puede representarse como una transformación:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \varepsilon \\ -\varepsilon & 1 \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \end{bmatrix},$$

la cual tiene determinante $1 + \varepsilon^2$. Esto significa que la discretización es levemente espiralada hacia afuera. Una forma de solucionar este problema es utilizar una recurrencia cuya transformación sea

$$\begin{bmatrix} 1 & \varepsilon \\ -\varepsilon & 1 - \varepsilon^2 \end{bmatrix}.$$

Esta transformación representa en realidad una elipse y no una circunferencia, pero la excentricidad de la misma es del orden de ε^2 y por lo tanto la diferencia es más reducida. Utilizar dicha expresión

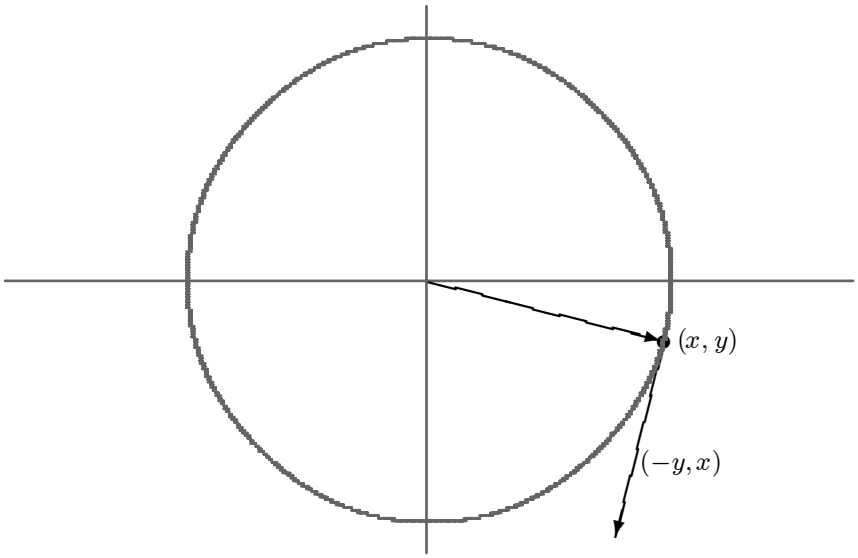


Figura 2.16 *Interpretación geométrica de la ecuación diferencial de la circunferencia centrada en el origen.*

```
procedure circ(radio:real;col:integer);
var x,y:integer;
    rx:real;
begin
    rx:=radio;
    x:=round(rx); y:=0;
    while (y<x) do begin
       putpixel(x,y,col);  putpixel(y,x,col);
       putpixel(-x,y,col); putpixel(-y,x,col);
       putpixel(x,-y,col); putpixel(y,-x,col);
       putpixel(-x,-y,col);putpixel(-y,-x,col);
       rx:=rx-y/rx;
       x:=round(rx); y:=y+1;
    end;
end;
```

Figura 2.17 *Algoritmo DDA para discretización de circunferencias.*

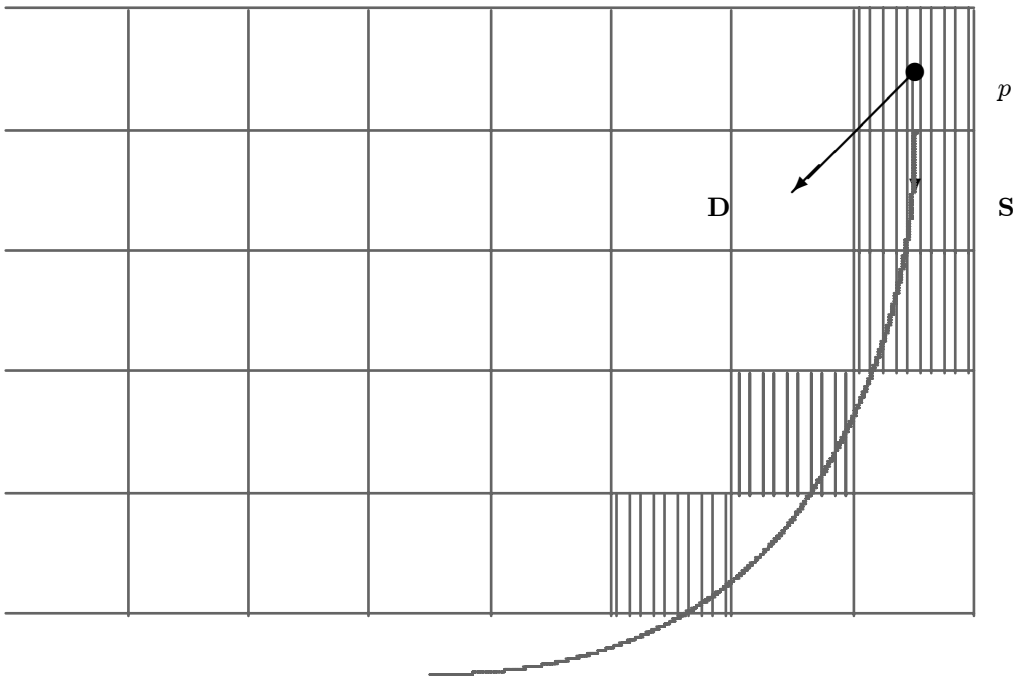


Figura 2.18 *Si p pertenece a la discretización, el próximo* pixel *solo puede ser* **S** *o* **D**.

altera la evaluación de y_{k+1}:

$$\begin{aligned} y_{k+1} &= y_k(1-\varepsilon^2) - \varepsilon x_k \\ y_{k+1} &= y_k - \varepsilon(\varepsilon y_k + x_k) \\ y_{k+1} &= y_k - \varepsilon x_{k+1}. \end{aligned}$$

Por lo tanto, la recurrencia para computar la discretización de una circunferencia centrada en el origen con DDA queda modificada a

$$\begin{cases} x_{k+1} = x_k - \varepsilon y_k \\ y_{k+1} = y_k + \varepsilon x_{k+1} \end{cases}$$

De todas maneras, la desventaja más importante de este algoritmo es que debe realizar una división en punto flotante para cada paso.

2.4.2 Discretización de Bressenham para circunferencias

Como en el caso planteado para segmentos de recta, este método se basa en analizar el error entre la verdadera circunferencia y su discretización. Si p pertenece a la discretización, el próximo pixel de la discretización solo puede ser **S** ("ir hacia el sur") o **D** ("ir en diagonal", ver Figura 2.18).

El error (en este caso la distancia al cuadrado) de un pixel $p = (x, y)$ a una circunferencia de radio r con centro en $(0, 0)$ es:

$$e = x^2 + y^2 - r^2.$$

```
procedure circ(radio:real;col:integer);
var x,y,e:integer;
begin
    x:=radio; y:=0;
    e:=0;
    while (y<x) do begin
       putpixel(x,y,col);  putpixel(y,x,col);
       putpixel(-x,y,col); putpixel(-y,x,col);
       putpixel(x,-y,col); putpixel(y,-x,col);
       putpixel(-x,-y,col);putpixel(-y,-x,col);
       e:=e+2*y+1;
       y:=y+1;
       if ((2*e)>(2*x-1)) then do begin
          x:=x-1;
          e:=e-2*x+1;
       end;
    end;
end;
```

Figura 2.19 *Algoritmo de Bressenham para circunferencias.*

Si elegimos el paso **S** entonces el próximo pixel es $p_S = (x, y+1)$, y el error pasa a ser

$$e_S = (x)^2 + (y+1)^2 - r^2 = e + 2y + 1.$$

Si elegimos el paso **D** entonces el próximo pixel es $p_D = (x-1, y+1)$, y el error pasa a ser

$$e_D = (x-1)^2 + (y+1)^2 - r^2 = e_S - 2x + 1.$$

La elección de un paso **S** o **D** dependerá de cuál error tiene menor módulo:

$$\text{Si } |e + 2y + 1| > |e + 2y + 1 - 2x + 1| \text{ entonces } \mathbf{D}.$$

Teniendo en cuenta que tanto en **D** como en **S** se incrementa y, entonces se actualiza el error a e_S antes de la comparación.

$$\text{Si } |e| > |e - 2x + 1| \text{ entonces } \mathbf{D}.$$

Por último, teniendo en cuenta que $-2x + 1$ es siempre negativo (recordar que $x > 0, x > y$), entonces:

$$\text{Si } 2e > 2x - 1 \text{ entonces } \mathbf{D}.$$

De esa manera, el algoritmo queda planteado exclusivamente con operaciones enteras y de aritmética sencilla (ver Figura 2.19.)

2.5 Discretización de Polígonos

El objetivo de la discretización de polígonos es encontrar el conjunto de pixels que determinan el área sólida que dicho polígono cubre en la pantalla. Si bien existen diversos métodos, aquí presentaremos el más económico y difundido, que se basa en encontrar la intersección de todos los lados del polígono con cada línea de barrido (a y constante), por lo que el método se denomina *conversión scan* del polígono. Este método es de gran importancia porque se generaliza a una

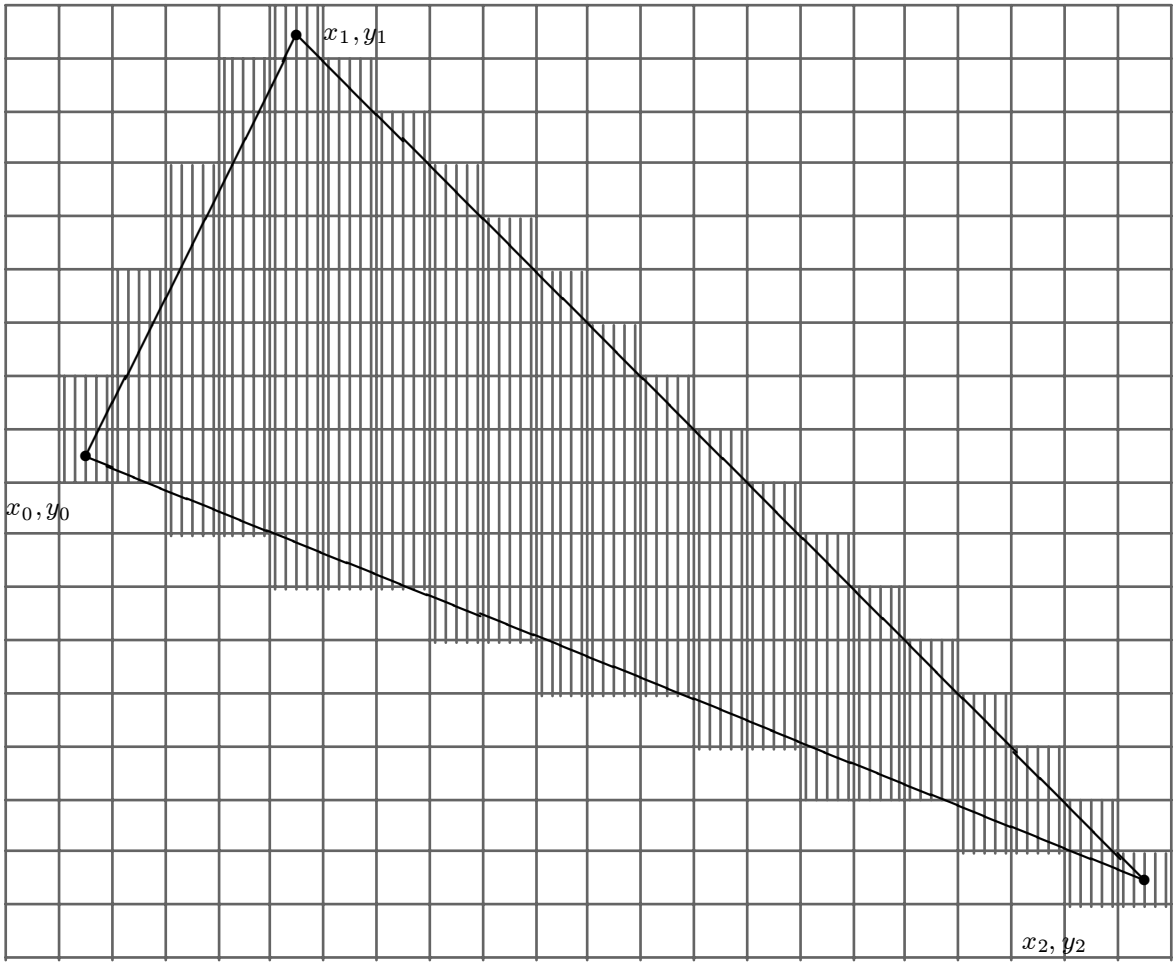


Figura 2.20 *Conversión scan de un polígono.*

clase de algoritmos denominados *scan-line* para resolver determinados problemas de iluminación y sombreado en tres dimensiones (ver Capítulo 7).

Todo polígono plano puede descomponerse en triángulos. Por lo tanto el triángulo será la base del análisis de la conversión scan de polígonos en general. Para computarla es necesario dimensionar dos arreglos auxiliares de enteros `minx, maxx` que para cada línea de barrido almacenarán el menor y mayor `x` respectivamente (ver Figura 2.20).

1. Inicializar `minx` a infinito y `maxx` a menos infinito.
2. Discretizar cada arista del triángulo (con DDA o Bressenham) reemplazando la sentencia
   `putpixel(x,y,col);`
   por
   ```
   if x>maxx[y] then maxx[y]:=x;
   if x<minx[y] then minx[y]:=x;
   ```
3. Para cada `y` activo graficar una línea de `minx[y]` a `maxx[y]`.

2.6 Ejercicios

1. Implementar los algoritmos de discretización de segmentos de recta por DDA y Bressenham con todas las simetrías.

2. Implementar los algoritmos de discretización de circunferencias por DDA y Bressenham. Modificar el DDA para minimizar el error. Modificar ambos algoritmos para que reciban las coordenadas del centro.

3. Implementar el algoritmo de conversión scan para triángulos. ¿Es necesario modificarlo sustancialmente para cualquier polígono convexo?

2.7 Bibliografía recomendada

La mayor parte de los trabajos que dieron origen a los algoritmos presentados en este Capítulo pertenecen a publicaciones virtualmente inaccesibles (excepto en colecciones de *reprints* de trabajos clásicos, ver por ejemplo [36]).

Una descripción interesante de los dispositivos gráficos puede encontrarse en los Capítulos 2 y 19 del libro de Newman y Sproull [66], y en el Capítulo 7 del libro de Giloi [40]. Una descripción más moderna puede encontrarse en el Capítulo 4 del libro de Foley et. al. [33]. Los interesados en conocer arquitecturas avanzadas para sistemas gráficos pueden consultar además el Capítulo 18 del mismo libro. La discusión relacionada con la correspondencia entre el espacio de los objetos y el espacio de pantalla está inspirada en las conclusiones de un trabajo de Jim Blinn [12].

La descripción más directa de los algoritmos de discretización de segmentos y circunferencias puede encontrarse en el Capítulo 2 del libro de Newman y Sproull. Aunque difieren de la presentada aquí por adoptarse distintas convenciones, los desarrollos son similares. En el Capítulo 3 del libro de Foley et. al. es posible encontrar una detallada descripción de varios algoritmos que generalizan los presentados en este Capítulo (por ejemplo, la discretización de segmentos de un determinado grosor, o el "llenado" de polígonos con patrones repetitivos). La conversión scan de polígonos es tratada en el Capítulo 16 del libro de Newman y Sproull.

3 Computación Gráfica en Dos Dimensiones

3.1 Estructuras de Datos y Primitivas

En lo que sigue de este libro suponemos que existe un procedimiento `putpixel(x,y,c)` y que además es posible determinar el rango de las coordenadas de pantalla `maxx, maxy` y el rango de definiciones de colores `numcol`. De esa manera podemos definir estructuras de datos para manejar los elementos de pantalla:

```
type xres = 0..maxx;
     yres = 0..maxy;
      col = 0..numcol;

type pixel = record x:xres;
                    y:yres;
                    c:col
             end;
```

Los algoritmos de discretización de primitivas nos permiten representar los distintos elementos gráficos con cierta independencia de dispositivo. Si elaboramos un sistema en el cual todas las diversas entidades gráficas están compuestas solamente por dichas primitivas, entonces el mismo puede transportarse de un dispositivo de raster a otro de mayor o menor resolución utilizando la discretización de primitivas ya mostrada, o también puede portarse a un dispositivo de vectores utilizando directamente las primitivas que éste provee.

Por lo tanto es necesario definir un conjunto de estructuras de datos para poder representar las primitivas en el espacio de la escena, y estructurar todas las entidades gráficas en términos de dicha representación. Por ejemplo podemos definir tipos para las siguientes entidades:

```
type punto = record x,y:real;
                    c:col
             end;

type linea = record p0,p1:punto;
                      c:col
             end;

type circulo = record centro:punto;
                      radio:real;
                          c:col
               end;

type cuadrilatero = record p0,p1,p2,p3:punto
                                     c:col
                    end;
```

Los procedimientos para graficar las primitivas gráficas pueden implementarse de muchas maneras. Una opción es la siguiente:

```
procedure graficar_punto(p:punto);
 var pl:pixel;
 begin
  pl.x:=round(p.x);
  pl.y:=round(p.y);
  putpixel(pl.x,pl.y,p.c);
 end;
end;

procedure graficar_linea(l:linea);
 var p0,p1:pixel;
 begin
  p0.x:=round(l.p0.x);
  p0.y:=round(l.p0.y);
  p1.x:=round(l.p1.x);
  p1.y:=round(l.p1.y);
  linea(p0.x,p1.x,p0.y,p1.y,l.c);    %por ejemplo, Bressenham
 end;
end;
```

Una ventaja de manejar las entidades de esta manera es que la estructura de tipos nos protege de cometer errores en la llamada de los procedimientos. Por ejemplo, podemos olvidar el significado de los parámetros de llamada del procedimiento `linea` (¿Era (x0,x1,y0,y1) o (x0,y0,x1,y1)?). Pero el procedimiento `graficar_linea` recibe un segmento de línea especificado por dos puntos, y entonces no hay error posible. Por otra parte, si el punto o la linea se van fuera del área graficable, la estructura de tipos nos avisa en tiempo de ejecución sin que se genere un error por invadir áreas de memoria no asignadas. La forma de manejar estos problemas sin producir un error de tipo será expuesta en la sección 3.4.

3.2 Transformaciones y Coordenadas Homogeneas

Una de las técnicas más poderosas de modelado de entidades gráficas consiste en descomponer ingeniosamente las mismas en términos de primitivas. Por ejemplo, la escena mostrada en la Figura 3.1 está compuesta exclusivamente por cuadrados, cada uno ubicado en un lugar específico y con una escala (tamaño) adecuada.

Sin embargo, describir la escena como *un conjunto* de instancias de cuadrados es quedarse en la superficie del fenómeno. Lo verdaderamente importante es que hay una descomposición jerárquica que es más rica y versátil. Podemos pensar que la escena está compuesta por un suelo y dos instancias de casa. Cada casa está compuesta por un techo, un frente, una puerta y una ventana, etc. De esa manera, las entidades gráficas son *jerarquías* de estructuras, que se definen como la composición de instancias de estructuras más simples, hasta terminar en el cuadrado. Es necesario, entonces, definir una única vez cada estructura.

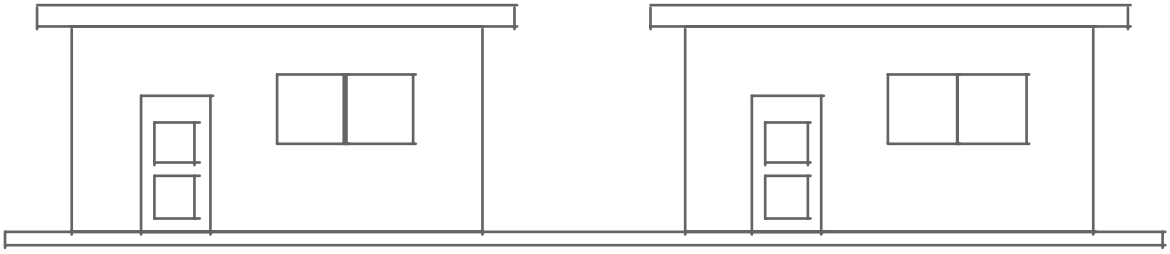


Figura 3.1 *Una escena compuesta exclusivamente de instancias de cuadrados.*

3.2.1 Transformaciones afines

Para utilizar una entidad varias veces, es decir, para que ocurran varias instancias de una entidad, es necesario hacerla "aparecer" varias veces con distintas transformaciones. Por ejemplo, en la Figura 3.1 aparecen dos instancias de casa, cada una con una traslación distinta. Cada elemento de la casa está definido también con transformaciones de otros elementos, las cuales deben *concatenarse* con la transformación correspondiente a cada casa para obtener el resultado final. Por lo tanto, las transformaciones desempeñan un papel decisivo en los modelos de la Computación Gráfica, en particular las transformaciones rígidas o lineales.

Como es sabido, la traslación, rotación (alrededor del origen) y escalamiento, conforman una *base funcional* para las transformaciones rígidas en un espacio lineal. Es decir, toda transformación afín o lineal puede ponerse en términos de una aplicación de estas tres operaciones.

Utilizaremos por un momento la notación $p = \begin{bmatrix} x \\ y \end{bmatrix}$ para referirnos a un punto en un espacio de dos dimensiones (ver Apéndices A y B). El resultado de aplicar a p una transformación de escalamiento, por ejemplo, es un punto $p' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ tal que

$$x' = e_x x, \qquad y' = e_y y.$$

Es más conveniente una notación matricial:

$$p' = E \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} e_x & 0 \\ 0 & e_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Del mismo modo puede representarse para rotar un ángulo θ alrededor del origen:

$$p' = R \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Es un inconveniente que no pueda representarse la traslación como preproducto por una matriz:

$$x' = t_x + x, \qquad y' = t_y + y.$$

De esa manera se pierden dos propiedades convenientes:

1. Toda transformación de un punto es su preproducto por una matriz cuadrada.
2. La *concatenación* de transformaciones es el preproducto de las matrices respectivas.

3.2.2 Coordenadas homogéneas

Es necesario tener en cuenta que los espacios lineales (Euclídeos) son conjuntos de valores vectoriales cerrados bajo suma y multiplicación escalar (ver Apéndice B). Por lo tanto, los puntos en estos espacios no tienen una representación que los distinga. Una de las confusiones más nocivas en la Computación Gráfica se debe a que la representación de un *punto* y un *vector* en un espacio Euclídeo de n dimensiones puede hacerse con una n-upla de reales, en particular, representando a un punto p con un vector que va del origen a p. Uno de los problemas de esta confusión es que, si bien la suma de vectores es independiente del sistema de coordenadas (al estar los vectores "libres" en el espacio), la "suma" de puntos pasa a depender de la elección particular del origen (ver Figura 3.2).

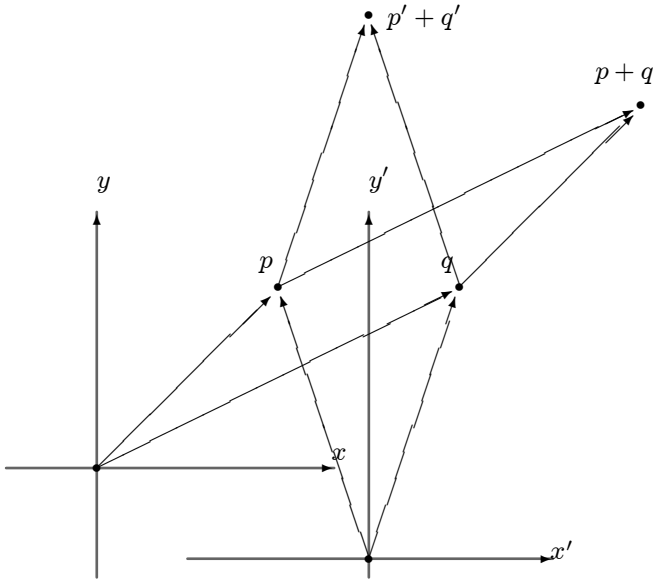


Figura 3.2 *La "suma" de puntos está mal definida en un espacio vectorial dado que depende de la elección particular de un sistema de coordenadas.*

Sin embargo, en un espacio vectorial de n dimensiones no afín, es posible encontrar subespacios afines de $n-1$ dimensiones, es decir, subespacios en los cuales la combinación afín de elementos está bien definida. Por ejemplo, en un plano, dados dos puntos p y q, la recta que pasa por dichos puntos es un espacio afin unidimensional que contiene todas las combinaciones afines de p y q. La proyección (intersección) de la suma vectorial de p y q con dicha recta es un punto que está a mitad de camino entre dichos puntos, independientemente del sistema de coordenadas elegido (ver Figura 3.3). Si realizamos la suma vectorial $p + 9q$ y proyectamos, se obtiene un punto que esta a un 10% de distancia de q y a 90% de p sobre la recta.

Todos los puntos de la recta, como dijimos, pueden representarse como combinación afín de p y q, es decir:

$$r = sp + tq,$$

donde $s + t = 1$. Si además tanto t como s son no negativos, entonces la combinación afín se denomina *convexa* porque el punto resultante pertenece al segmento de recta que va de p a q. Una

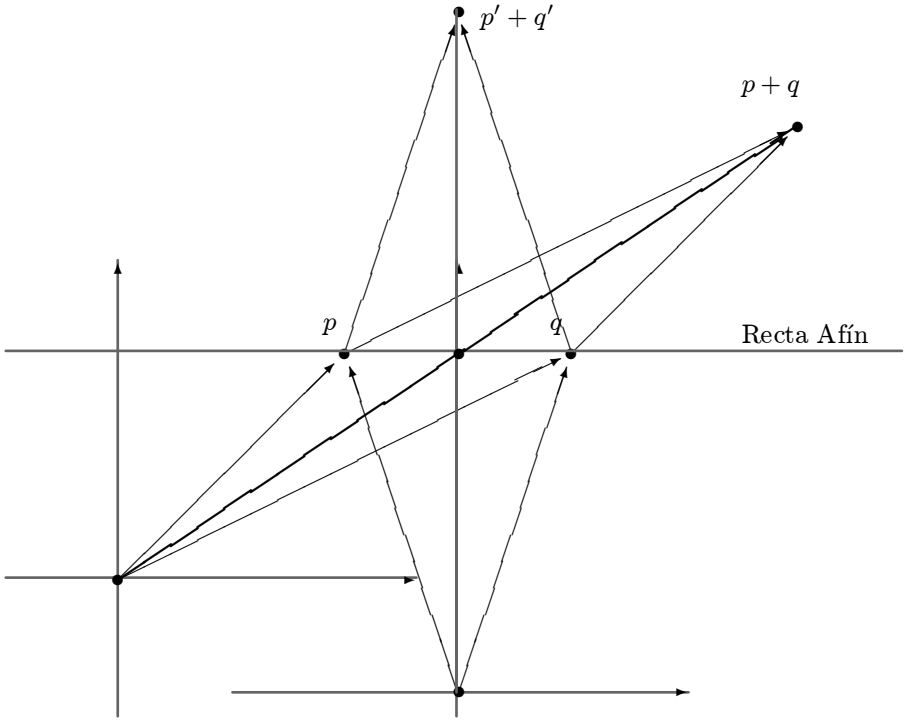


Figura 3.3 *La combinación afín de puntos está bien definida en un subespacio afín del espacio vectorial.*

inspección más detallada nos permite expresar el resultado de una combinación afín del siguiente modo:

$$r = sp + tq = (1 - t)p + tq = p + t(q - p),$$

la cual es una expresión conocida para representar un segmento en forma paramétrica.

Recíprocamente, todo espacio E de n dimensiones puede hacerse afín dentro de un espacio vectorial V de $n + 1$ dimensiones por medio de un procedimiento denominado *homogenización*. Para ello se considera que E está "inmerso" dentro de V como un hiperplano para un valor no trivial (distinto de cero) de una nueva variable h llamada variable de homogenización. En la Figura 3.4 se muestra la homogenización del espacio R^2 como (hiper)plano $h = 1$ de un espacio vectorial R^3. Todo elemento del espacio homogéneo debe considerarse proyectado sobre el plano $h = 1$. De esa manera, la suma de puntos se transforma en una combinación afín.

Es importante observar que en un espacio afín de estas características la representación de un punto pasa a ser

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

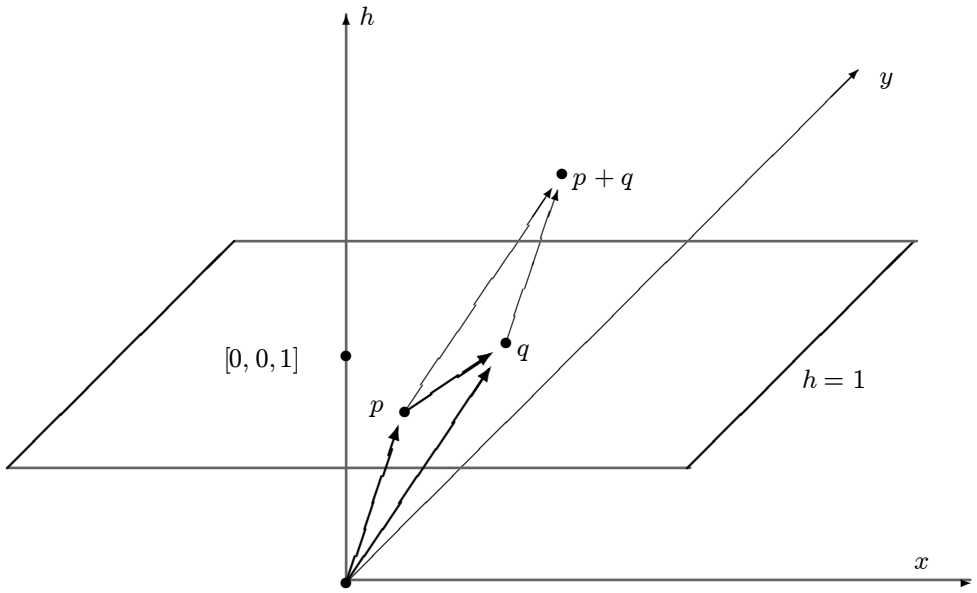


Figura 3.4 *Homogenización del espacio R^2.*

y la representación de un vector (como resta de puntos) es

$$v = \begin{bmatrix} a \\ b \\ 0 \end{bmatrix}.$$

Por lo tanto, se supera la confusión entre puntos y vectores. Tambien es importante destacar la equivalencia entre puntos para cualquier valor no negativo de h, dado que todos los puntos que se proyectan al mismo lugar en el plano $h = 1$ son equivalentes:

$$\begin{bmatrix} xh_1 \\ yh_1 \\ h_1 \end{bmatrix} \sim \begin{bmatrix} xh_2 \\ yh_2 \\ h_2 \end{bmatrix} \sim \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Para pasar del espacio homogéneo $n+1$-dimensional al lineal n-dimensional es necesario dividir las primeras n componentes por la última:

$$\begin{bmatrix} x \\ y \\ h \end{bmatrix} \rightsquigarrow \begin{bmatrix} \frac{x}{h} \\ \\ \frac{y}{h} \end{bmatrix}.$$

Este costo adicional de dos cocientes, sin embargo, permite el beneficio de una representación uniforme de las transformaciones. Además, en 3D es necesaria la división para realizar la transformación perspectiva, por lo que el costo en realidad no existe (ver Sección 6.3).

3.2.3 Transformaciones revisitadas

Podemos solucionar el inconveniente mencionado en la subsección 3.2.1 por medio del uso de coordendadas homogéneas. De esa manera, una matriz de escalamiento es:

$$p' = E \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} e_x & 0 & 0 \\ 0 & e_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Del mismo modo puede representarse la rotación

$$p' = R \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

y también la traslación $x' = t_x + x, y' = t_y + y$:

$$p' = T \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Observar que la concatenación no conmuta en general, es decir, es importante el orden en el que se aplican (premultiplican) cada una de las transformaciones.

3.3 Representación Estructurada

Podemos ahora retornar a nuestro ejemplo de escena mostrado en la Figura 3.1. La representación de objetos se realiza por medio de una estructuración jerárquica en forma de grafo, de modo tal que cada objeto se compone de una definición y una transformación "de instancia" (es decir, dónde debe aparecer el objeto definido), y cada definición es una colección de objetos. En la Figura 3.5 es posible ver parte de la estructura necesaria para definir la escena de la Figura 3.1.

Es importante tener en cuenta que si bien las estructuras son recursivas (un objeto es un par definición-transformación, y una definición es una colección de objetos), las mismas terminan indefectiblemente en objetos primitivos, en este caso, cuadrados. Para graficar la escena, entonces, es necesario recorrer la estructura `escena`. Para cada objeto apuntado por la misma, se guarda la transformación en una pila y se recorre recursivamente. La recursión termina cuando la definición no es una estructura de objetos sino un `cuad`, en cuyo caso se transforman los cuatro vértices del mismo por el preproducto de todas las matrices apiladas y se grafican los segmentos de recta que los unen.

En la Figura 3.6 se observan las declaraciones de tipos y los procedimientos que implementan estos algoritmos. Es importante notar que la definición de un objeto se realiza por medio de registros variantes, de manera de poder guiar la recursión.

Una forma de hacer este cómputo más eficiente es utilizar una matriz de "transformación corriente", a la cual se multiplica cada una de las matrices que se van apilando. Cuando se agota una rama de la recursión es necesario desapilar la última transformación y recalcular la transformación corriente. El trozo de código de la Figura 3.6 ilustra una implementación posible de estas ideas, mientras que en la Figura 3.7 se muestra parte de la inicialización de la estructura de información para representar una escena como la de la Figura 3.8.

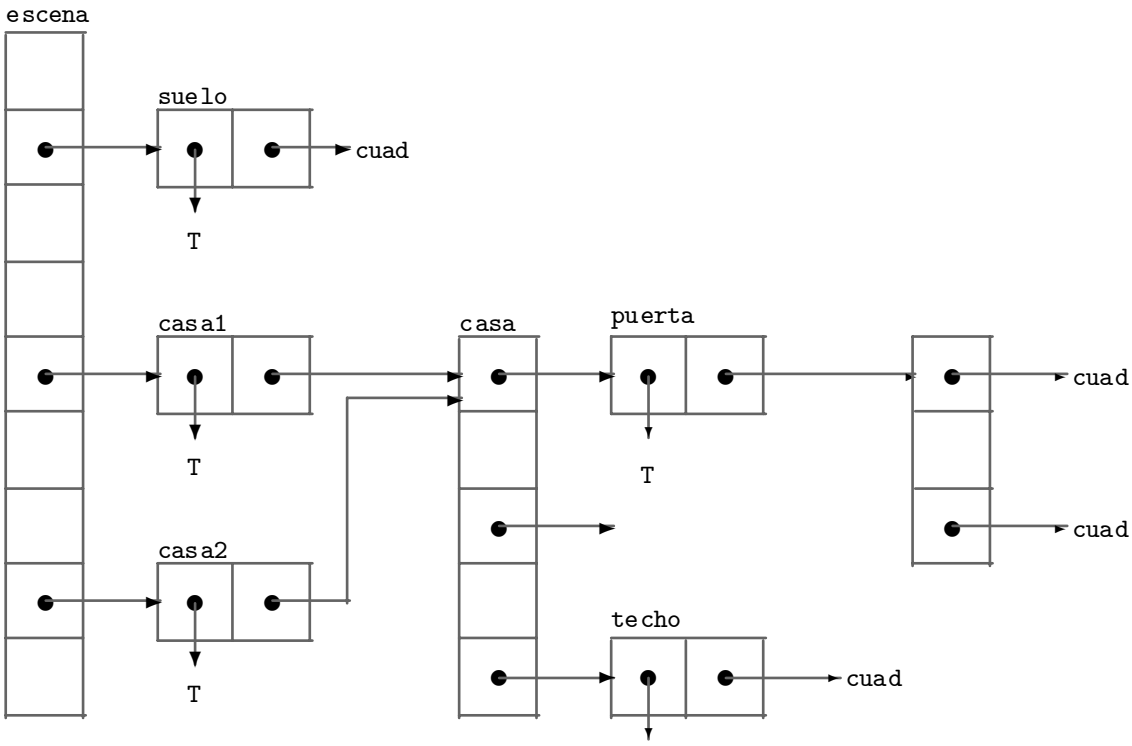


Figura 3.5 *Estructuras de datos que representan la escena de la Figura 3.1.*

Es muy importante destacar la flexibilidad que tiene este esquema para el manejo de las escenas. Modificar la transformación de una de las instancias de `casa` modifica *todo* lo que se grafica para dicha casa, y modificar la transformación que define uno de los elementos de la casa modifica cómo se grafica el mismo en *todas* las casas (ver Figura 3.8).

Es necesario tener en cuenta que las matrices homogéneas de transformación normalmente son inversibles en forma muy sencilla (la inversa de la escala es escalar por la inversa, la inversa de la rotación es rotar por el ángulo inverso, y la inversa de la traslación es trasladar por la distancia invertida). Esto hace que la actualización de la transformación corriente pueda hacerse posmultiplicando por la inversa de la matriz que se saca de la pila.

3.4 Windowing y Clipping

La representación estructurada de entidades gráficas permite una cierta abstracción del dispositivo gráfico de salida, independizándonos del tipo de primitivas que soporta y de la tecnología del mismo. Sin embargo, aún queda por resolver el problema del rango de valores de los pixels representables. Puede suceder que los gráficos que se acomodaban bien a la salida en una situación particular, al cambiar de resolución se vean de manera inadecuada. Esto puede empeorar aún más si nuestra aplicación produce una salida gráfica sin utilizar la pantalla completa, por ejemplo en un sistema cuya interfase trabaja por medio de ventanas.

```
type entidades = (cuads,defis);
          cuad = array [0..3] or punto;
        transf = array [0..2] of array [0..2] of real;
          defi = array[0..9] of ^objeto;
        objeto = record   t:^transf;
                  case tipo:entidades of
                        defis:(d:^defi);
                        cuads:(c:^cuad);
                  end;

procedure graf_cuad(o:objeto; at:transf);
var t,t1:transf;
    cu:cuad;
begin
  t:=o.t^;                cu:=o.c^;
  prod_matriz(t,at,t1);
  transformar(cu,t1);
  linea(cu[0],cu[1]);     linea(cu[1],cu[2]);
  linea(cu[2],cu[3]);     linea(cu[3],cu[0]);
end;

procedure graf_obj(o:objeto;at:transf);
var t,t1:transf;
    d:defi;
    i:integer;
begin
  case o.tipo of
   cuads : graf_cuad(o,at);
   defis : begin
             t:=o.t^;       d:=o.d^;
             while d[i]<>nil do begin
               prod_matriz(t,at,t1);
               graf_obj(d[i]^,t1);
             end;
           end;
   end;
end;
```

Figura 3.6 *Estructuras de datos y procedimientos para recorrer una representación estructurada de escenas.*

```
procedure graficar;
var   cu:cuad;
      t1, ... :transf;
      o1, ... :objeto;
      casa,sitio,puerta,escena:defi;
begin
 {inicializacion del cuadrado}
 cu[0].x := -1;  cu[0].y := -1;  cu[0].w := 1;
 ...
 {inicializacion objeto 1}
 o1.t:=@t1;      o1.c:=@cu;
 t1[0][0]:=2.2;  t1[0][1]:=0;    t1[0][2]:=0;
 ...
 {inicializacion objeto 2}
 o2.t:=@t2;      o2.c:=@cu;
 t2[0][0]:=2.2;  t2[0][1]:=0;    t2[0][2]:=0;
 ...
 {definicion de sitio como conteniendo objetos 1 y 2}
 sitio[0]:=@o1;  sitio[1]:=@o2;  sitio[2]:=nil;

 {inicializacion objeto 3: sitio con su transformacion}
 o3.t:=@t3;      o3.d:=@sitio;
 t3[0][0]:=1;    t3[0][1]:=0;    t3[0][2]:=0;
 ...
 {inicializacion objeto 4}
 o4.t:=@t4;      o4.c:=@cu;
 t4[0][0]:=1;    t4[0][1]:=0;    t4[0][2]:=0;
 ...
 {idem objetos 5 y 6}

 {definicion de puerta como conteniendo objetos 4, 5 y 6}
 puerta[0]:=@o4;  puerta[1]:=@o5;  puerta[2]:=@o6;  puerta[3]:=nil;

 {inicializacion objeto 7: puerta con su transformacion}
 o7.t:=@t7;      o7.d:=@puerta;
 t7[0][0]:=1;    t7[0][1]:=0;    t7[0][2]:=0;
 ...
 {idem para definicion de ventana}

 {definicion de casa como conteniendo objetos 3, 7 y 9}
 casa[0]:=@o3;   casa[1]:=@o7;    casa[2]:=@o9;  casa[3]:=nil;

 {inicializacion objeto 10: casa con su transformacion}
 o10.t:=@t10;      o10.d:=@casa;
 t10[0][0]:=100;  t10[0][1]:=0;    t10[0][2]:=300;
 ...
 {inicializacion objeto 11: casa con otra transformacion}
 o11.t:=@t11;      o10.d:=@casa;
 ...
 {definicion de escena como conteniendo objetos 10 y 11}
 escena[0]:=@o10;      escena[1]:=@o11;      escena[2]:=nil;

 {inicializacion objeto 12: escena con su transformacion}
 o12.t:=@t12;    o12.d:=@escena;
 ...
 graf_obj(o11,identidad);
end;
```

Figura 3.7 *Parte de la inicialización de las estructuras necesarias para representar la escena de la Figura 3.8.*

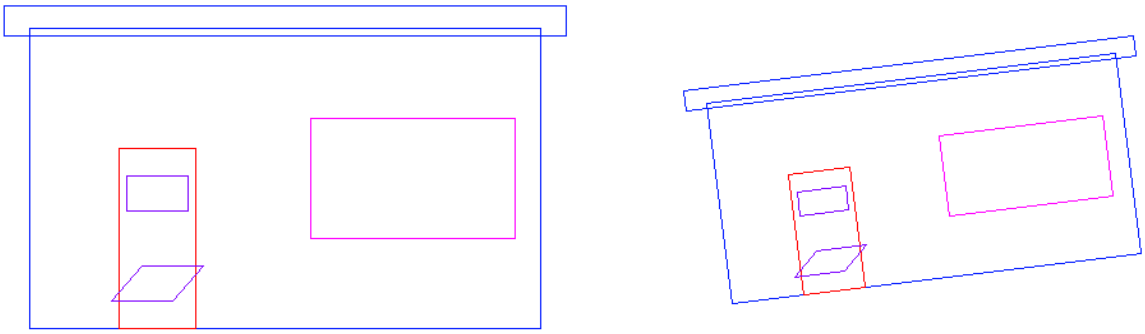


Figura 3.8 *Efecto de modificar una instancia de casa y parte de la definición de la ventana.*

Para solucionar este problema se define un nuevo sistema de coordenadas, denominado sistema de coordenadas del mundo, del cual se representará gráficamente un segmento denominado ventana o *window*. Dicho segmento es puesto en correspondencia con un sector de la pantalla (subconjunto del sistema de coordenadas físico) denominado *viewport*. La operación de transformar las entidades gráficas del window al viewport se denomina *windowing*, y la de eliminar la graficación de entidades que caen fuera del viewport se denomina *clipping*.

3.4.1 Windowing

Las aplicaciones en Computación Gráfica trabajan en sistemas de coordenadas adecuados para cada caso particular. Por ejempo, un histograma meteorológico puede estar en un sistema de coordenadas en milímetros de lluvia vs. tiempo. El dispositivo de salida, por su parte, trabaja en su propio sistema de coordenadas físico. Para lograr una independencia de dispositivo, es necesario trabajar en un espacio que tenga un sistema de coordenadas normalizado. Por dicha razón se define el sistema de coordenadas del mundo, dentro del cual la porción visible de la escena es la contenida en la ventana (en algunos casos se sugiere utilizar una ventana normalizada 0.0—1.0 en x e y, denominada NDC, Normalised Device Coordinate, garantizando un estándar gráfico).

Sean x_{w_i} y x_{w_d} los límites en x (izquierdo y derecho respectivamente), y y_{w_a} y y_{w_b} los límites en y (superior e inferior respectivamente) del window. Dicha ventana debe ser mapeada al subconjunto de la imagen denominado *viewport* definido en el sistema de coordenadas de la pantalla (ver Figura 3.9), cuyos límites son x_{v_i} y x_{v_d} para x (izquierdo y derecho respectivamente), y y_{v_a} y y_{v_b} para y (superior e inferior respectivamente). Por lo tanto, es necesario encontrar los parámetros de una transformación de windowing que lleva un punto p_w del window a un punto p_v del viewport.

Podemos encontrar dichos parámetros planteando semejanzas entre segmentos. Por ejemplo, para la coordenada x de un punto p_w en el window es posible encontrar

$$\frac{x_w - x_{wi}}{x_{wd} - x_{wi}} = \frac{x_v - x_{vi}}{x_{vd} - x_{vi}},$$

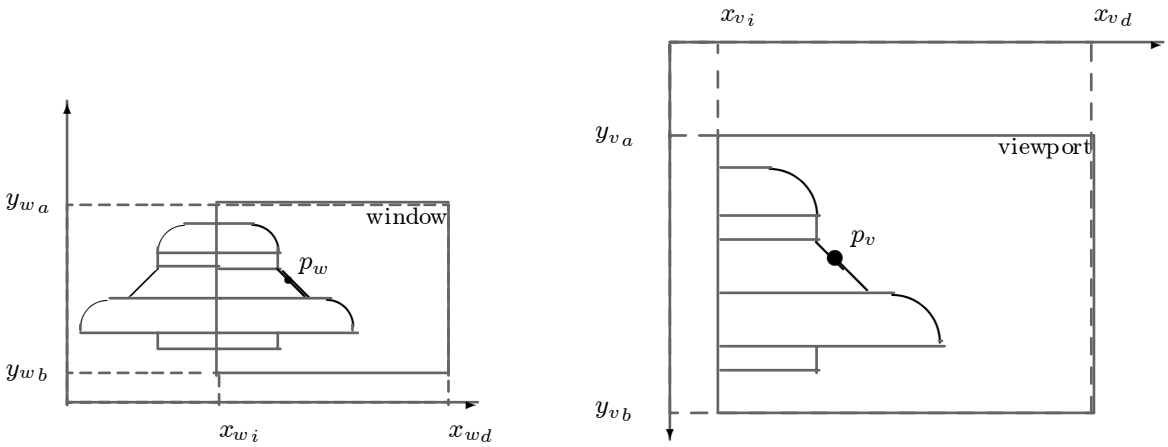


Figura 3.9 *Elementos que definen la transformación de windowing.*

de donde se sigue

$$x_v = \frac{(x_w - x_{w\,i})}{(x_{w\,d} - x_{w\,i})}(x_{v\,d} - x_{v\,i}) + x_{v\,i}.$$

Llamando a al factor constante $\frac{(x_{v\,d} - x_{v\,i})}{(x_{w\,d} - x_{w\,i})}$ obtenemos

$$x_v = (x_w - x_{w\,i})a + x_{v\,i} = ax_w + b,$$

donde $b = x_{v\,i} - ax_{w\,i}$.

De haber utilizado el sistema NDC, la expresión se simplifica a

$$x_v = x_w(x_{v\,d} - x_{v\,i}) + x_{v\,i}.$$

Por último, es necesario discretizar el punto obtenido para transformarlo en pixel (por medio del `round` de x_v).

Es necesario enfatizar un punto de gran importancia en el análisis de la eficiencia de los algoritmos utilizados en Computación Gráfica. Todas las primitivas que hemos utilizado son invariantes frente a las transformaciones que se utilizan. Esto garantiza que es posible aplicar todas las transformaciones (windowing incluída) a los puntos que definen a una entidad gráfica, y luego aplicar la discretización sobre los puntos transformados. Es decir, conmutan las operaciones de transformar y de discretizar. De no haber sucedido así, no hubiésemos tenido otro remedio que discretizar en el espacio de la escena, y luego transformar cada punto de la discretización, lo cual es evidentemente de un costo totalmente inaceptable.

3.4.2 Clipping

Como ya mencionáramos, al aplicar windowing es necesario "recortar" la graficación de aquellas partes de la escena que son transformadas a lugares que quedan fuera del viewport. La manera más

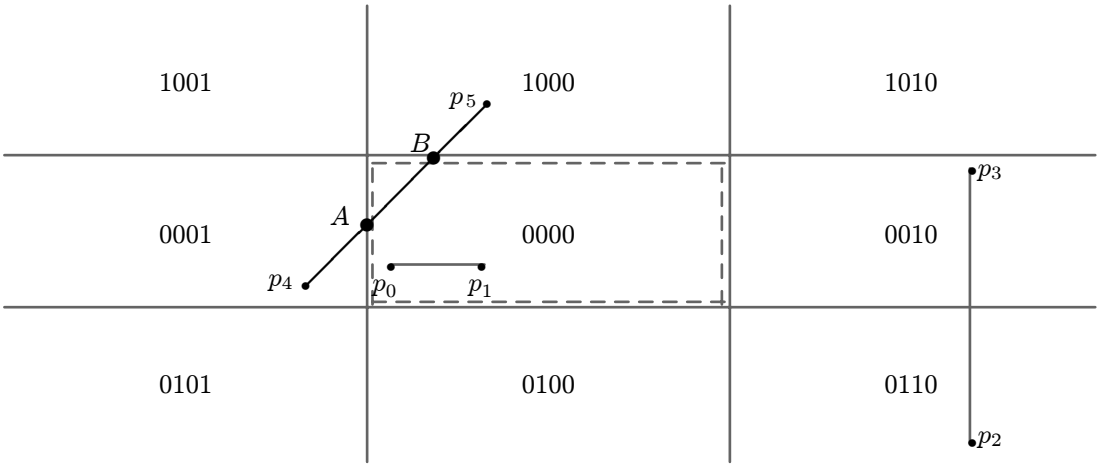


Figura 3.10 *Cómo se etiquetan los extremos de los segmentos en el algoritmo Cohen-Sutherland para clipping de segmentos.*

directa de realizar esta tarea es modificar los algoritmos de discretización de primitivas para que utilizen las coordenadas del viewport (por ejemplo, definidas con variables globales) y grafiquen un pixel sólo si está dentro del mismo.

```
procedure linea_clip(p0,p1:punto;c:col);
...
if (xvi<=x) and (x<=xvd) and                    { xvi, xvd, yva, yvb }
      (yva<=y) and (y<=yvb)                   { son variables globales }
      then putpixel(x,y,col);
...
end;
```

Es muy probable que un mecanismo de este estilo esté implementado dentro de la tarjeta gráfica para proteger el acceso a lugares de memoria inadecuados cuando se reclama un `putpixel` fuera del área de pantalla. De todas maneras, es fácil ver que si gran parte de nuestra escena está fuera del window, un algoritmo por el estilo perderá cantidades de tiempo enormes chequeando pixel por pixel de entidades gráficas que caen fuera del área graficable.

Una técnica mucho más práctica fue propuesta por Cohen y Sutherland para el clipping de segmentos de rectas. La filosofía subyacente es clasificar los segmentos como totalmente visibles o totalmente invisibles (en ambos casos no hay que aplicar clipping!). Aquellos que escapan a la clasificación son subdivididos, y luego se clasifican los subsegmentos resultantes.

La clave del método está en etiquetar los pixels extremos de los segmentos con un cuádruplo de booleanos. Cada uno de ellos es el resultado de preguntar si dicho extremo está fuera del viewport por exceder alguno de sus cuatro límites, en un orden arbitrario. En la Figura 3.10, por ejemplo, se observa cómo quedan etiquetados los pixels extremos en función del área de pantalla en que caen, al ser etiquetados en el orden originalmente propuesto por los autores (`yva,yvb,xvd,xvi`). El viewport está delimitado con una caja punteada.

Podemos ver que el segmento de recta que va de p_0 a p_1 cae totalmente dentro del viewport, y por lo tanto cada pixel de su discretización puede graficarse en forma incondicional. En cambio, el segmento que va de p_2 a p_3 cae totalmente fuera del área graficable, y por lo tanto ninguno de sus pixel debe graficarse. Una situación más compleja ocurre con el segmento que va de p_4 a p_5, dado que tiene partes graficables y partes no graficables.

Las etiquetas de los pixels extremos del segmento permiten clasificar estos tres casos de manera sencilla:

- Si el `or` de ambos extremos es cero, el segmento es totalmente graficable.
- Si el `and` de ambos extremos no es cero, el segmento no es graficable.
- En todo otro caso el segmento puede tener partes graficables y partes no graficables.

En el segmento de recta que va de p_0 a p_1, ambos extremos están etiquetados con `0000` y por lo tanto estamos en el primer caso. En cambio, en el segmento que va de p_2 a p_3 tiene etiquetas `0010` y `0110` y por lo tanto estamos en el segundo caso. Con el segmento que va de p_4 a p_5 tenemos etiquetas `0001` y `1000`, cuyo `or` no es cero y cuyo `and` es cero, por lo que estamos en el tercer caso. Aquí es necesario subdividir el segmento y aplicar recursivamente a cada subsegmento.

Para subdividir un segmento es posible encontrar la intersección del mismo con alguno de los cuatro "ejes" del viewport, dado que todo segmento parcialmente visible intersecta por lo menos a uno de dichos ejes. Por ejemplo, se subdivide el segmento en el punto A, obteniéndose un segmento de p_4 a A (totalmente fuera del viewport) y otro segmento de A a p_5 parcialmente graficable, que se puede subdividir en B en dos segmentos, uno totalmente fuera del viewport y otro totalmente dentro. Otra estrategia, que algunos proclaman más rápida, consiste en subdividir los segmentos por su punto medio, el cual es sencillo de encontrar como promedio de sus extremos. Esta forma de subdividir es análoga a una búsqueda binaria, y por lo tanto la cantidad de subdivisiones es del orden del logaritmo de la longitud del segmento de recta.

3.4.3 La "tubería" de procesos gráficos

Corresponde realizar en este punto una pequeña revisión del capítulo para tener una visión integrada de los distintos procesos involucrados. El punto de vista más importante para tener en cuenta es que se definen las entidades gráficas como compuestas por primitivas discretizables, cuyos parámetros geométricos son puntos. Por ejemplo, describimos una escena como un conjunto de cuadrados transformados, y cada cuadrado es un conjunto de segmentos de recta uniendo los vértices del mismo. Por lo tanto, la tarea esencial del sistema gráfico es indicarle al mecanismo de discretización la ubicación de dichos puntos.

De esa manera, podemos pensar que los procesos del sistema gráfico se ubican en una "tubería", que va tomando puntos de la definición de las entidades gráficas, los va transformando por las diversas transformaciones que ocurren, y los deposita en los algoritmos de discretización de primitivas. A dichos procesos se agrega lo visto en esta sección, es decir la transformación de windowing que relaciona el sistema de coordenadas del mundo con el sistema de coordenadas de pantalla, y el clipping de las partes no graficables de la escena.

También es necesario agregar una transformación más, denominada transformacion del mundo, que lleva a los objetos del sistema de coordenadas de la escena al sistema de coordenadas del mundo en el cual está definido el window (probablemente en NDC). Esta última transformación es también importante porque permite alterar la ubicación y tamaño de *toda* la escena, para realizar efectos

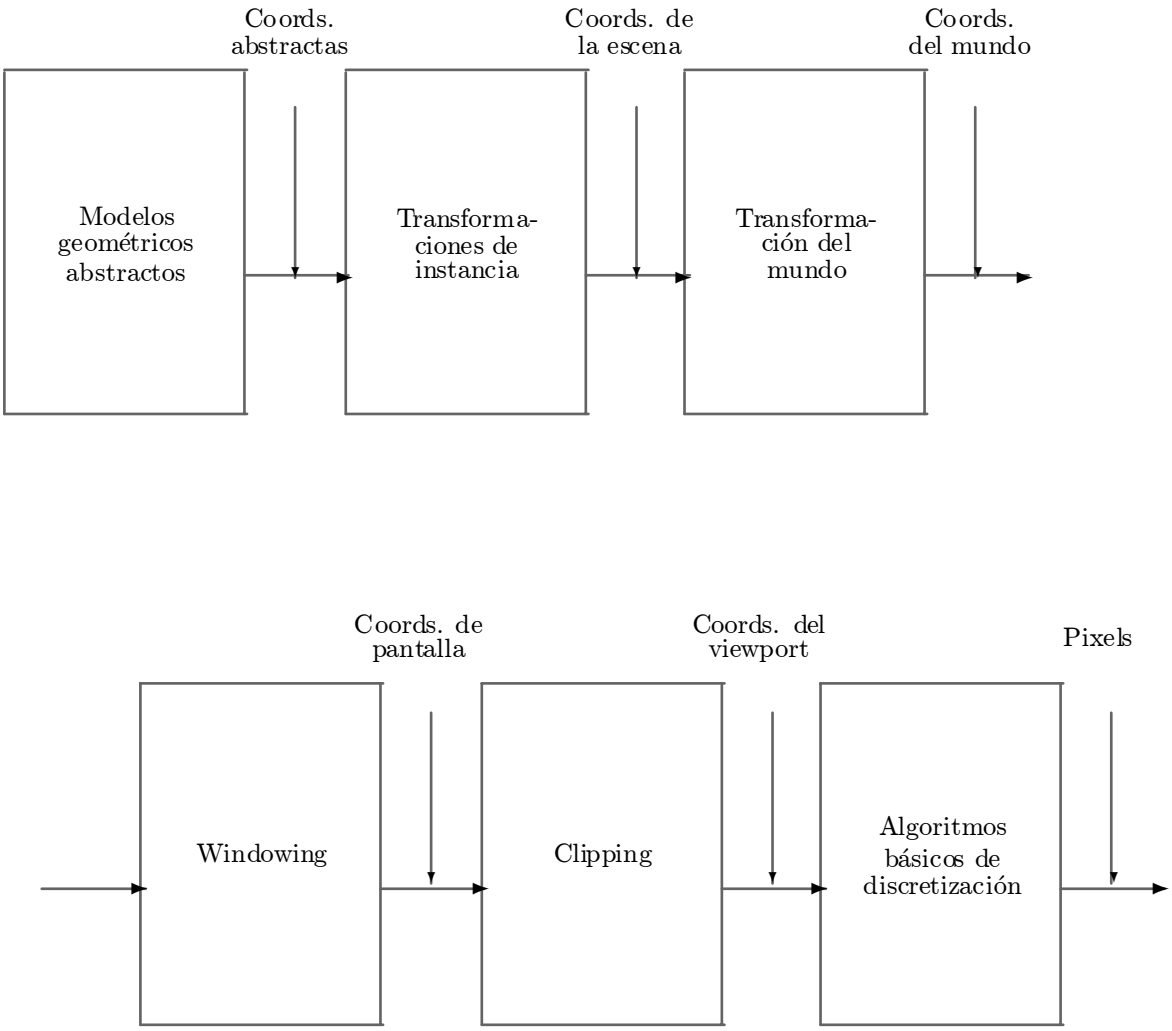


Figura 3.11 *Estructura de la "tubería" de procesos gráficos.*

de zooming, desplazamiento, rotación, etc. En la Figura 3.11 podemos ver cómo se estructuran todos estos procesos.

3.5 Implementación de Paquetes Gráficos

El propósito de esta sección es brindar algunos conceptos introductorios relacionados con la implementación de paquetes gráficos. Un paquete gráfico es una pieza de software destinada a facilitar el desarrollo de aplicaciones gráficas sobre un conjunto de dispositivos gráficos determinados. Un ejemplo sería desarrollar sobre una PC los procesos gráficos descriptos a lo largo de este capítulo sin tener software gráfico específico. Para ello tenemos que solucionar dos problemas. Primero es necesaria la interacción con el hardware, por ejemplo por medio del uso del sistema de interrupciones del sistema operativo. Segundo, es necesario definir un conjunto de operaciones que permitan al usuario manejar el paquete.

Podemos especificar un conjunto de requisitos para un sistema de estas características.

Simplicidad: La interfase con el usuario debe ser amigable, fácil de aprender, y que utilize un lenguaje sencillo. Una de las alternativas más obvias es utilizar una interfase de ventanas con menúes descolgables.

Consistencia: Se espera que un sistema se comporte de una manera consistente y predecible. Una forma de lograr dicho comportamiento es a través del uso de modelos conceptuales. También se espera que no ocurran excepciones, es decir, funciones que actúan incorrectamente en algunos casos particulares.

Completitud: Es necesario encontrar un conjunto razonablemente pequeño de funciones que manejen convenientemente un rango amplio de aplicaciones. También es necesario que no ocurran omisiones irritantes que deban ser suplidas desde el nivel de la aplicación.

Integridad: Los programadores de aplicaciones y los usuarios normalmente tienen una extraordinaria capacidad para vulnerar la integridad de los sistemas. Por lo tanto debe existir un buen manejo de errores operativos y funcionales, normalmente impidiendo el acceso o la manipulación indebida del sistema. Los mensajes de error que se generan en tiempo de ejecución deben ser útiles para los usuarios. Debe existir una buena documentación que sea comprensible para un usuario promedio.

Economía: En tiempo de ejecución, tamaño y recursos utilizados.

Podemos también describir brevemente las capacidades de un paquete gráfico a partir de una clasificación del tipo de funciones y procedimientos que implementa:

Procedimientos de seteado: Se utilizan para inicializar o cambiar el estado de las variables globales del sistema.

- `initgraph`: Se utiliza para inicializar el paquete a un determinado modo gráfico.
- `cls`: Para limpiar el viewport.
- `color`: Para setear el color corriente con el que se graficará en adelante.
- `move`: Para desplazar el cursor gráfico.

Funciones primitivas gráficas: Son las entidades gráficas más sencillas, en función de las cuales se estructuran todas las demás. Por ejemplo las funciones

- `line`: Se utiliza para graficar un segmento de recta con el color corriente, desde la posición del cursor hasta un lugar determinado del viewport.
- `point`: Para prender un pixel con el color corriente y en el lugar determinado por el cursor.
- `circle`: Para trazar una circunsferencia del color corriente, centrada en el cursor y de un radio determinado.
- `poly`: Para trazar una poligonal.

Funciones de transformación: Usualmente para poder modificar la transformación del mundo. Es más intuitivo referirse a dicha matriz a través de funciones `rotate, translate, scale`.

Procedimientos de graficación: Muchas aplicaciones gráficas se utilizan para la generación de diagramas, histogramas y otro tipo de representación de datos. Para ello es muy útil proveer procedimientos como `graphpaper, graphdata, bardata`, que grafican en el viewport el contenido de determinadas estructuras de datos utilizando escalas y métodos establecidos.

Funciones de windowing: Es importante también que se pueda programar desde el nivel de aplicación el uso de ventanas con clipping. Para ello es necesaria la implementación de funciones `setwindow, setviewport` que permitan asignar desde la aplicación los parámetros correspondientes al sistema gráfico.

Modelos y estructuras: Es importante que desde la aplicación se puedan definir objetos complejos como estructuras de otros objetos. Para ello es necesario tener procedimientos como `define .. as record ... end` que permiten describir una entidad gráfica como la unión de entidades más sencillas.

3.6 Ejercicios

1. Modificar los algoritmos del capítulo anterior para utilizar los tipos de datos definidos en este capítulo.

2. Modificar los mismos algoritmos para utilizar las primitivas provistas por el compilador. Comparar los resultados.

3. Implementar la representación de una escena con entidades gráficas estructuradas. Graficar la misma modificando las transformaciones de instancia y la transformación del mundo.

4. Implementar las rutinas de windowing y clipping. Graficar las escenas del ejercicio anterior en una ventana que muestre parte de la escena. Graficar las mismas escenas con toda la pantalla como viewport, sin y con clipping. Evaluar los tiempos.

3.7 Bibliografía recomendada

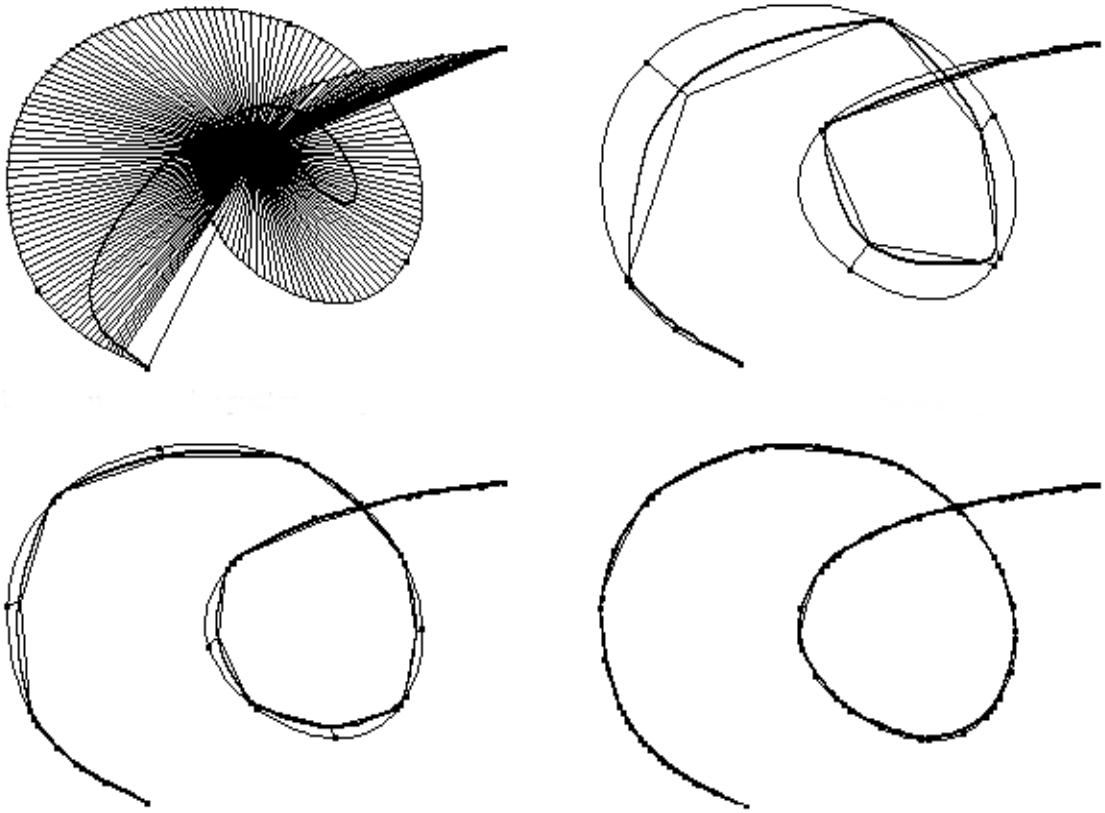
El manejo de las transformaciones en 2D y las coordenadas homogéneas puede consultarse en el Capítulo 4 y el apéndice II del libro de Newman y Sproull [66], aunque utiliza los puntos como vector fila, y las transformaciones como posmultiplicación. Es también recomendable el Capítulo 5 del libro de Foley et. al. [33].

La descripción de los modelos estructurados de entidades gráficas puede consultarse en el Capítulo 9 del Newmann-Sproull, y algunos aspectos avanzados en el Capítulo 2 del libro de Giloi [40]. Una descripción del mismo tema, pero en 3D, figura en el Capítulo 7 del Foley.

El tema de windowing y clipping de segmentos de recta puede leerse en el Capítulo 5 del Newman-Sproull y en el Capítulo 3 del Foley. Blinn discute en [13] la descripción y ventajas del windowing sobre viewports no cuadrados (los cuales son los más usuales, sin duda). El clipping de entidades más complejas, por ejemplo polígonos, caracteres o círculos, puede ser bastante difícil. Recomendamos la lectura de [79] para el clipping de polígonos no convexos, y de [11] para el clipping de entidades en general.

El tema de la implementación de paquetes gráficos está adecuadamente tratado en el Capítulo 6 del Newman-Sproull. Los aspectos más detallados pueden consultarse en el Capítulo 8 del libro de Giloi. La descripción de un paquete gráfico tridimensional interactivo, junto con una descripción de su modelo conceptual y algunos detalles de implementación, pueden consultarse en [20].

4 Aproximación e Interpolación de Curvas



4.1 Motivaciones

Hasta ahora hemos visto cómo elaborar gráficos a partir de los comandos gráficos más comunes en un lenguaje de programación (el punto y el segmento de línea recta). Estas primitivas son suficientes en el sentido de que cualquier otra construcción geométrica puede ser convenientemente aproximada con puntos y segmentos, hasta el punto de ser indistinguible para una resolución gráfica dada. Muy pronto se comprendió en la Computación Gráfica que las técnicas de representación de objetos están muy limitadas en las posibilidades geométricas de las entidades gráficas que se pueden modelar y graficar.

Esencialmente, los modelos basados en la representación de *puntos* destacados, los cuales son unidos entre sí por aristas, tienen la ventaja de que todas las operaciones involucradas en la graficación, excepto el clipping, se realizan por medio de la transformación de dichos puntos. Los mismos son posteriormente unidos entre sí por medio de algoritmos eficientes de discretización de de segmentos de recta. Abandonar esta filosofía de trabajo implica un aumento de varios órdenes de magnitud en el costo de procesamiento.

Podemos describir un objeto determinado –una cuádrica, por ejemplo– por medio de un sistema de ecuaciones, Esto implicaría que debemos muestrear dicho sistema de ecuaciones una cantidad suficiente de veces para obtener un resultado gráfico adecuado, procesando luego cada muestra por la misma tubería de transformaciones. Por otra parte, si dicho objeto ocupa una porción pequeña o muy grande de la pantalla, este esquema de representación permite relacionar la cantidad necesaria de muestras para producir un resultado adecuado con el tamaño o porporción que el objeto ocupa en la pantalla.

Sin embargo, mantenernos dentro del modelo de puntos y vértices puede tener grandes desventajas a la hora de representar objetos con geometrías más variadas. Por ejemplo, para representar objetos cuya descripción ecuacional se desconoce, se puede utilizar un conjunto determinado de puntos que conformen un "cuadriculado" del mismo. Dichos datos pueden provenir de un *scanner*, de muestras, o de simulaciones procedimentales. Esto presupone de por sí un compromiso complejidad-calidad, es decir, representaciones más fieles implican un mayor costo en tiempo y memoria.

En esta representación "poligonal", podemos elegir un buen compromiso para garantizar una apariencia adecuada del objeto graficado en circunstancias normales. Pero al representarse el modelo a muy grandes escalas, su naturaleza poligonal se volvería evidente, pudiendo inclusive "desaparecer" el objeto si en su proyección en la pantalla no aparece ningún vértice o arista. Y por el contrario, si la escala es muy pequeña, el objeto ocupa una parte reducida de la pantalla pero para ser graficado requiere procesar la misma cantidad de puntos y aristas.

Pero no es ésta la desventaja más importante de una representación de objetos por medio de polígonos. A comienzos de la década del 60 ya era posible comandar por computadora las maquinarias necesarias para producir piezas en madera, plástico o acero, en lo que se denomina Computer Aided Manufacturing (CAM) o manufactura asistida por computadora. Dichas piezas pueden ser luego utilizadas en el desarrollo industrial, como por ejemplo la matricería o estampado de carrocerías de automóviles. Para ello es necesario representar la forma de dichas piezas de modo que la ejecución del programa produzca el resultado adecuado. Con la representación poligonal podemos llegar eventualmente a obtener una estructura de datos adecuada, pero, como puede suceder, si se requieren cambios o modificaciones de último momento, entonces es necesario acometer la trabajosa tarea de editar punto por punto en la base de datos que representa los polígonos.

Los sistemas comerciales se hicieron sensibles a estas dificultades. Los utilitarios de dibujo y diseño gráfico comenzaron a incluir la posibilidad de trabajar con círculos, arcos y cónicas, mientras

que sistemas más avanzados de dibujo técnico proveen procedimientos para aproximar y ajustar curvas planas, y para combinar perfiles planos con direcciones de extrusión para elaborar modelos de objetos tridimensionales más complejos. Muy pronto se llegó a la conclusión que esta forma de trabajar es esencialmente incorrecta, trabajosa y sujeta a errores. Por dicha razón se comenzaron a estudiar los fundamentos de los métodos que estudiaremos en este Capítulo, y que constituyen esencialmente lo que se conoce como Computer Aided Design (CAD), Computer Aided Geometric Design (CAGD) o diseño (geométrico) asistido por computadora.

El pleno poder para modelar objetos tridimensionales complejos a partir de descripciones geométricas precisas solo se obtuvo a partir del desarrollo en el CAD de las técnicas de aproximación de curvas y superficies paramétricas a partir de *puntos de control*. En la actualidad el CAD se independizó del CAM y constituye una disciplina aparte, con sus propias motivaciones, e intereses que van más allá de la manufactura industrial, aunque esta última siempre requiere una fase previa de diseño.

Los métodos de aproximación e interpolación de curvas que veremos a continuación, entonces, están pensados en función de facilitar la tarea de diseño, la cual es muchas veces una tarea iterativa de prueba y error. En algunas aplicaciones el diseño tiene un uso *analítico*, es decir, a partir de un determinado modelo real se obtiene una representación característica manipulable y más económica. Otro tipo de uso es el *sintético*, en el cual se parte de un conjunto de especificaciones geométricas o analíticas que van determinando la forma final del objeto. Por lo tanto la creación y modificación de modelos geométricos debe ser sencilla y predecible. Pero al mismo tiempo debe ser eficiente en su cómputo y por lo tanto compatible con el modelo de procesamiento visto en los Capítulos anteriores.

El origen de estos sistemas, que permiten modelar objetos con total libertad, puede rastrearse a los alrededores de la década del 60. Los desarrollos más importantes se iniciaron en la industria automotriz francesa, donde Pierre de Casteljau en Citroën, y Pierre Bézier en Renault elaboraron el fundamento teórico de los primeros sistemas de aproximación de curvas que se sobreponían exitosamente a los problemas técnicos y geométricos de los métodos matemáticos de interpolación de funciones basados en los polinomios de Lagrange. Esencialmente ambos trabajos coinciden, aunque fueron independientemente desarrollados. Como Bézier fue el único que publicó sus resultados, se llevó todo el crédito y la fama.

Muy pronto se produjo una convergencia con los métodos de la teoría de aproximación de funciones, especialmente con las aproximaciones polinomiales a trozos o *splines*. Ya en 1975 comenzaron a existir foros de discusión y Conferencias especializadas, y en la década del 80 el CAD se independizó como disciplina, con sus propias metodologías, publicaciones y grupos de interés. Pese al gran desarrollo ocurrido desde entonces, las curvas de Bézier-de Casteljau continúan siendo la base, tanto porque se basan en las bases funcionales numérica y analíticamente más estables conocidas, como por el hecho demoledor de que todo otro método puede ponerse en términos de casos particulares de estas curvas (salvo casos excepcionales, que normalmente no son aceptados como “buenos” métodos).

Los métodos paramétricos de aproximación de curvas logran flexibilidad de representación y eficiencia en el cómputo, al utilizar una formulación basada en *puntos de control*. Dichos puntos de control son localizaciones en el plano o el espacio que gobiernan la forma final de la curva o superficie. La implementación de un método consiste en encontrar funciones polinomiales de grado bajo que aproximan o interpolan a dichos puntos de control con características analíticas o geométricas específicas. Dichas funciones pueden ser luego evaluadas para obtener una cantidad suficiente de puntos, los cuales son procesados como en el caso poligonal.

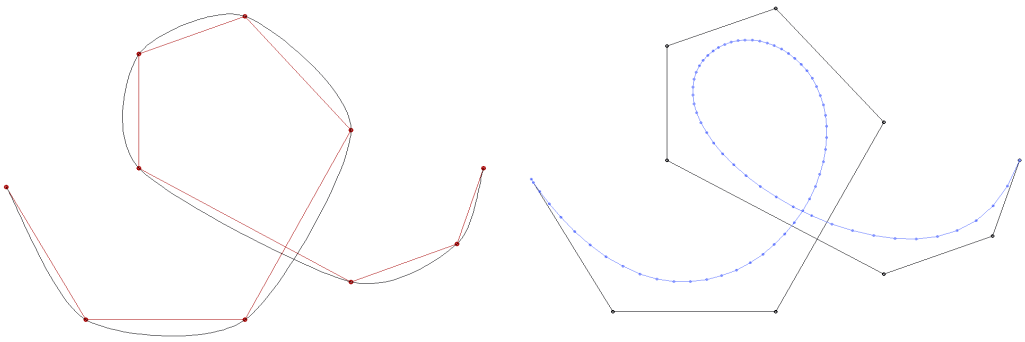


Figura 4.1 *Interpolación* vs. *aproximación.*

4.1.1 Especificaciones y requisitos

Las curvas que buscamos deben tener ciertas propiedades geométricas que es necesario especificar con claridad, del mismo modo que lo hicimos en el Capítulo 2 con los métodos de discretización. Como ya mencionáramos, la descripción de estas entidades se realiza por medio de puntos de control. La secuencia de puntos de control, unidos por segmentos de rectas, se denomina *grafo de control*. Podemos mencionar, entre otras, las siguientes especificaciones en el caso de curvas, aunque no es difícil extrapolar los requisitos para superficies.

Interpolación o aproximación: Dado un conjunto de puntos de control, es posible que la curva resultante pase por todos ellos (ver Figura 4.1(a)). En dicho caso el método que encuentra la curva a partir de los puntos de control se denomina *método de interpolación*. Puede suceder, en cambio, que la curva normalmente no pase por los puntos, excepto tal vez por los puntos extremos (ver Figura 4.1(b)). Entonces estamos frente a un *método de aproximación*.

Invariancia afín: Esta propiedad es indispensable si buscamos mantenernos dentro del "modelo de tubería", con su eficiencia computacional asociada. Básicamente, se espera que sea indistinto aplicar una transformación afín al grafo de control y luego aproximar (o interpolar) una curva sobre el mismo, o aproximar una curva sobre el grafo original y luego transformarla. Dicho de otra manera, las operaciones de aproximar (o interpolar) y de transformar *conmutan*. Esta propiedad permite garantizar no solo que la apariencia de una curva aproximante no varía si se la traslada, rota, etc., sino además que es correcto encontrar un conjunto de puntos sobre la curva y aplicarle el mismo procesamiento que a las demás entidades de la escena.

Entidades multivaluadas: Si las curvas aproximantes son de la forma $y = f(x)$ entonces están limitadas a no poder representar entidades multivaluadas (ver Figura 4.2). Esta limitación es intolerable, dado que la mayor parte de las curvas o superficies tienen características geométricas de esta naturaleza. Además, la representación debe ser necesariamente independiente del sistema de coordenadas.

Disminución de variaciones: El grafo de control "sugiere" la forma general de la curva, pero localmente ésta podría tener variaciones u oscilaciones (ver Figura 4.3(a)). Esto es indeseable en general. Se desea que toda variación u oscilación esté ya presente en el grafo de control, y que, por el contrario, las mismas sean suavizadas o atenuadas por la curva resultante (ver Figura 4.3(b)).

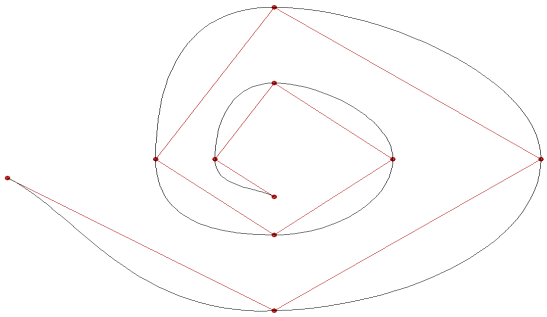


Figura 4.2 *Entidades multivaluadas.*

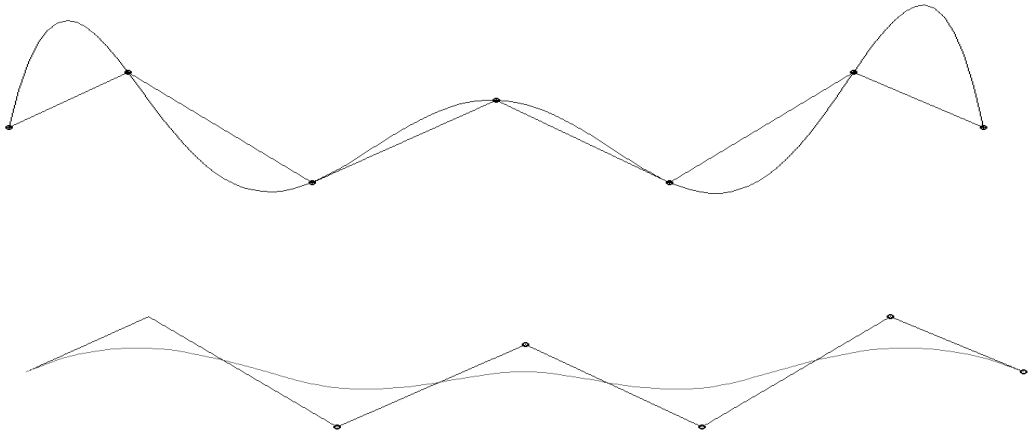


Figura 4.3 *Amplificación o disminución de variaciones.*

Orden de continuidad: Dada una curva, sea k el máximo orden de derivación de la curva tal que la función resultante sea contínua. En dicho caso diremos que la curva es C^k contínua, o que tiene un orden de continuidad k. Es indispensable conocer de antemano el orden de continuidad resultante de una curva aproximante.

Control local o global: En algunos métodos la curva aproximante experimenta un cambio localizado si se modifica un único punto de control, mientras que en otros, toda la curva es modificada. Los primeros métodos tienen, entonces, la propiedad de *control local* y los segundos de *control global*. Ninguna de las dos propiedades es en sí mejor que la otra, sino que depende del tipo de aplicación.

4.1.2 La representación paramétrica

La clave para poder satisfacer los requisitos planteados más arriba consiste en suponer que la curva resultante de aproximar o interpolar el grafo de control es un conjunto de funciones paramétricas, donde el parámetro u varía dentro de un intervalo cerrado (usualmente $u \in [0, 1]$). Por ejemplo, para una curva en el espacio podemos encontrar tres funciones $f_x(u)$, $f_y(u)$, y $f_z(u)$ que gobiernen la posición de un punto sobre la curva para un valor dado de u.

$$\boldsymbol{C}(u) = \begin{cases} x(u) = f_x(u) \\ y(u) = f_y(u) \\ z(u) = f_z(u) \end{cases}$$

En principio no hay nada que limite la forma que pueden asumir estas funciones. Sin embargo, desde el punto de vista de la Computación Gráfica, es importante que la evaluación de las mismas sea estable, económica y robusta. Otro de los puntos importantes, entonces, es que dichas funciones normalmente son polinomios de grado relativamente bajo.

$$\boldsymbol{C}(u) = \begin{cases} f_x(u) = c_x^n u^n + c_x^{n-1} u^{n-1} + \cdots + c_x^1 u + c_x^0 \\ f_y(u) = c_y^n u^n + c_y^{n-1} u^{n-1} + \cdots + c_y^1 u + c_y^0 \\ f_z(u) = c_z^n u^n + c_z^{n-1} u^{n-1} + \cdots + c_z^1 u + c_z^0 \end{cases}$$

El grado y los coeficientes de dichos polinomios se extraen de la ubicación de los puntos de control. Por lo tanto, los métodos de interpolación y aproximación tienen como objetivo encontrar las funciones $f_x(u)$, $f_y(u)$, y $f_z(u)$ a partir de los datos geométricos de dichos puntos.

4.2 Interpolación de Curvas

Los métodos de interpolación de curvas fueron por primera vez estudiados en el análisis matemático y se utilizan en la actualidad en el cálculo numérico. Su aplicación en la Computación Gráfica, sin embargo, es muy limitada dado que en los últimos 20 años se desarrollaron técnicas más específicas para dicho ámbito. De todas maneras, las ideas básicas y terminología utilizadas en los modelos de aproximación de curvas se apoyan en los modelos de interpolación. Por dicha razón es que dedicaremos una Sección para describir brevemente los métodos de Lagrange y de Hermite para interpolar puntos.

4.2.1 Interpolación de Curvas de Lagrange

La determinación del interpolante de Lagrange es uno de los métodos polinomiales más directos. Sea una secuencia de $n+1$ números reales $x_0 < x_1 < \cdots < x_n$, denominados *nudos* y un segundo conjunto de $n+1$ números $y_0 < y_1 < \cdots < y_n$, denominados *valores*, de modo que $p_i = (x_i, y_i)$ es un punto de control. Una secuencia de puntos de control es llamada *grafo de control*, y debe ser tal que la secuencia de nudos (valores de la coordenada x sea no decreciente. Entonces buscamos una función polinomial $f(x)$ de grado n tal que satisfaga el problema de interpolación del grafo de control

$$\forall i \in [0..n], f(x_i) = y_i.$$

```
const pts_ ctrol= (cantidad de puntos de control);

type  punto = record x,y,z,w:real
              end;
grafo_contr =  array [1..n] of punto;
...
procedure lagrange(var g:grafo_contr);
var x,y,incrx,num,den:real;
   i,j,k:integer;
   p:punto;
begin;
 incrx:=(g[pts_ctrol].x-g[1].x)/100;
 for i:=0 to 100 do begin
   x:=g[1].x+i*incrx;        y:=0;
   p.z:=1;                   p.w:=1;
   for j:=1 to pts_ctrol do begin
     num:=1;                 den:=1;
     for k:=1 to pts_ctrol do
       if (k<>j) then begin
        num:=num*(x-g[k].x);
        den:=den*(g[j].x-g[k].x);
       end;
     y:= y+g[j].y*num/den;
   end;
   p.x:=x;                   p.y:=y;
   graf_linea(p);
  end;
end;
```

Figura 4.4 *Procedimiento que implementa la interpolación de Lagrange.*

El interpolante de Lagrange se define como

$$f(x) = \sum_{i=0}^{n} L_i(x) y_i,$$

donde

$$L_i = \frac{(x-x_0),(x-x_1),\cdots,(x-x_{i-1}),(x-x_{i+1}),\cdots,(x-x_n)}{(x_i-x_0),(x_i-x_1),\cdots,(x_i-x_{i-1}),(x_i-x_{i+1}),\cdots,(x_i-x_n)}.$$

Por ejemplo, si $n = 1$, tenemos dos nudos x_0 y x_1, y sus valores asociados, encontrando que

$$f(x) = \frac{x-x_1}{x_0-x_1} y_0 + \frac{x-x_0}{x_0-x_1} y_1,$$

expresión que es idéntica a

$$f(x) = y_0 + (y_1 - y_0)\frac{x_0 - x}{x_0 - x_1}.$$

Es decir, para un par de puntos de control, el interpolante de Lagrange coincide con el segmento de recta que une a dichos puntos. Es posible demostrar que el interpolante de Lagrange siempre existe

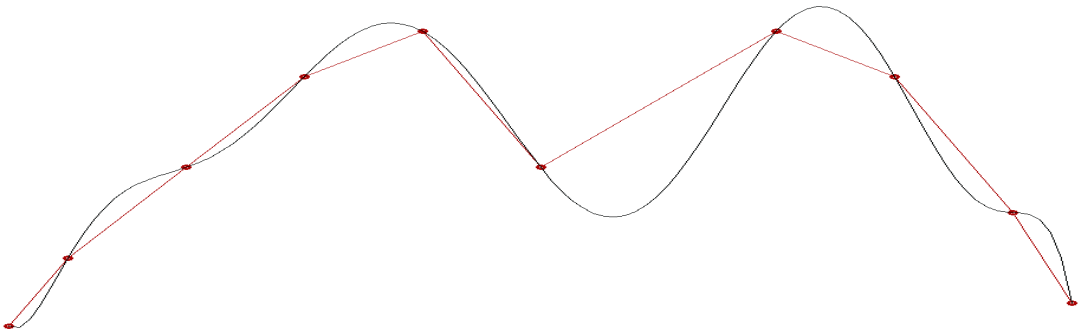


Figura 4.5 *Interpolación de una secuencia de puntos de control por medio del interpolante de Lagrange.*

y es único, para un n finito, y que es el único polinomio de grado n que satisface el problema de interpolar $f(x_i) = y_i$. El procedimiento de la Figura 4.4 muestra una implementación del método, el cual es utilizado en la Figura 4.5 con un grafo de control dado.

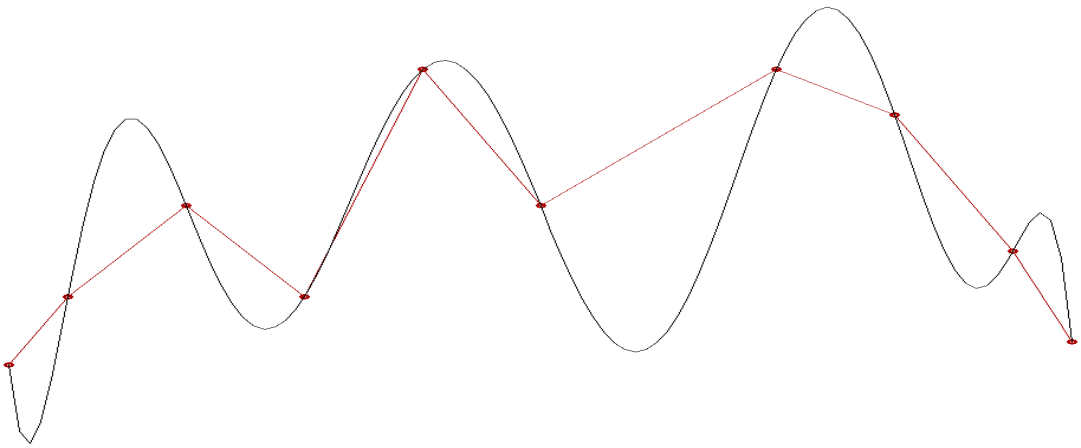


Figura 4.6 *Efecto de mover un punto de control al grafo en el método de Lagrange.*

Es importante notar que si se modifica el valor de un nudo cualquiera, se modifica la expresión completa del polinomio. Es decir, este método tiene control global. Al mismo tiempo, dado que la secuencia de nudos es no decreciente, el método no permite interpolar curvas multivaluadas.

La desventaja más considerable del método de Lagrange es que si aumentamos la cantidad de puntos (tal vez para mejorar la calidad de la entrada al procedimiento), entonces aumenta el grado de los polinomios. Esto en sí no sería más que un inconveniente computacional, pero en la práctica, a partir de $n = 5$, comienzan a ocurrir oscilaciones indeseadas en el interpolante entre los puntos de

control, las cuales se vuelven de mayor amplitud cuando n aumenta o cuando se altera la posición de un punto de control (ver Figuras 4.5 y 4.6). Es decir, al modificar o agregar un punto de control para mejorar la interpolación, en realidad lo que ocurre es que aumentan las oscilaciones.

4.2.2 Interpolación de Curvas de Hermite

Una solución posible para el problema de las oscilaciones es la propuesta por Hermite. En Hermite, la interpolación se realiza entre pares sucesivos de puntos de control, de modo que no solo los valores son interpolados en los nudos, sino también un cierto número de derivadas. Por ejemplo, para interpolar una curva plana $\boldsymbol{C}_i(u)$ que pase por $p_i = (x_i, y_i)$ y por $p_{i+1} = (x_{i+1}, y_{i+1})$, con una determinada derivada $\dot{p}_i$ en el primer punto y con una derivada $\dot{p}_{i+1}$ en el segundo, necesitamos expresar a las funciones $f_x(u)$ y $f_y(u)$ como polinomios cúbicos en u. Esto es así dado que tenemos, para cada uno de ellos, cuatro "datos" (las posiciones y derivadas al comienzo y al final de la curva), y por lo tanto podemos encontrar cuatro "incógnitas" (los coeficientes del polinomio).

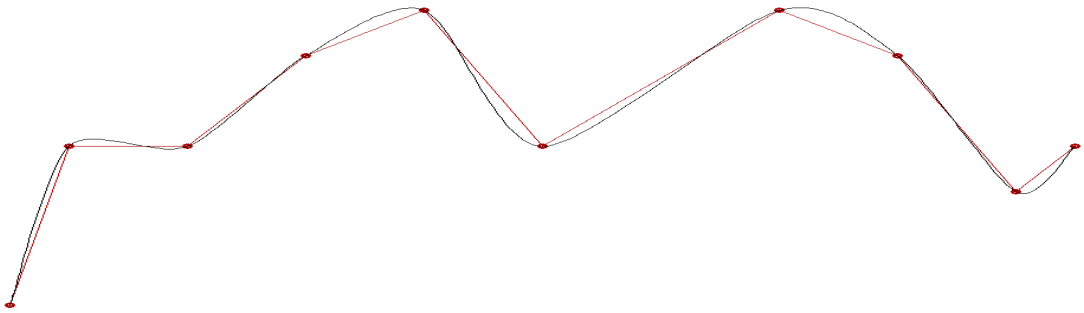


Figura 4.7 *Interpolación de Hermite de un grafo de control.*

Es decir, a partir de los datos de posición (x_i, y_i), (x_{i+1}, y_{i+1}), y de derivada $(\dot{x}_i, \dot{y}_i)$, $(\dot{x}_{i+1}, \dot{y}_{i+1})$, existen dos únicos polinomios $f_x(u)$ y $f_y(u)$ tales que el interpolante de Hermite

$$\boldsymbol{C}(u) = \begin{cases} f_x(u) = c_x^3 u^3 + c_x^2 u^2 + c_x^1 u + c_x^0 \\ f_y(u) = c_y^3 u^3 + c_y^2 u^2 + c_y^1 u + c_y^0 \end{cases}$$

es un segmento de curva plana que pasa por los puntos con las derivadas establecidas.

Si bien los datos de posición surgen de una manera relativamente directa del problema de diseño, no sucede lo mismo con los datos de derivada. Si queremos encontrar el modelo de un objeto cualquiera, es posible estimar la posición de un punto dado y de la pendiente en el mismo, pero la magnitud de la derivada puede ser difícil de estimar. Por lo tanto, trabajar con estimadores de derivadas puede ser una tarea un tanto artesanal.

Para encontrar los coeficientes de los polinomios debemos primero asignar un dominio para el parámetro u. Normalmente se utiliza $u \in [0, 1]$, con lo cual por ejemplo $f_x(0) = x_i$, $f_x(1) = x_{i+1}$, y también $\dot{f}_x(0) = \dot{x}_i$, etc., y también podemos expresar por ejemplo $\boldsymbol{C}_i(0) = p_i$ y $\dot{\boldsymbol{C}}_i(1) = \dot{p}_{i+1}$.

```
procedure hermite(var p1,p2:punto; var d1,d2:deri);
var u,x,y,cy0,cy1,cy2,cy3,cx0,cx1,cx2,cx3:real;
    p:punto;
    i:integer;
begin
 p.z:=1;                              p.w:=1;
 cx3:=2*p1.x-2*p2.x+d1.x+d2.x;        cy3:=2*p1.y-2*p2.y+d1.y+d2.y;
 cx2:=-3*p1.x+3*p2.x-2*d1.x-d2.x;     cy2:=-3*p1.y+3*p2.y-2*d1.y-d2.y;
 cx1:=d1.x;                           cy1:=d1.y;
 cx0:=p1.x;                           cy0:=p1.y;
 p.x:=cx0;                            p.y:=cy0;
 graf_punto(p);
 for i:=1 to 30 do begin
  u:=i/30;
  p.x:=cx0+u*(cx1+u*(cx2+u*cx3)); p.y:=cy0+u*(cy1+u*(cy2+u*cy3));
  graf_linea(p);
 end;
end;
```

Figura 4.8 *Procedimiento que implementa la interpolación de Hermite.*

Además, $\boldsymbol{C}_i(1) = \boldsymbol{C}_{i+1}(0)$, etc. Esta asignación de dominio para el parámetro es arbitraria, pudiéndose encontrar una formulación equivalente para cualquier otra asignación no trivial.

Podemos ahora expresar a los polinomios de Hermite como producto escalar de un vector (fila) de funciones de u por un vector (columna) de coeficientes:

$$f_x(u) = c_x^3 u^3 + c_x^2 u^2 + c_x^1 u + c_x^0 = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}. \tag{4.1}$$

Si bien esta manipulación algebraica es relativamente trivial, sus implicaciones son importantes. Por ejemplo, nos permite ver a un polinomio como un elemento o "punto" en un espacio de funciones. En este caso la base del espacio funcional es

$$\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

y el vector de coeficientes determina "las coordenadas" de $f_x(u)$ dentro de dicho espacio. Esta base funcional es denominada la *base polinomial*, en este caso para funciones de orden cúbico, pero pueden existir otras bases. De hecho, en las Secciones siguientes utilizaremos otras familias de bases funcionales más adecuadas que la base polinomial para la interpolación y aproximación de curvas.

Es posible expresar a las derivadas de la curva de una manera similar:

$$\dot{f}_x(u) = 3c_x^3 u^2 + 2c_x^2 u + c_x^1 + 0 = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}.$$

Podemos ahora expresar todas nuestras restricciones de una manera homogénea:

$$f_x(0) = c_x^3 0^3 + c_x^2 0^2 + c_x^1 0 + c_x^0 = \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}$$

$$f_x(1) = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}$$

$$\dot{f}_x(0) = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}$$

$$\dot{f}_x(1) = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}.$$

Por fin, podemos representar todas las restricciones en un único sistema de ecuaciones:

$$\begin{bmatrix} f_x(0) \\ f_x(1) \\ \dot{f}_x(0) \\ \dot{f}_x(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}.$$

Por lo tanto, para encontrar el arreglo de coeficientes debemos encontrar la matriz inversa, denominada matriz de Hermite M_H:

$$M_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Nuestro problema de interpolación se transformó en un producto de un vector por una matriz. Lo más importante a tener en cuenta con este método es que M_H es la misma matriz para los polinomios f_x y f_y de cada segmento $\boldsymbol{C}_i$ de la curva. Con los dos polinomios de cada segmento de curva, entonces, se elige una secuencia (probablemente uniforme) de valores de u entre cero y uno. La evaluación de los polinomios en cada uno de dichos valores de u produce una secuencia de puntos, los cuales, unidos por una poligonal, se aproximan a la curva de Hermite con un grado de precisión arbitrario. Por lo tanto, un procedimiento que recibe 2 puntos de control y sus derivadas y los interpola con un segmento polinomial cúbico puede seguirse del procedimiento mostrado en la Figura 4.8.

Dicho procedimiento debe ser llamado $n-1$ veces para un grafo de control de n puntos. La modificación de cualquiera de los cuatro datos involucrados en los coeficientes de un polinomio modifica a todo el polinomio. Podemos ver en la Figura 4.9 la interpolación de Hermite entre dos puntos, variando la dirección de la derivada en el primero de ellos, y en la Figura 4.10 el efecto de modificar la magnitud de la misma. En este caso se modifican los dos segmentos de curva que concurren a dicho punto. Por lo tanto, el método de Hermite tiene control local. También podemos destacar que los datos de las derivadas son parámetros de diseño que se pueden mofificar localmente para alterar la forma definitiva de la curva (ver Figura 4.11).

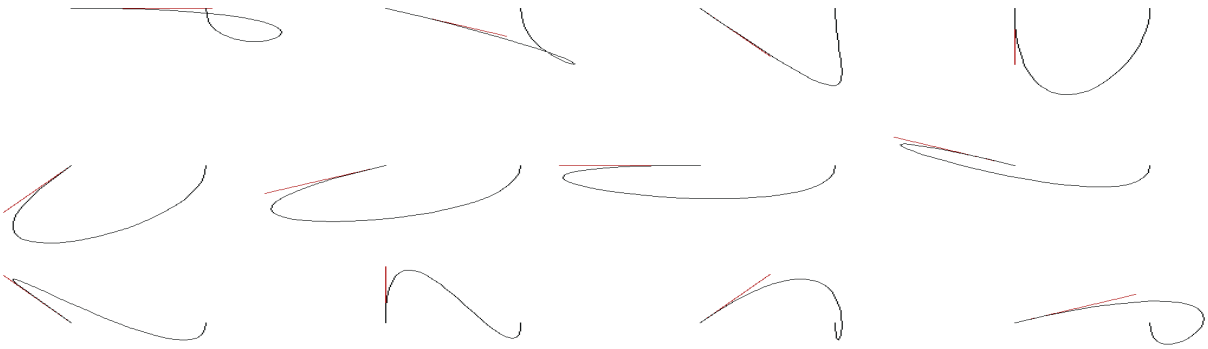


Figura 4.9 *Modificación de la dirección de la derivada en el primer punto de control.*

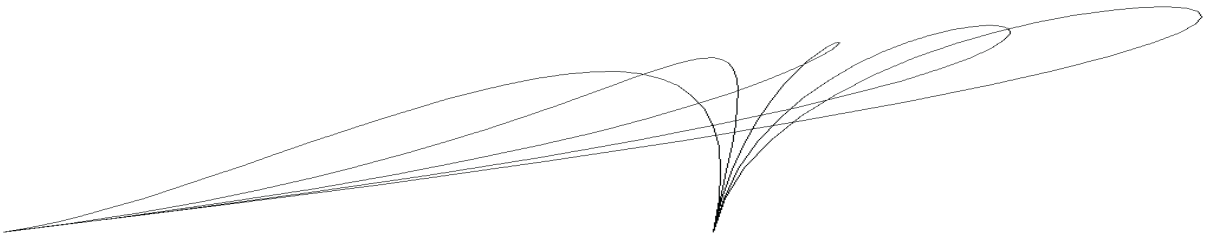


Figura 4.10 *Modificación de la magnitud de la derivada en el primer punto de control.*

Es importante observar que la segunda derivada de los segmentos de curva no están restringidos, es decir, normalmente podemos esperar que $\ddot{\boldsymbol{C}}_i(1) \neq \ddot{\boldsymbol{C}}_{i+1}(0)$, es decir, existen discontinuidades de curvatura en los nudos. Una forma de evitar este problema es agregar condiciones de igualdad en la segunda derivada, y realizar el mismo desarrollo pero para un vector de coeficientes de un polinomio de quinto grado. Sin embargo, esto complica aún más la tarea artesanal de estimar la entrada al algoritmo, dado que hay que agregar estimadores de curvatura en los nudos. Dicha complicación, sumada a la relativa complejidad resultante de trabajar con polinomios de quinto grado, hacen que este método resulte inadecuado en la práctica, prefiriéndose los métodos que veremos en las siguientes Secciones de este Capítulo.

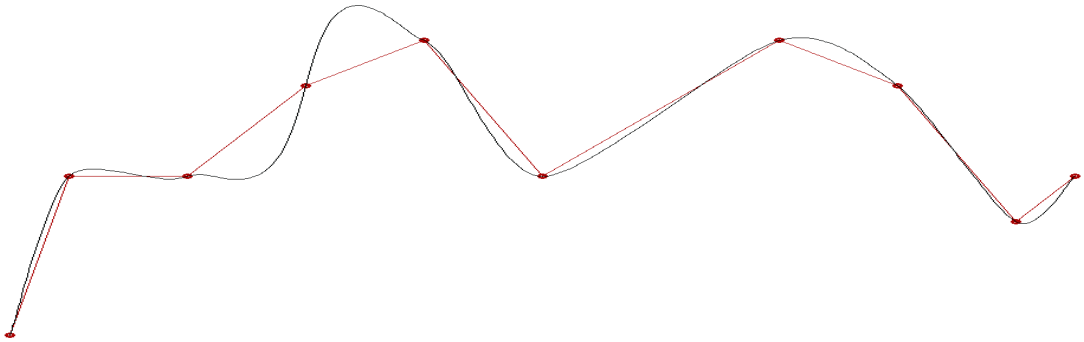


Figura 4.11 *Interpolación de Hermite del mismo grafo de control que la Figura 4.7 modificando las restricciones de derivadas en el cuarto punto de control.*

4.3 Aproximación de Curvas I: de Casteljau, Bernstein y Bézier

Los modelos de interpolación de curvas que vimos en la Sección anterior cumplen con el requisito de encontrar una curva que pase por una secuencia de puntos de control obedeciendo ciertas restricciones geométricas. Los resultados, sin embargo, son insatisfactorios desde el punto de vista del CAD y del modelado de primitivas gráficas en general. El modelo para construir curvas que veremos en esta Sección, denominadas *curvas de Bézier* es el primer método de aproximación desarrollado exclusivamente para los objetivos de la Computación Gráfica, principalmente en las compañías fabricantes de autos en Francia, durante los comienzos de la década del 70. De alguna manera, las curvas de Bézier comparten con los polinomios de Lagrange el hecho de que todos los puntos de control participan de la forma final de las mismas, es decir, tienen control global. Sin embargo, como veremos, la contribución de cada uno de los puntos determina una suma convexa, por lo que la curva no puede tener las oscilaciones indeseadas de Lagrange. Por el contrario, las curvas de Bézier suelen ser excesivamente suaves, y se amoldan al grafo de control de una manera perezoza.

El estudio de las curvas de Bézier es fundamental en este tema, porque provee los elementos geométricos y analíticos necesarios para comprender todos los demás temas en CAD. Esto es así dado que, como veremos a lo largo de este capítulo, todos los métodos de aproximación e interpolación (excepto el de Lagrange) pueden plantearse como un conjunto de casos particulares de aproximaciones de Bézier.

4.3.1 Construcción de parábolas

Una de las técnicas más antiguas para el trazado de parábolas en el dibujo técnico es la siguiente. Sean p_0, p_1, $p_2 \in E^2$ (o E^3) y un parámetro $u \in [0,1]$ (ver Figura 4.12). Entonces podemos mapear distintos valores de u en $[0,1]$ a los segmentos $\overline{p_0, p_1}$ y $\overline{p_1, p_2}$, obteniendo, respectivamente,

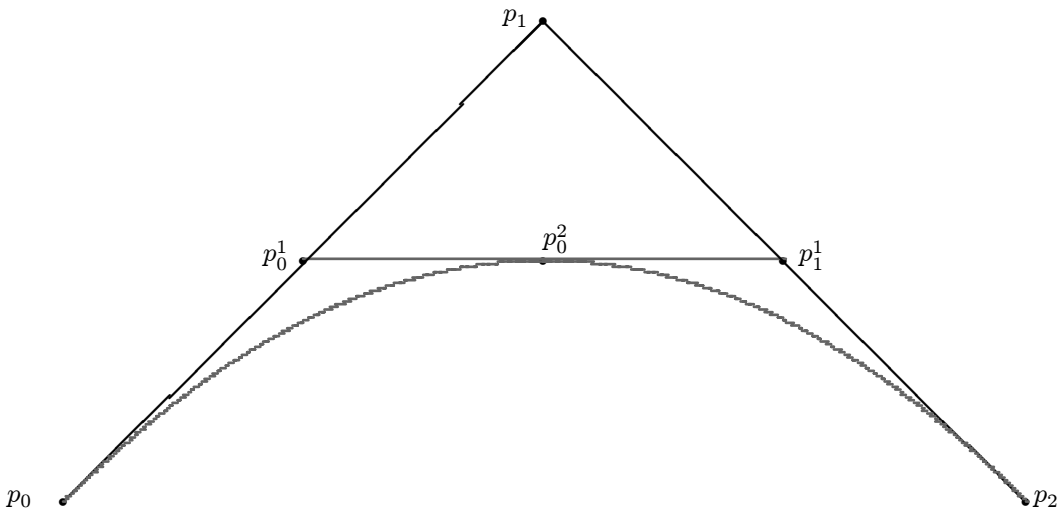


Figura 4.12 *Construcción de parábolas.*

los nuevos puntos $p_0^1(u)$ y $p_1^1(u)$. Por último, mapeamos los mismos valores de u en el segmento $p_0^1(u), p_1^1(u)$, obteniendo un punto $p_0^2(u)$ que pertenece a la parábola.

Utilizando la expresión paramétrica para representar un punto a lo largo de un segmento de recta, podemos encontrar expresiones para los puntos

$$\begin{aligned} p_0^1(u) &= (1-u)p_0 + up_1 \\ p_1^1(u) &= (1-u)p_1 + up_2 \\ p_0^2(u) &= (1-u)p_0^1(u) + up_1^1(u), \end{aligned}$$

donde $p_0^2(u)$ es la expresión paramétrica de una parábola tangente a los segmentos $\overline{p_0, p_1}$ y $\overline{p_1, p_2}$ en p_0 y p_2, respectivamente. Si $u \in [0,1]$, entonces la curva es el segmento de dicha parábola que va exactamente desde p_0 hasta p_2 (ver Figura 4.13).

Es importante tener en cuenta que el algoritmo sigue siendo correcto fuera del intervalo $u \in [0,1]$. En dicho caso, se obtienen las partes correspondientes de la parábola por *extrapolación* lineal de los puntos de control (ver Figura 4.14). La demostración de que dicha curva es en efecto una parábola es un tanto indirecta y apela a la intuición geométrica del lector. Desdoblando las ecuaciones anteriores obtenemos

$$p_0^2(u) = (1-u)^2 p_0 + 2u(1-u)p_1 + u^2 p_2,$$

lo cual es una expresión de segundo grado, es decir, una cónica. Como es fácil ver, la curva no es un caso degenerado de cónica, por lo que puede ser una parábola, una elipse o una hipérbola. Pero analizando su derivada vemos que si u tiende a infinito, la derivada tiende a $p_0 - 2p_1 + p_2$, y si u tiende a menos infinito, la derivada tiende a $-(p_0 - 2p_1 + p_2)$. Por lo tanto, la curva tiene una sola asíntota, lo que excluye a las elipses (que no tienen asíntotas) y a las hipérbolas (que tienen dos asíntotas).

Es importante notar sin embargo que si $u \in [0,1]$, entonces el segmento de curva es una suma convexa de los tres puntos que la determinan. Además, como todos los pasos involucrados en la construcción de la parábola son combinaciones lineales, una transformación afín aplicada a los puntos de control determina una nueva parábola que coincide con el resultado de aplicar la

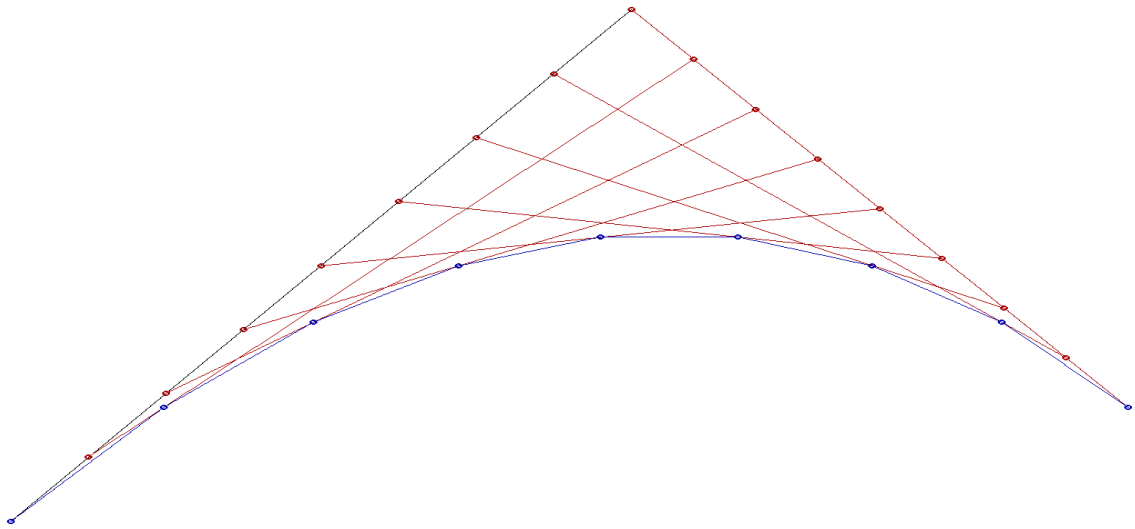


Figura 4.13 *Aproximación de una parábola con interpolaciones lineales.*

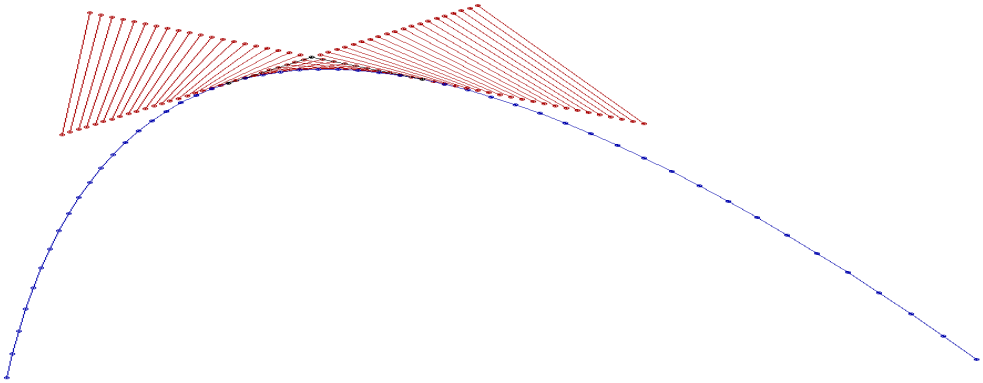


Figura 4.14 *Aproximación de una parábola fuera del intervalo* $u \in [0,1]$.

misma transformación afín a la parábola obtenida con los puntos sin transformar. Es decir, el procedimiento de construir una parábola es invariante frente a transformaciones afines.

4.3.2 El algoritmo de de Casteljau

Consideraremos ahora una generalización de la construcción de la parábola. Sea una secuencia de $n+1$ puntos o grafo de control $p_0, \cdots, p_n \in E^2$ (o E^3) y un parámetro $u \in [0,1]$. Podemos mapear el valor de u en $[0,1]$ a los segmentos $\overline{p_i, p_{i+1}}$ obteniendo nuevos puntos $p_i^1(u), i = 0..n-1$, donde

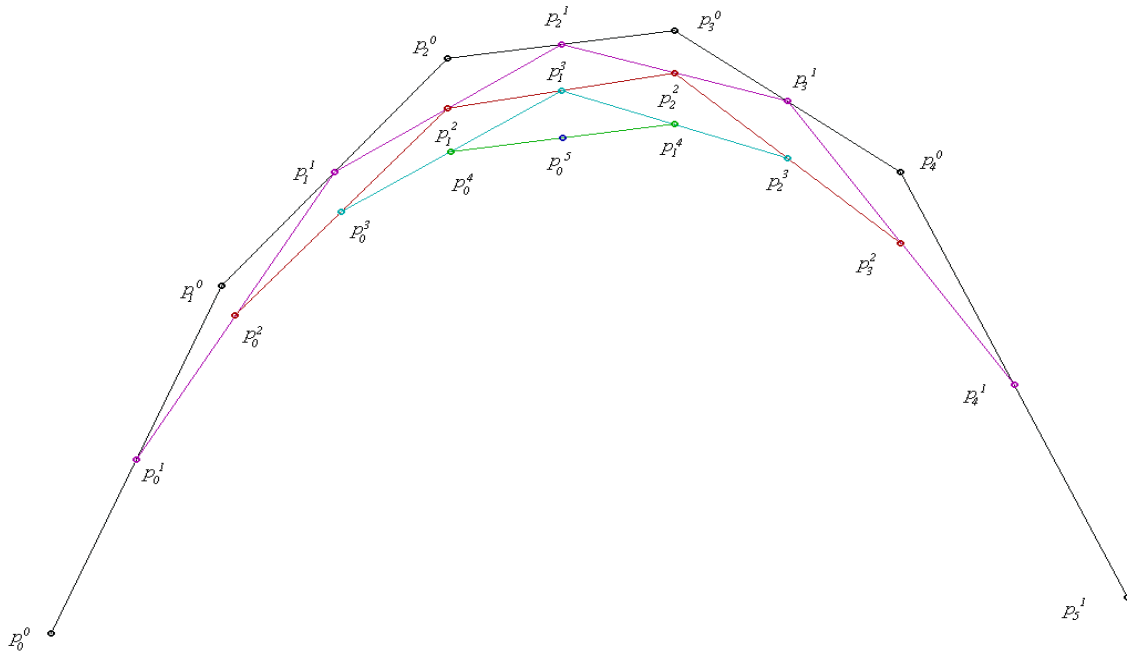


Figura 4.15 *Aproximación de una secuencia de puntos de control por interpolaciones lineales sucesivas (algoritmo de de Casteljau).*

p_i^1 denota al i-ésimo punto de la secuencia de puntos de la primer interpolación. Luego mapeamos el valor de u en $[0,1]$ a los segmentos $\overline{p_i^1(u), p_{i+1}^1(u)}$ obteniendo nuevos puntos $p_i^2(u), i = 0..n-2$, donde p_i^2 denota al i-ésimo punto de la secuencia de puntos de la segunda interpolación. En general, mapeamos el valor de u en el segmento $\overline{p_i^k(u), p_{i+1}^k(u)}$, obteniendo un punto $p_i^{k+1}(u), k = 1..n, i = 0..n-k$ (ver Figura 4.15). La recurrencia termina en el punto $p_0^n(u)$, el cual es un punto que pertenece a la *curva de Bézier* $\boldsymbol{C}(u)$ que *aproxima* al grafo de control inicial.

Si llamamos $p_i^0(u) = p_i$ a los puntos de control iniciales (interpolaciones de orden cero), entonces el algoritmo define a la curva $\boldsymbol{C}(u) = p_0^n(u)$ computando la recurrencia

$$p_i^k(u) = (1-u)p_i^{k-1}(u) + up_{i+1}^{k-1}(u) \tag{4.2}$$

con $k = 1..n$, e $i = 0..(n-k)$ (ver Figura 4.16). Podemos ver el resultado de aplicar el algoritmo de de Casteljau sobre el mismo grafo de control pero con distintos valores de u entre 0 y 1 en la Figura 4.18.

Las curvas de Bézier se obtienen computando la recurrencia de de Casteljau para una cantidad suficiente de valores de u entre 0 y 1. Normalmente se asigna un paso fraccionario, se computa el valor de la curva incrementando dicho paso, y se unen dichos valores con una poligonal (ver Figura 4.17).

Es posible demostrar geométricamente que la curva de Bézier así definida tiene todas las propiedades requeridas. Es invariante frente a transformaciones afines, dado que está definida exclusivamente en base a interpolaciones lineales. Tiene la propiedad de *control global*, dado que

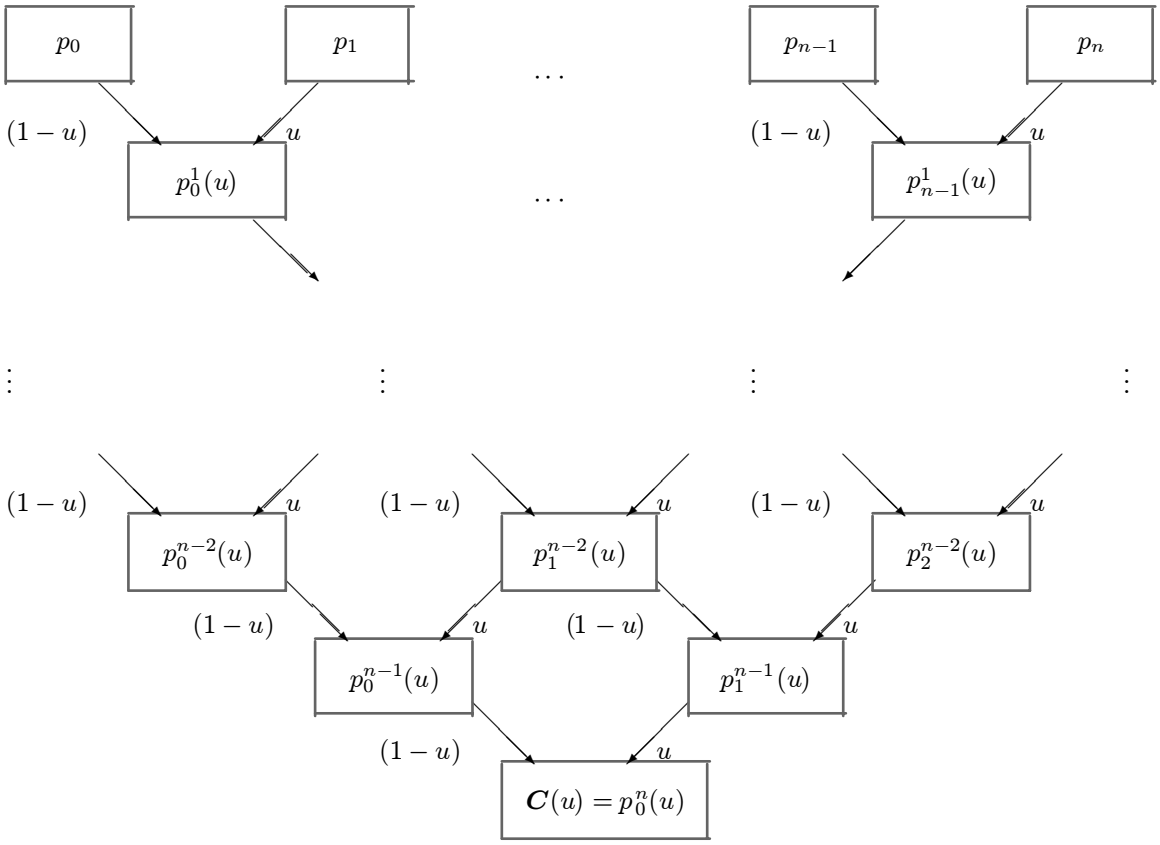


Figura 4.16 *Secuencia de interpolaciones en el algoritmo de de Casteljau.*

(excepto en los extremos), todos los puntos de control ejercen influencia sobre su forma. Tiene la propiedad de *armazón convexo*, dado que cada punto de la curva se obtiene como combinaciones afines convexas de los puntos de control. La curva comienza en p_0 con tangente $\overline{p_1, p_0}$ y termina en p_n con tangente $\overline{p_{n-1}, p_n}$ (ver Figura 4.19).

Es posible hacer coincidir los puntos de control extremos para obtener una curva cerrada. Es más, si $\overline{p_1, p_0}$ es colineal con $\overline{p_{n-1}, p_n}$, entonces la curva es continuamente derivable en dicha unión. También se muestra en dicha figura que es posible repetir consecutivamente la posición de un punto para que éste ejerza una mayor influencia en la forma de la curva (ver Figura 4.20).

4.3.3 La base de Bernstein

Uno de los inconvenientes del algoritmo de de Casteljau es que debe realizar n^2 interpolaciones, muchas de las cuales son redundates, dado que lo importante es el resultado final y no los resultados intermedios de la computación. Es posible encontrar una formulación alternativa de menor costo computacional. La idea es expresar a la curva $\boldsymbol{C}(u)$ como una combinación afín convexa de los

```
procedure de_casteljau(u:real;g:grafo_control;var p:punto);
...
begin
  for k:=1 to n do begin
    for l:=0 to n-k do begin
      g[l].x:=(1-u)*g[l].x+u*g[l+1].x;
      g[l].y:=(1-u)*g[l].y+u*g[l+1].y;
    end;
   end;
   p.x:=g[0].x;           p.y:=g[0].y;
end;

procedure Bezier(var g:grafo_control);
...
begin
  p.x:= g[0].x;           p.y:=g[0].y;
  for k:=1 to paso do begin
    u:=k/paso;
    de_casteljau(u,g,p);
    graf_linea(p);
  end;
end;
```

Figura 4.17 *Procedimiento que implementa el algoritmo de de Casteljau.*

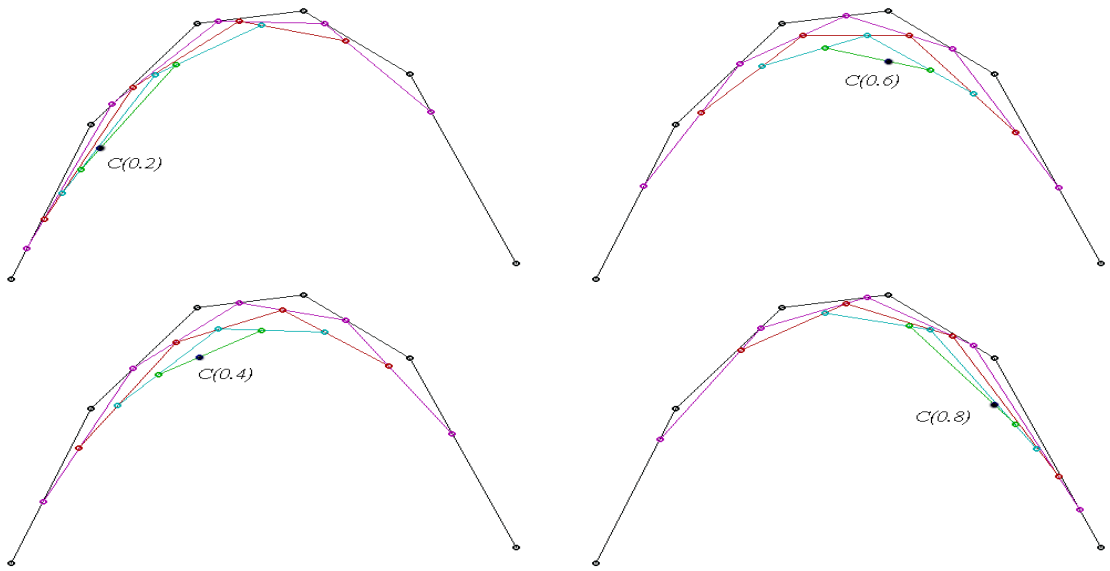


Figura 4.18 *Algoritmo de de Casteljau efectuado con distintos valores del parámetro u.*

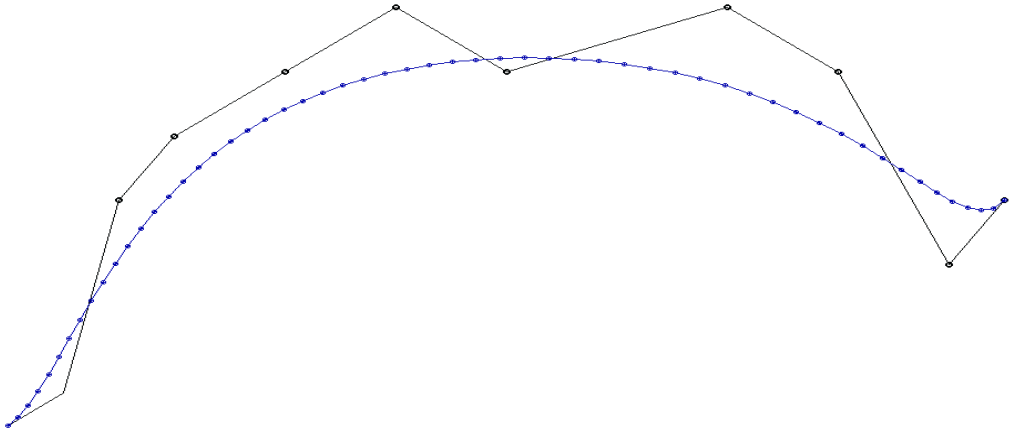


Figura 4.19 *Un ejemplo de curvas de Bézier.*

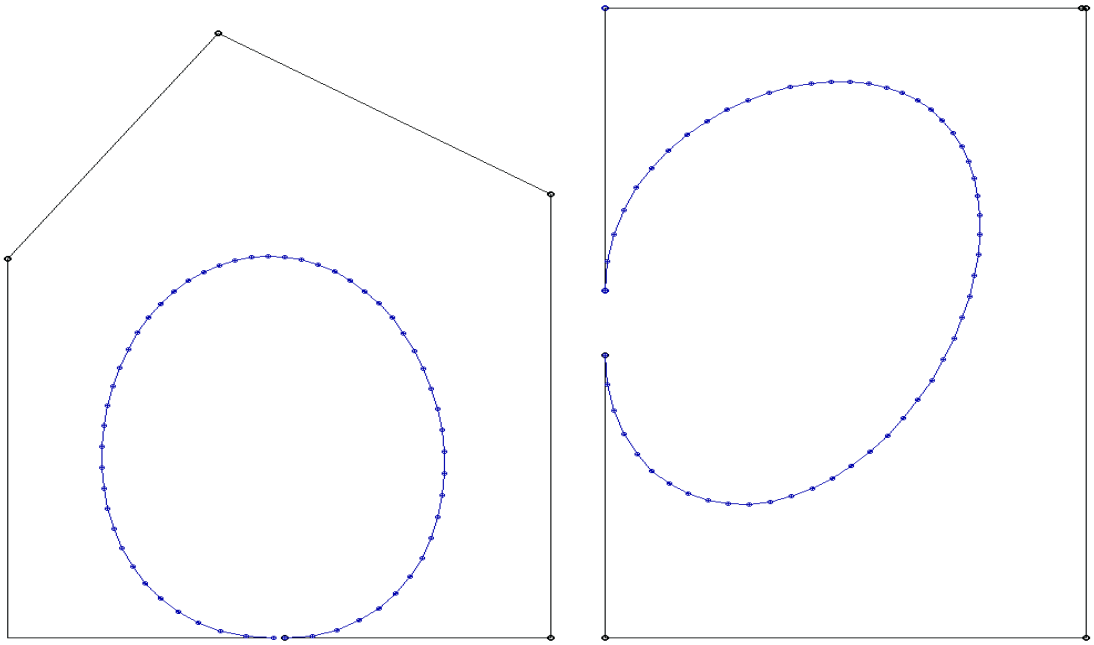


Figura 4.20 *Curvas de Bézier cerradas y con puntos repetidos.*

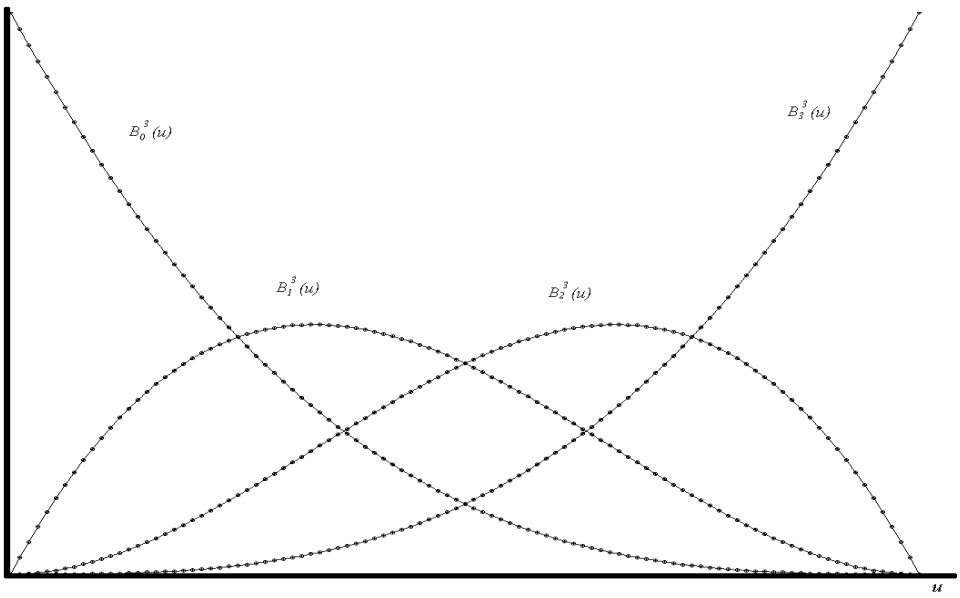


Figura 4.21 *Polinomios de Bernstein cúbicos.*

puntos de control:

$$\boldsymbol{C}(u) = [B_0^n(u), B_1^n(u), \cdots, B_n^n(u)] \begin{bmatrix} p_0 \\ p_1 \\ \cdots \\ p_n \end{bmatrix}, \tag{4.3}$$

donde el vector $[B_0^n(u), B_1^n(u), \cdots, B_n^n(u)]$ es un conjunto de funciones que debe conformar una *base funcional* para los polinomios de grado n, diferente a la base polinomial de Hermite vista en la Sección anterior.

Otra forma de interpretar esta formulación es como una sumatoria:

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} B_i^n(u) p_i,$$

donde la función $B_i^n(u)$ es la *función de mezcla* asociada al punto de control p_i. Luego de examinar la Figura 4.16, podemos ver que la contribución de cada punto a $\boldsymbol{C}(u)$ depende de productos de u y $(1-u)$. Por ejemplo, p_0 tiene una contribución de la forma $(1-u)^n$, p_1 influye con $n(1-u)^{n-1}u$, etc.

Bézier propuso utilizar dichas funciones como base para la curva, siendo que las mismas habían ya sido estudiadas con el nombre de los polinomios de Bernstein. De esa manera, la forma de Bézier de una curva utiliza la base de Bernstein:

$$B_i^n(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i}.$$

La serie de números $\frac{n!}{i!(n-i)!}$ es la conocida como "triángulo de Pascal", y es muy sencilla de calcular. Es posible comprobar que la formulación de la curva $\boldsymbol{C}(u)$ expresada en la Ecuación 4.2,

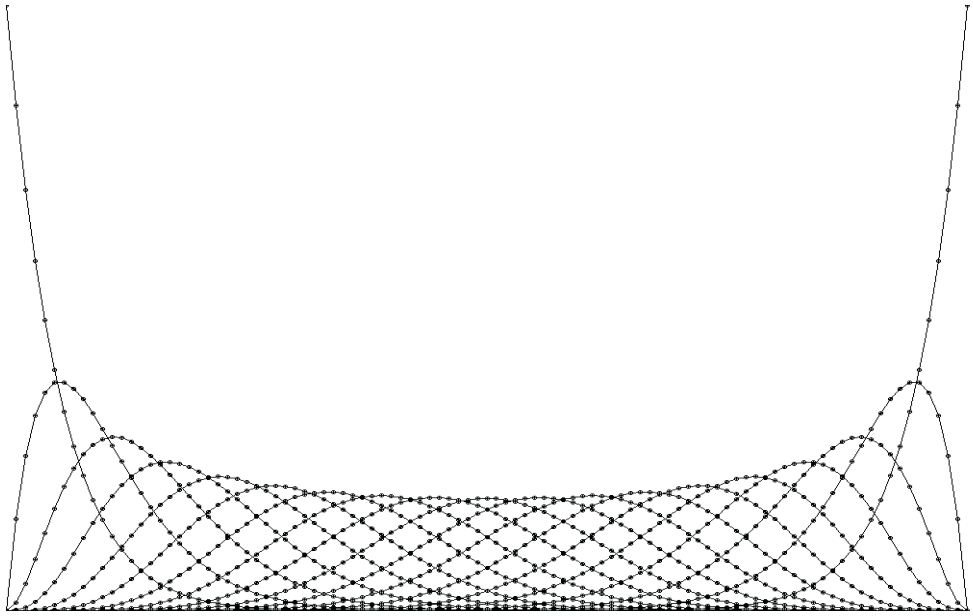


Figura 4.22 *Polinomios de Bernstein de orden 20.*

con $\boldsymbol{C}(u) = p_0^n(u)$, coincide con la Ecuación 4.3 a través de un análisis cuidadoso de la Figura 4.21, donde se puede ver por ejemplo el gráfico de la base de polinomios de Bernstein cúbicos. Dejamos como ejercicio para el lector que constate geométricamente, en la Figura 4.19, que puntos interpolados sucesivamente con el algoritmo de de Casteljau para distintos valores de u son idénticos a los obtenidos por medio de la Ecuación 4.3.

La base de Bernstein posee varias propiedades:

- $\sum_{i=0}^{n} B_i^n(u) = 1$ (es una base).
- $B_i^n(u) \geq 0$ (es convexa).
- $B_i^n(u) = (1-u)B_i^{n-1}(u) + uB_{i+1}^{n-1}(u)$ (es recursiva).
- Es muy económica de computar.

4.4 Aproximación de Curvas II: B-Splines

Las curvas de Bézier son de grado muy alto para grafos de control complejos. Además tienden a suavizar excesivamente la geometría del grafo de control, es decir, se vuelven "insensibles" a pequeños cambios locales. Este efecto se puede constatar al considerar la escasa importancia que tiene un punto de control intermedio en una curva de grado 20 (ver Figura 4.22). Además, en determinadas circunstancias es simplemente preferible no tener control global. Por dicha razón, otra de las alternativas usuales en Computación Gráfica para aproximar curvas es utilizar Splines. El objetivo de estas curvas es obtener control local y bajo grado.

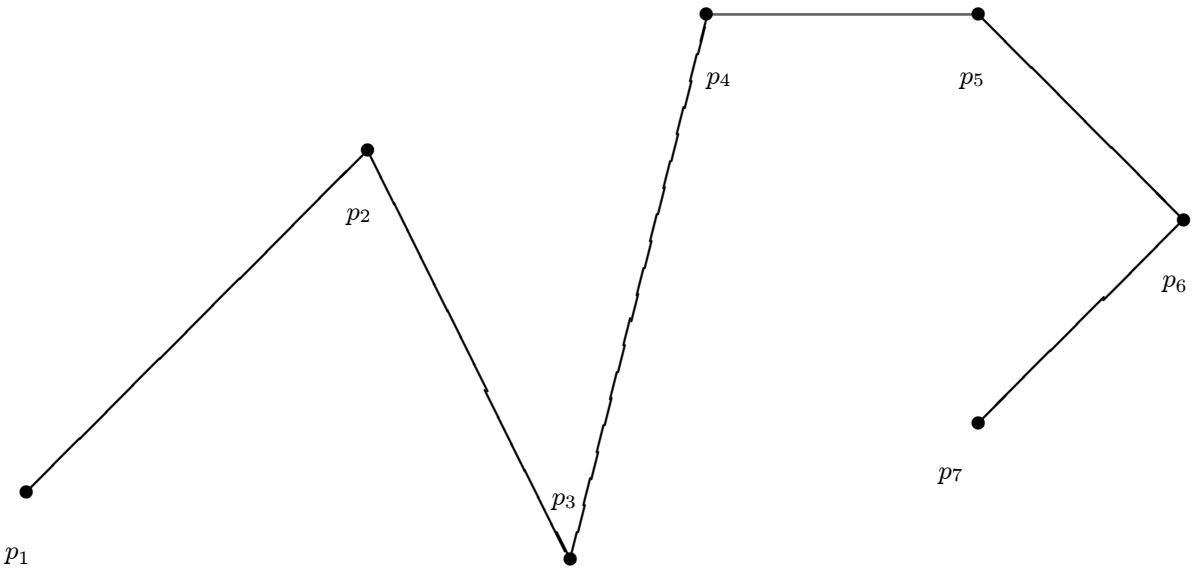


Figura 4.23 *Una "curva" Spline, de primer grado.*

Los Splines deben su nombre a las herramientas curvilineas utilizadas por los dibujantes para el trazado de figuras curvas. En matemática se utiliza el término para referirse a funciones definidas como uniones de segmentos polinomiales. Por lo tanto, el método de Hermite visto hace un par de Secciones es implícitamente un Spline cúbico. Los Splines más usados en Computación Gráfica, sin embargo, utilizan formulaciones semejantes a la de las curvas de Bézier vista en la Sección anterior. Es decir, son Splines determinados por bases funcionales, y por lo tanto se los denomina B-Splines.

La diferencia entre Bézier y B-Splines es que las bases de esta última formulación tienen *soporte local*, es decir, son no nulas solamente para un subconjunto de la secuencia de puntos de control. La curva, entonces, es una combinación afín convexa de un subconjunto localmente determinado de los puntos de control. El grado de los polinomios en un B-Spline está determinado por la cantidad de puntos que localmente afectan a la curva, y determina el orden de continuidad de la misma, es decir, la cantidad de veces que es continuamente diferenciable en las uniones. El grado usual de una curva B-Spline es $n = 3$, lo que determina que cada segmento queda definido por $n+1$ puntos de control sucesivos, y que en las uniones el orden de continuidad es $n - 1$.

4.4.1 Parámetro local

Una forma de entender la mecánica en la definición de una curva B-Spline es partir del caso trivial en el cual la "curva" aproximante es de primer grado. Podemos pensar que la poligonal es la "curva" resultante de unir los puntos de control con segmentos de polinomios de grado $n = 1$, definidos entre $n+1 = 2$ puntos sucesivos, y con un orden de continuidad $n-1 = 0$ en las uniones (ver Figura 4.23).

Cada segmento está definido como

$$\boldsymbol{C}_i(u) = (1-u)p_i + up_{i+1},$$

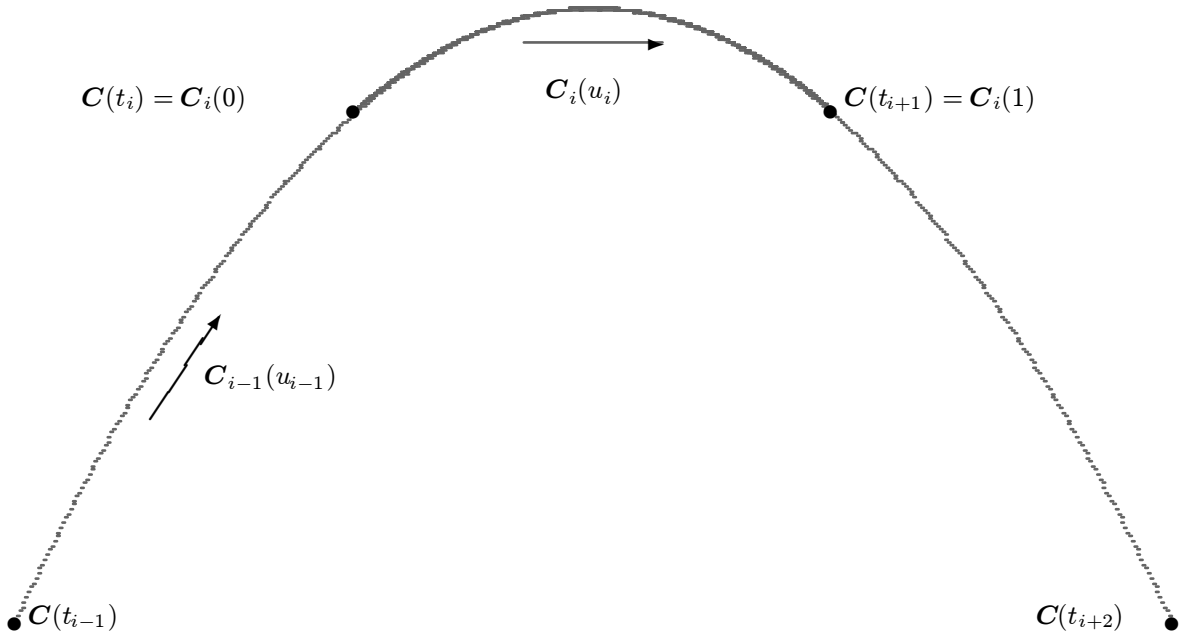


Figura 4.24 *Relación entre segmentos de curva, parámetros locales y parámetro global.*

donde $0 \geq u \geq 1$. Podríamos definir la curva como la unión de todos los segmentos:

$$\boldsymbol{C} = \bigcup_{i=0}^{k-1} \boldsymbol{C}_i.$$

El problema es que todos los segmentos están definidos para un mismo parámetro u. Por lo tanto, es necesario utilizar algún mecanismo que permita describir a la curva con un único parámetro global, el cual está relacionado con los parámetros locales en cada segmento polinomial.

Observemos que los segmentos polinomiales se unen entre sí en ciertos puntos definidos para formar la curva. Dichos puntos, por analogía con Lagrange, se denominan *nudos*. El parámetro global u asume el valor t_i al pasar por cada uno de dichos nudos. La secuencia $t_0, t_1, \cdots, t_k$ de valores del parámetro global asociados a cada uno de los nudos es no decreciente. Se define entonces un *parámetro local* u_i para cada segmento polinomial de curva $\boldsymbol{C}_i$. u_i varía entre 0 y 1 mientras u varía entre t_i y t_{i+1}. El punto que ocupa la curva cuando el valor del parámetro global es el valor del nudo t_i tiene que coincidir con el comienzo del i-ésimo segmento de curva, es decir, $\boldsymbol{C}(t_i) = \boldsymbol{C}_i(0)$, y también con el final del $i-1$-ésimo segmento, es decir, $\boldsymbol{C}(t_i) = \boldsymbol{C}_{i-1}(1)$ (ver Figura 4.24).

El cálculo del valor del parámetro local en función del valor del parámetro global y de los valores en los nudos se efectúa por interpolación lineal:

$$u_i = \frac{u - t_i}{t_{i+1} - t_i}.$$

De esa manera, podemos pasar de la representación global de la curva $\boldsymbol{C}(u)$ a las representaciones locales (es decir, definidas en cada uno de los segmentos polinomiales). Por ejemplo, si la secuencia de nudos son los enteros no negativos, entonces dado el un valor u del parámetro global, el segmento

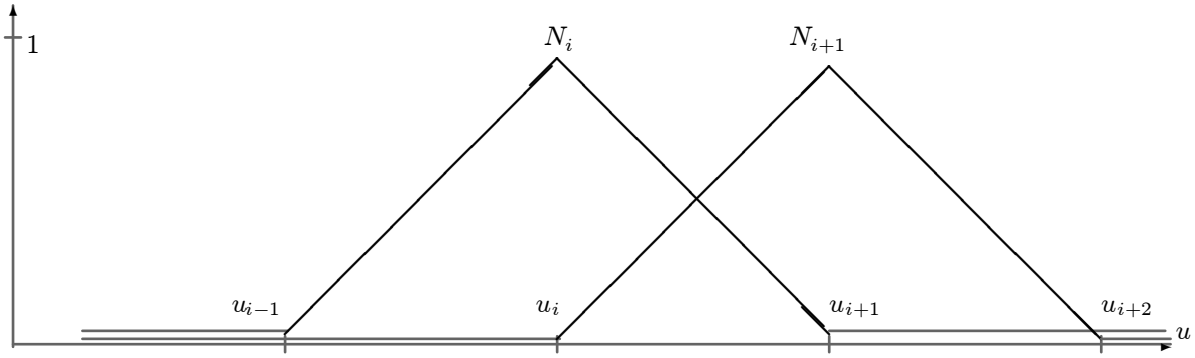


Figura 4.25 *Las bases B-Splines de primer grado.*

polinomial "activo" es $\boldsymbol{C}_i$, donde i es la parte entera de u, y el valor del parámetro local u_i es la parte fraccionaria de u, es decir $u - i$.

4.4.2 La base de Splines

Buscaremos ahora una formulación en términos de una base funcional para nuestro ejemplo de poligonal como "curva" polinomial a trozos, y luego extenderemos dicha formulación para grados superiores. Cada $\boldsymbol{C}_i$ es una combinación convexa de los puntos p_i y p_{i+1}:

$$\boldsymbol{C}_i(u_i) = (1 - u_i)p_i + u_i p_{i+1}.$$

Recordando la relación entre parámetro global y parámetros locales podemos reescribir la combinación convexa como:

$$\boldsymbol{C}_i(u) = \frac{t_{i+1} - u}{t_{i+1} - t_i} p_i + \frac{u - t_i}{t_{i+1} - t_i} p_{i+1},$$

con $t_i \leq u \leq t_{i+1}$.

El punto p_i contribuye al segmento de curva $\boldsymbol{C}_{i-1}(u)$ con un factor

$$\frac{u - t_{i-1}}{t_i - t_{i-1}},$$

y al segmento de curva $\boldsymbol{C}_i(u)$ con un factor

$$\frac{t_{i+1} - u}{t_{i+1} - t_i}.$$

El factor que aporta p_i puede pensarse como una función $N_i(u)$, la cual es, en efecto, una base funcional para funciones lineales con soporte local, es decir, distinta de cero solo en un subconjunto conexo del dominio. La definición de $N_i(u)$ es entonces la de una función lineal a trozos (ver Figura 4.25):

$$N_i(u) = \begin{cases} \dfrac{u - t_{i-1}}{t_i - t_{i-1}} & \text{si } t_{i-1} \leq u \leq t_i, \\ \dfrac{t_{i+1} - u}{t_{i+1} - t_i} & \text{si } t_i \leq u \leq t_{i+1}, \\ 0 & \text{en todo otro caso.} \end{cases} \tag{4.4}$$

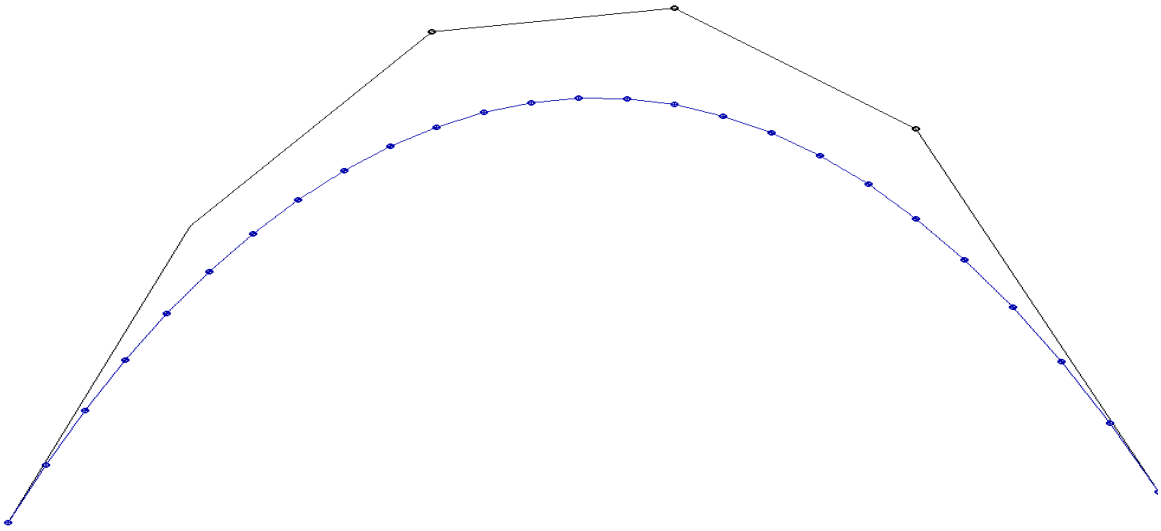


Figura 4.26 *Curva B-Splines de grado 5.*

Entonces nuestra "curva" queda expresada como

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} N_i(u) p_i.$$

Lo importante de la formulación en Splines es que la base de funciones está compuesta por una *única* función, cuyo soporte local es trasladado (ver Figura 4.25).

Para curvas grado $k > 1$, la idea general es encontrar una familia de bases $N_i^k(u)$ con soporte local, respetando la formulación de la curva como producto de una base funcional por un conjunto de puntos. La derivación de la familia $N_i^k(u)$ es relativamente compleja. Puede hacerse por integración sucesiva, lo cual lleva a una expresión recursiva, bastante similar a la encontrada en la Sección anterior para de Casteljau-Bézier:

$$N_i^k(u) = \frac{u - t_i}{t_{i+k-1} - t_i} N_i^{k-1}(u) + \frac{t_{i+k} - u}{t_{i+k} - t_{i+1}} N_{i+1}^{k-1}(u), \tag{4.5}$$

con

$$N_i^1(u) = \begin{cases} 1 & \text{si } t_i \leq u \leq t_{i+1}, \\ 0 & \text{en todo otro caso.} \end{cases}$$

$N_i^1(u)$ es idéntica a la función $N_i(u)$ vista en la ecuación 4.4. Un ejemplo de curva B-Spline de grado quinto se puede observar en la Figura 4.26.

Por el momento asumimos que la secuencia de nudos t_i es uniforme, en particular que $t_i = i$, aunque luego veremos los efectos de alterar esta uniformidad. Para obligar a la curva a pasar por el primer y último punto de control, se repiten los mismos la cantidad necesaria de veces (según el grado de la curva), por medio de asignar un valor $t_i = 0$ cuando $i < 0$, y un valor $t_i = n - grado + 2$ cuando $i > n$. También luego veremos los efectos de alterar estas restricciones.

```
const pts_ctrol=10; {n=10, luego 11 puntos}
      pasos=20;     {cantidad de evaluaciones}
      grado=3;      {grado deseado de la curva}

function nudo(i:integer):real;
begin
  if i < grado then nudo:=0
   else if i > pts_ctrol then nudo:=pts_ctrol-grado+2
     else nudo:=i-grado+1
end;

function base(u:real; i,k:integer):double;
var n1,n2,d1,d2,c1,c2:double;
begin
  if k=1 then
    if (u<nudo(i+1)) and (nudo(i)<=u) then base:=1
      else base:=0;
  else begin
    n1:=(u-i)*base(u,i,k-1);
    n2:=(nudo(i+k)-u)*base(u,i+1,k-1);
    d1:=nudo(i+k-1)-nudo(i);
    d2:=nudo(i+k)-nudo(i+1);
    if d1=0 then c1:=0 else c1:=n1/d1;
    if d2=0 then c2:=0 else c2:=n2/d2;
    base:=c1+c2;
  end;
end;
```

Figura 4.27 *Funciones para el cómputo de los valores de los nudos en la parametrización uniforme, y para el cómputo de la base de Splines recursiva.*

Los siguientes procedimientos muestran las implementaciones necesarias para computar las curvas B-Splines a partir de un grafo de control (asumimos vigentes muchas de las declaraciones de tipos de los trozos de código ya mostrados). En la Figura 4.27 podemos encontrar las funciones necesarias para computar los nudos para una parametrización uniforme, y la base recursiva de B-Splines, mientras que en la Figura 4.28 encontramos el procedimiento que utiliza las bases y la secuencia de puntos de control para computar la curva.

Podemos ver en la Figura 4.29 las funciones base de Splines de grado cúbico y de grado 10 utilizadas para aproximar un grafo de 11 puntos de control (comparar con los polinomios de Bernstein). Observar que para un grado k dado, desde $i = k - 1$ hasta $i = n - k + 1$ la base funcional $N_i^k(u)$, para distintos valores de i, es la misma función pero desplazada en u. El efecto del aumento del grado de la curva es producir una disminución de la influencia local de los puntos de control (ver Figura 4.30).

Es importante considerar el efecto que tiene aumentar el grado de una curva B-Spline con respecto a variaciones locales en las posiciones de los puntos de control. En la Figura 4.31 se puede observar de qué manera el aumento del grado de la curva disminuye la influencia de un desplazamiento local (comparar con el método de Bézier).

```
procedure bsplines(var g:grafo_contr);
var i,j:integer;
    u,v:real;
    p:punto;
begin
 for i:=0 to pasos*(pts_ctrol-grado+2) do begin
   u:=(i/pasos);
   p.x:=0;  p.y:=0;
   for j:=0 to pts_ctrol do begin
    v:=base(u,j,grado);
    p.x:=p.x + g[j].x*v;
    p.y:=p.y + g[j].y*v;
    end;
   graf_linea(p);
 end;
end;
```

Figura 4.28 *Procedimiento que calcula la curva a partir de la base de Splines y de la secuencia de puntos de control.*

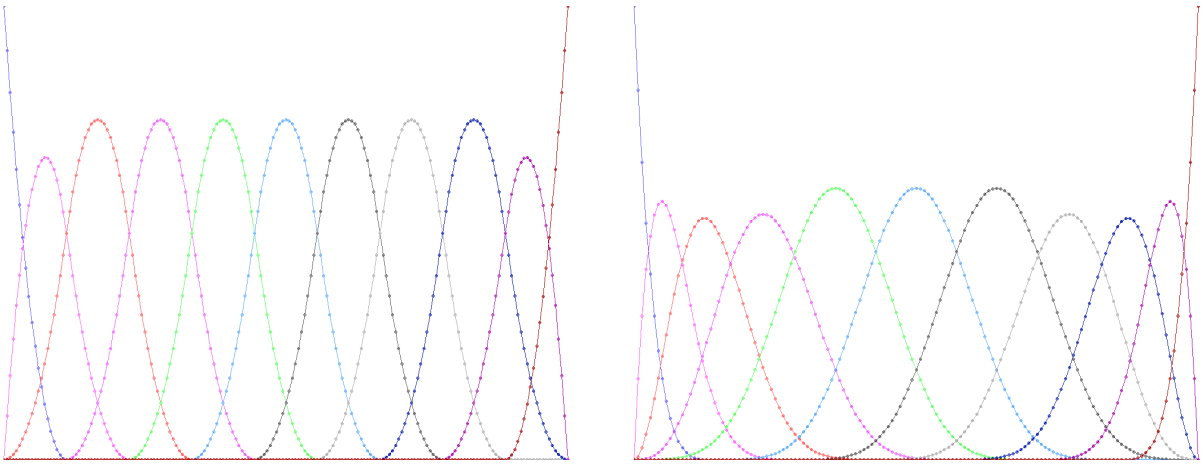


Figura 4.29 *La base funcional de Splines de grados 3 y 5 para 11 puntos de control ($n = 10$).*

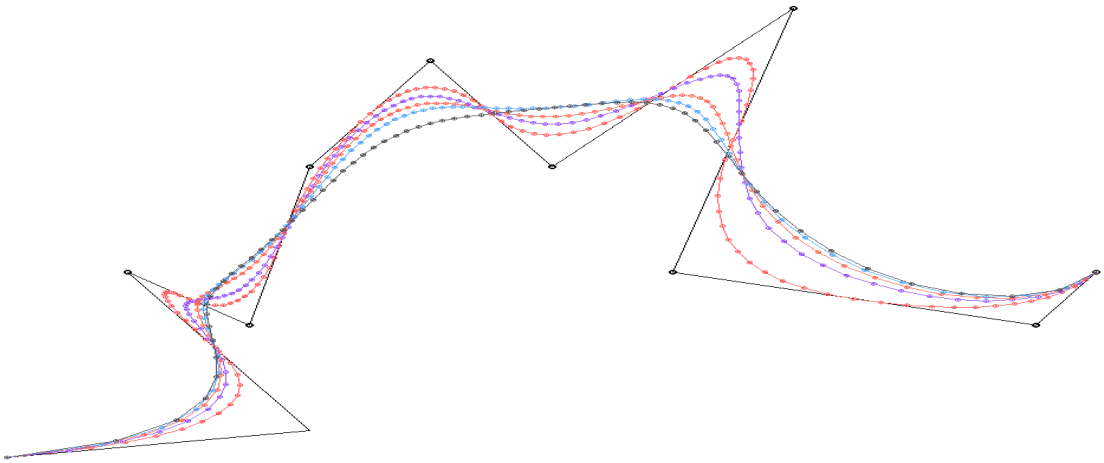


Figura 4.30 *Curvas B-Splines de grado 2, 3, 4 y 5 aproximando a un mismo grafo de control.*

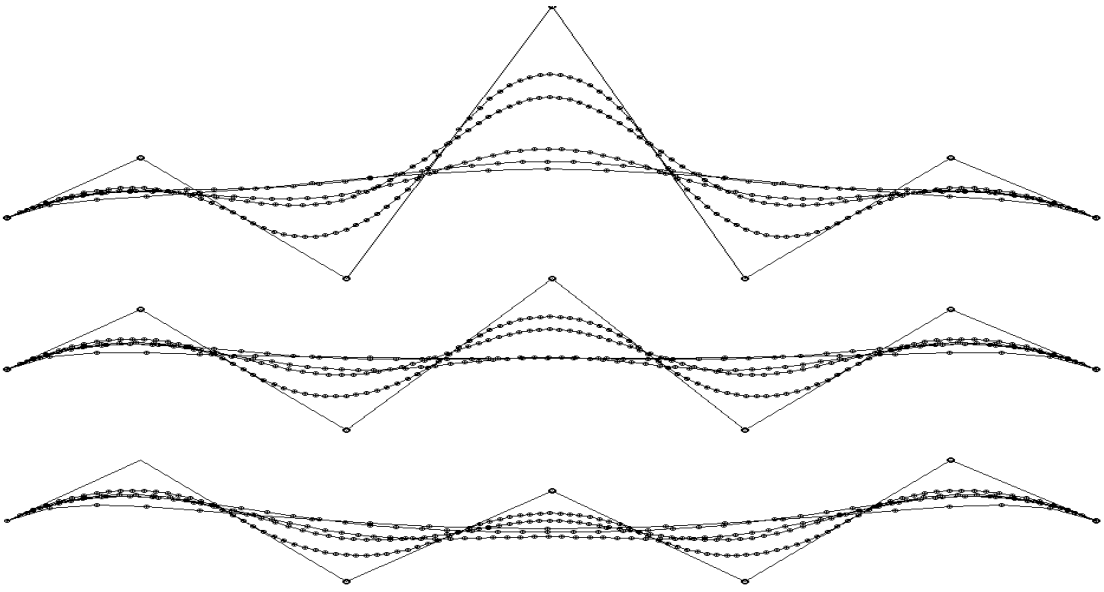


Figura 4.31 *Curvas B-Splines de grado 2, 3, 4 y 5 aproximando un grafo de control en el cual varía la posición de un punto.*

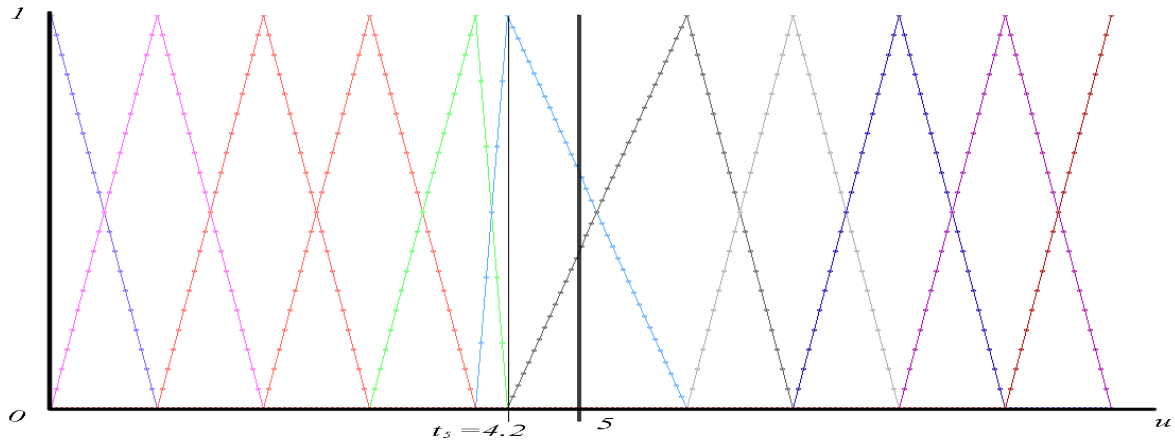


Figura 4.32 *Bases de Spline de primer grado modificando el valor de un nudo.*

Al mismo tiempo, y al igual que en el método de Bézier, es posible obtener curvas periódicas. Esto se consigue por medio de una asignación periódica a los índices de los puntos de control asociados a los nudos. De esa manera, se varía i entre $-k$ y $n+k$, utilizándose valores $t_i = i$ para los nudos, y la asignación periódica $p_{i \bmod n}$. Por último, también pueden repetirse puntos de control, aunque dado el control local de las curvas B-Splines, el efecto puede ser muy pronunciado.

4.4.3 B-Splines no uniformes

¿Cuál es el resultado de alterar los valores en los nudos, respecto de la asignación uniforme $t_i = i$? El efecto producido es el de alterar la velocidad paramétrica en la curva, es decir, cuánto se desplaza un punto $p(u)$ respecto de un cambio en el valor de u. Supongamos tener una "curva" B-Spline de grado 1 con la parametrización uniforme mencionada más arriba. Las bases funcionales de esta "curva" son las funciones triangulares de la Figura 4.25. Modificar el valor t_i asociado al i-ésimo punto de control implica desplazar el valor donde ocurre el máximo de la i-ésima base funcional (ver Figura 4.32).

En este caso, el resultado de asignar a t_i un valor menor que i (pero mayor que $i-1$, dado que la secuencia de nudos es no decreciente) es que la "curva" es recorrida más rápidamente que "lo normal" (el caso uniforme) con respecto al parámetro entre los puntos de control p_{i-1} y p_i, y más lentamente entre los puntos de control p_i y p_{i+1}. Pero el aspecto geométrico se mantiene constante. Sin embargo, cuando la curva es de un grado mayor, entonces es posible apreciar un efecto en las bases, dado que éstas provienen en definitiva de un promedio o integración ponderada de las bases lineales (ver Figura 4.33).

El resultado de disminuir o aumentar el valor de un nudo es similar a "acelerar" o "frenar" la curva cuando ésta pasa por el punto correspondiente a dicho nudo $p(t_i)$. Dicho efecto puede observarse en la Figura 4.34, donde se observa el resultado de aumentar o disminuir el valor de un nudo en la familia de bases. El resultado geométrico de dichas modificaciones puede observarse en la Figura 4.36, donde las curvas B-Splines no uniformes aproximan a un mismo grafo de control, pero modificando el valor del parámetro en el nudo asociado a un punto de control.

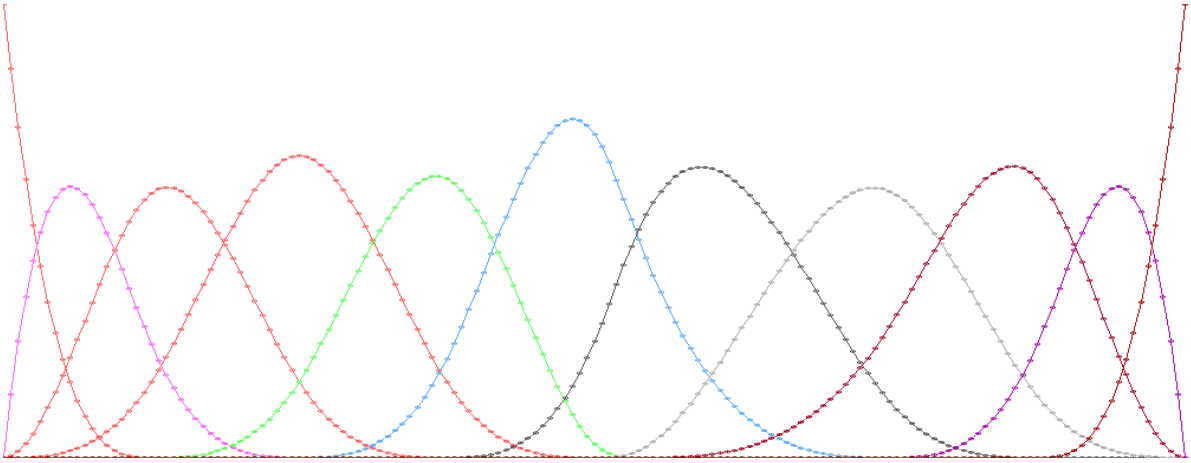


Figura 4.33 *Bases de Spline de tercer grado (con $n = 10$) modificando el valor de un nudo.*

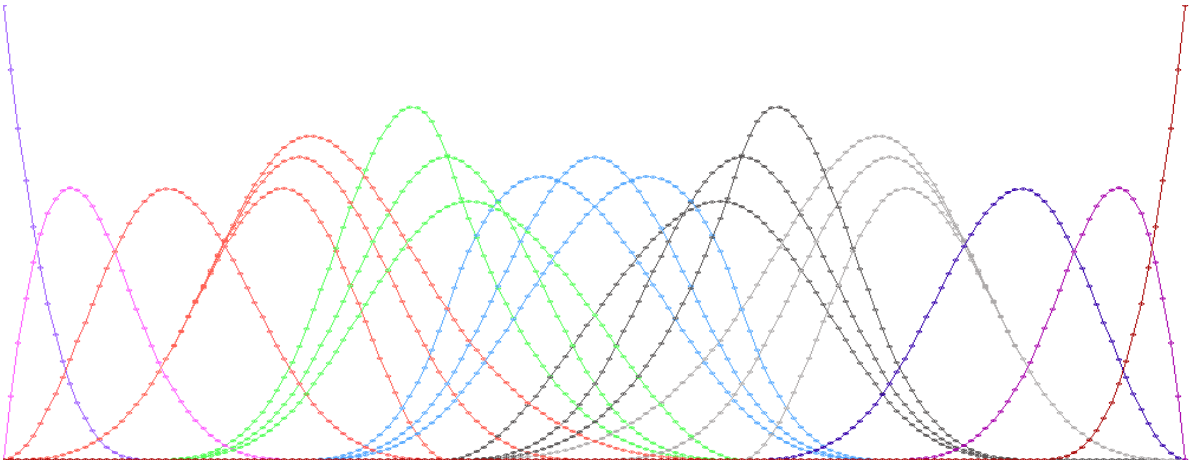


Figura 4.34 *Bases de Spline de tercer grado (con $n = 10$) aumentando y disminuyendo el valor de un nudo.*

```
...
var t: array [0..pts_ctrol-grado+2] of real;
...
function nudo(i:integer):real;
begin
  if i < grado then nudo:=t[0]
   else if i > pts_ctrol then nudo:=t[pts_ctrol-grado+2]
     else nudo:=t[i-grado+1]
end;
...
```

Figura 4.35 *Implementación de parametrizaciones no uniformes por medio de un arreglo de nudos.*

Está claro que pueden asignarse valores totalmente arbitrarios a los nudos, (inclusive a los que corresponden a la parte que antecede y sucede a los nudos asociados al grafo de control), siempre que se respete la condición de que la secuencia de nudos sea no decreciente. Una forma de implementar computacionalmente dicha secuencia es por medio de un arreglo auxiliar en vez de una función, como se muestra en la Figura 4.35.

Recordamos que el valor de un nudo t_i es el valor que debe asumir el parámetro u cuando la curva pasa de un segmento polinomial al siguiente. Por lo tanto, la secuencia de nudos de una curva se denomina usualmente la *parametrización* de la curva, y permite obtener una gran variedad de resultados a partir de un mismo grafo de control. Sin embargo, no siempre es predecible o intuitivo el efecto que produce una determinada parametrización.

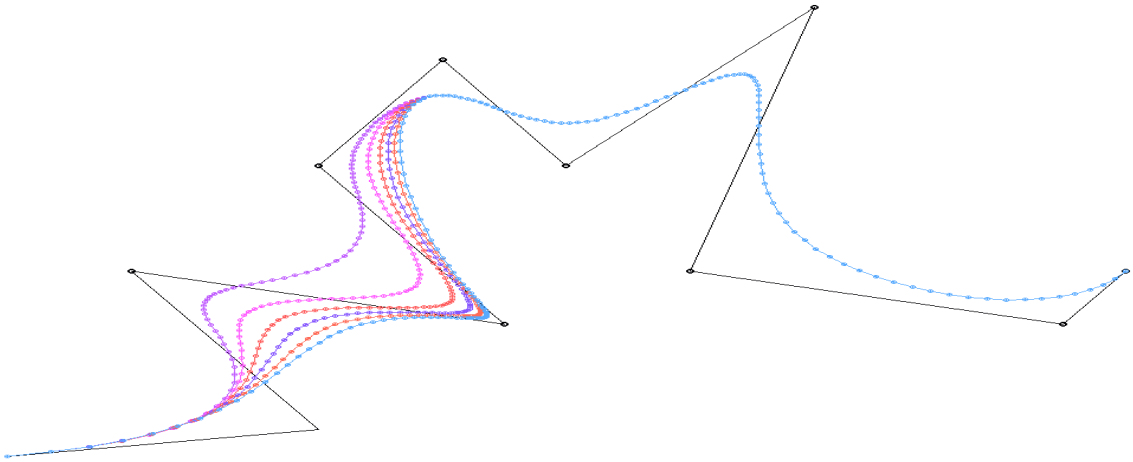


Figura 4.36 *Curva B-Spline de tercer grado disminuyendo y aumentando el valor de un nudo.*

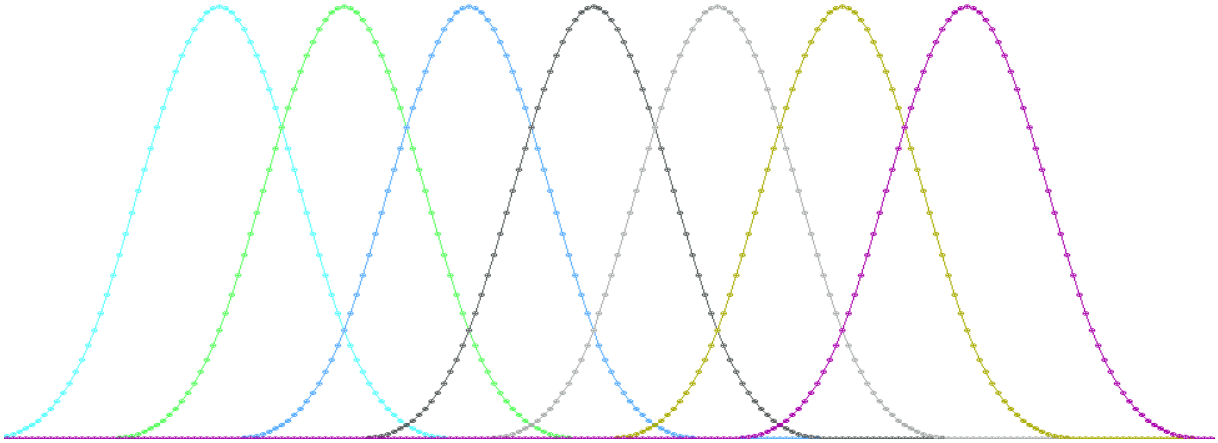


Figura 4.37 *La base funcional $N_i^3(u)$ ignorando las primeras y últimas tres.*

4.4.4 B-Splines cúbicos uniformes

En la búsqueda de un método de aproximación geométricamente versátil y computacionalmente económico, se plantean casos particulares para la ecuación 4.5 de modo tal que su expresión se simplifique. Específicamente se elige el menor grado que produzca resultados satisfactorios, junto con una parametrización uniforme que normalmente coincide con la secuencia de enteros no negativos. Lo usual es elegir $k = 3$ de modo de tener orden de continuidad $k - 1 = 2$. Esto implica que cada segmento polinomial queda definido a partir de la ubicación de $k + 1 = 4$ puntos de control. Este caso particular se conoce como B-Splines cúbicos uniformes, para los cuales derivaremos una formulación polinomial sencilla y un esquema computacional muy eficiente.

En este caso, las funciones de la base tienen que tener soporte entre $k + 1 = 4$ nudos sucesivos. Es decir, cada punto ejerce su influencia en cuatro segmentos polinomiales. Supongamos que la función $N_i^3(u)$ asociada a p_i tiene una forma como la mostrada en la Figura 4.37, y que la base asociado a todo otro punto de control es la misma, pero desplazada en u.

Entonces $N_i^3(u_i)$ está compuesta por cuatro segmentos funcionales (o sub-bases), etiquetados $b_1(u)$, $b_0(u)$, $b_{-1}(u)$, y $b_{-2}(u)$, respectivamente, por razones que veremos a continuación. En un intervalo entre dos nudos sucesivos, solo cuatro puntos influencian la posición de la curva. Es decir, cada segmento polinomial es una combinación convexa de cuatro puntos sucesivos, cada uno participando con una sub-base distinta (ver Figura 4.38).

Por dicha razón las sub-bases adoptan los curiosos nombres $b_1(u)$, $b_0(u)$, $b_{-1}(u)$, $b_{-2}(u)$. $b_1(u)$ es la sub-base activa asociada a p_{i+1}, $b_0(u)$ es la sub-base activa asociada a p_i, $b_{-1}(u)$ es la sub-base activa asociada a p_{i-1}, y $b_{-2}(u)$ es la sub-base activa asociada a p_{i-2}. Es decir

$$\boldsymbol{C}_i(u_i) \sum_{k=-2}^{1} b_k(u_i) p_{i+k}.$$

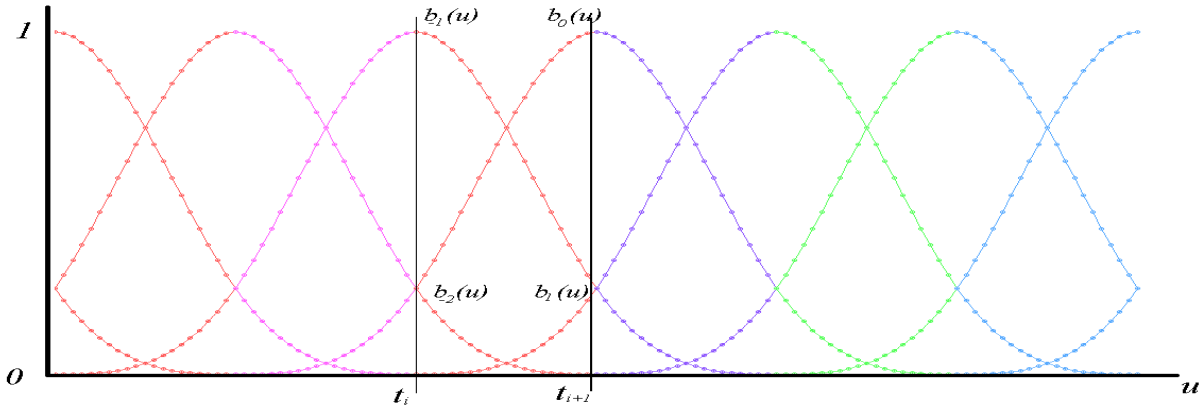


Figura 4.38 *La base funcional cúbica, y las cuatro sub-bases activas entre dos valores del parámetro global.*

Para encontrar las sub-bases podemos plantear las siguientes condiciones de continuidad.

$$\begin{array}{lll} b_1(0) = 0 & \dot{b}_1(0) = 0 & \ddot{b}_1(0) = 0 \\ b_1(1) = b_0(0) & \dot{b}_1(1) = \dot{b}_0(0) & \ddot{b}_1(1) = \ddot{b}_0(0) \\ b_0(1) = b_{-1}(0) & \dot{b}_0(1) = \dot{b}_{-1}(0) & \ddot{b}_0(1) = \ddot{b}_{-1}(0) \\ b_{-1}(1) = b_{-2}(0) & \dot{b}_{-1}(1) = \dot{b}_{-2}(0) & \ddot{b}_{-1}(1) = \ddot{b}_{-2}(0) \\ b_{-2}(1) = 0 & \dot{b}_{-2}(1) = 0 & \ddot{b}_{-2}(1) = 0 \end{array}$$

Es decir, tenemos cinco condiciones de continuidad en posición, en primera derivada y en segunda derivada. Cada sub-base es un polinomio cúbico, con cuatro incógnitas. Tenemos 16 incógnitas y 15 ecuaciones. La condición faltante es que

$$b_1(u) + b_0(u) + b_{-1}(u) + b_{-2}(u) = 1 \text{ para } 0 \leq u < 1$$

Resolviendo las ecuaciones obtenemos:

$$\begin{aligned} b_1(u) &= \frac{1}{6}u^3, \\ b_0(u) &= \frac{1}{6}(1 + 3u + 3u^2 - 3u^3), \\ b_{-1}(u) &= \frac{1}{6}(4 - 6u^2 + 3u^3), \\ b_{-2}(u) &= \frac{1}{6}(1 - 3u + 3u^2 - u^3). \end{aligned}$$

De esa manera, cada punto de control p_i tiene asociada una base compuesta por cuatro sub-bases no nulas, que determinan su influencia en la curva $\boldsymbol{C}(u)$. Dados $n+1$ puntos de control $p_0, p_1, \cdots, p_n$, la representación de la curva de B-Splines cúbica es:

$$\boldsymbol{C}(u) = \bigcup_{i=2}^{n-1} \boldsymbol{C}_i(u),$$

```
function base(u:real;i:integer):double;
begin
 case i of
  1 : base:=u*u*u/6;
  0 : base:=(1+u*3*(1+u*(1-u)))/6;
 -1 : base:=(4+u*u*3*(-2+u))/6;
 -2 : base:=(1+u*(-3+u*(3-u)))/6;
 end;
end;

procedure bsplines(var g:grafo_contr);
var i,j,k,index:integer;
    u,v:real;
    p:punto;
begin
   p.z:=1;     p.w:=1;
    for k:=1 to pasos do begin
      u:=k/pasos;
      p.x:=0;  p.y:=0;
      for i:=-2 to 1 do begin
        v:=base(u,i);
        index:=i+j;
        if index<0 then index:=0
          else if index>pts_ctrol then index:= pts_ctrol;
        p.x:=p.x+g[index].x*v;
        p.y:=p.y+g[index].y*v;
        end;
       graf_linea(p);
      end;
   end;
end;
```

Figura 4.39 *Implementación de curvas Spline cúbicas uniformes por medio de sub-bases.*

lo cual es equivalente a una expresión como sumatoria (al estilo de la formulación de Bézier):

$$\boldsymbol{C}(u) = \sum_{i=2}^{n-1} N_i^3(u) p_i.$$

Por otra parte, teniendo en cuenta que las funciones base tienen soporte solamente en cuatro segmentos polinomiales, la expresión como sumatoria es ineficiente, siendo preferible expresarla como unión de los segmentos

$$\boldsymbol{C}_i(u_i) \sum_{k=-2}^{1} b_k(u_i) p_{i+k}.$$

El trozo de código de la Figura 4.39 muestra una implementación de curvas B-Splines cúbicas uniformes. En la Figura 4.40 se observan curvas B-Spline cúbicas uniformes donde cada segmento polinomial fue computado con el algoritmo mostrado más arriba (y graficado en distinto color),

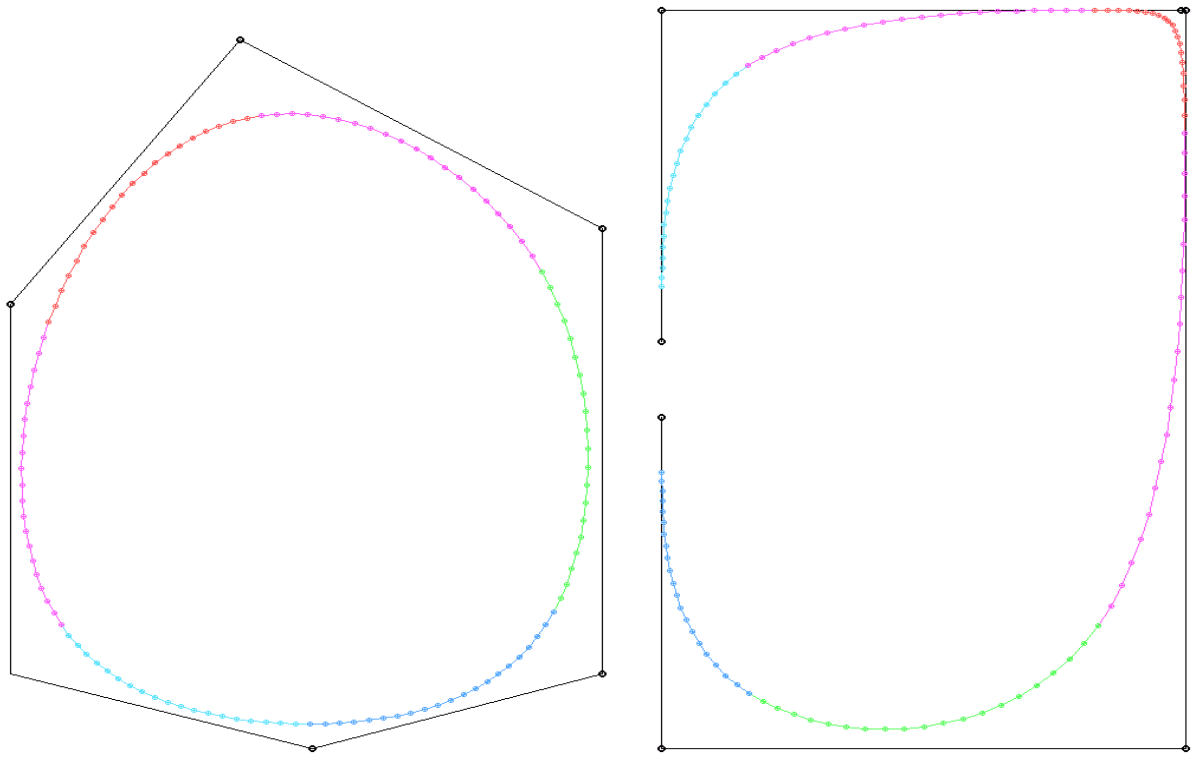


Figura 4.40 *Curvas B-Splines de grado 3. La curva de la izquierda es periódica por medio de la asignación $p_{i \bmod n}$, mientras que la curva de la derecha tiene un punto de control repetido (levemente separado para que se aprecie su ocurrencia).*

tomando como ejemplo una curva cerrada utilizando asignación periódica, y un grafo de control con puntos de control repetidos.

Es posible observar que las curvas B-Spline no pasan por el primer y último punto de control. En efecto, dada una secuencia de puntos de control $p_0, p_1, \cdots, p_n$, el segmento de curva

$$\boldsymbol{C}_2(u) = b_{-2}(u)p_0 + b_{-1}(u)p_1 + b_0(0)p_2 + b_1(u)p_3$$

comienza en

$$\boldsymbol{C}_2(0) = b_{-2}(0)p_0 + b_{-1}(0)p_1 + b_0(0)p_2 + b_1(0)p_3 = \frac{1}{6}p_0 + \frac{4}{6}p_1 + \frac{1}{6}p_2 + 0p_3.$$

El segmento $\boldsymbol{C}_1$ no está definido porque no tenemos punto de control p_{-1}. Lo mismo sucede con $\boldsymbol{C}_n(u)$. En general, están definidos los segmentos $\boldsymbol{C}_2$ a $\boldsymbol{C}_{n-1}$ (ver Figura 4.41).

Para garantizar que la curva comience en p_0 podemos *repetir* dicho punto. Repitiendo una sola vez, es decir, haciendo que la secuencia comienze con $p_{-1} = p_0$, obtenemos la posibilidad de que $\boldsymbol{C}_1(u)$ esté definida, y

$$\boldsymbol{C}_1(0) = b_{-2}(0)p_0 + b_{-1}(0)p_0 + b_0(0)p_1 + b_1(0)p_2 = \frac{1}{6}p_0 + \frac{4}{6}p_0 + \frac{1}{6}p_1 + 0p_2.$$

Es decir, la curva comienza en un punto que está a $\frac{1}{6}$ de p_0 y a $\frac{5}{6}$ de p_1. La única forma de que la curva pase por p_0 es repetirlo.

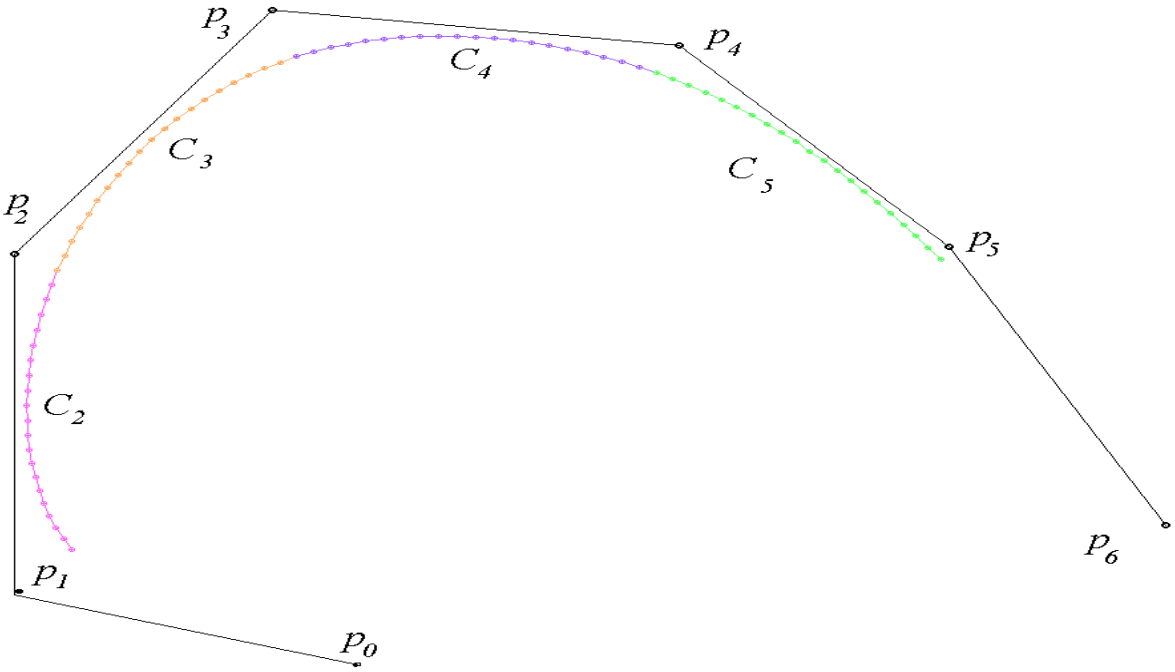


Figura 4.41 *Los segmentos definidos de una curva B-Spline cúbica uniforme.*

4.4.5 Evaluación de B-Splines cúbicos

La representación de curvas por medio de B-Splines cúbicos uniformes quedó determinada por un esquema en el cual cada punto de la curva está determinado por la suma convexa de cuatro puntos de control, donde el coeficiente de cada punto es un polinomio cúbico. Por lo tanto, se requiere un esquema eficiente para evaluar funciones de la forma

$$p(u) = au^3 + bu^2 + cu + d,$$

con $0 \leq u \leq 1$.

Es posible encontrar dicha evaluación económica utilizando una metodología similar a la evaluación por diferencias finitas (DDA) como hicimos en el Capítulo 2 con rectas y circunferencias. Sea $n+1$ la cantidad de evaluaciones y sea el incremento en cada evaluación el valor

$$\delta = \frac{1}{n}.$$

La diferencia entre dos evaluaciones sucesivas

$$\Delta_1(i\delta) = p((i+1)\delta) - p(i\delta)$$

es una función de grado 2. Lo mismo puede hacerse para diferencias de mayor orden:

$$\Delta_k(i\delta) = \Delta_{k-1}((i+1)\delta) - \Delta_{k-1}(i\delta),$$

hasta llegar a una función constante, en este caso

$$\Delta_3(i\delta) = \Delta.$$

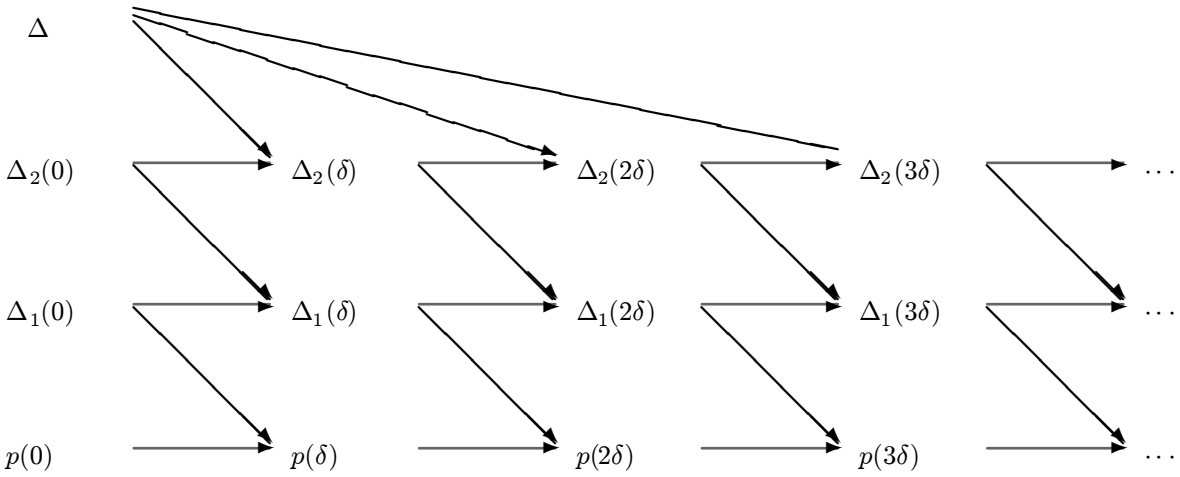


Figura 4.42 *Evaluación de un polinomio cúbico por DDA.*

De esa manera, conociendo Δ y $\Delta_2((i-1)\delta)$ podemos obtener $\Delta_2(i\delta)$, para todo i tal que $0 \leq i \leq n$. Igualmente, con $\Delta_2(i\delta)$ y $\Delta_1((i-1)\delta)$ obtenemos $\Delta_1(i\delta)$, $0 \leq i \leq n$, y con $\Delta_1(i\delta)$ y $p((i-1)\delta)$ obtenemos $p(i\delta)$, $0 \leq i \leq n$.

Por lo tanto, solamente es necesario conocer p y los distintos Δ en $u = 0$. Un rápido análisis nos permite encontrar

$$\begin{aligned} p(0) &= d, \\ \Delta_1(0) &= p(\delta) - p(0) &&= a\delta^3 + b\delta^2 + c\delta, \\ \Delta_2(0) &= \Delta_1(\delta) - \Delta_1(0) &&= 6a\delta^3 + 2b\delta^2, \\ \Delta_3 &= \Delta_2(\delta) - \Delta_2(0) &&= 6a\delta^3. \end{aligned}$$

De esa manera, es posible computar un polinomio cúbico evaluado a intervalos regulares entre 0 y 1, utilizando solamente tres sumas (ver Figura 4.42).

4.5 Ejercicios

1. Implementar los algoritmos de aproximación e interpolación de curvas de Hermite, Bézier (por de Casteljau y por Bernstein) y B-Spline cúbicos uniformes.

2. Aplicar los métodos a un grupo de grafos de control de prueba (por ejemplo, una "onda cuadrada", una curva cerrada o para aproximar un mapa de la Provincia). Comparar los resultados obtenidos y el tiempo insumido, y encontrar ventajas y desventajas de cada uno.

3. Dibuje su propia firma. ¿Qué método utilizó? ¿Por qué? ¿Cúantos puntos de control fueron necesarios y cuánto tiempo de trabajo requirió encontrarlos? ¿Qué se concluye?

4. Implementar los B-Splines cúbicos uniformes por medio de curvas de Bézier cúbicas. Comparar resultados y tiempos con la implementación de B-Splines propiamente dichos.

5. Implementar B-Splines cúbicos no uniformes por medio de curvas de Bézier. Dado un mismo grafo de control, asignar parametrizaciones no uniformes y comparar los resultados.

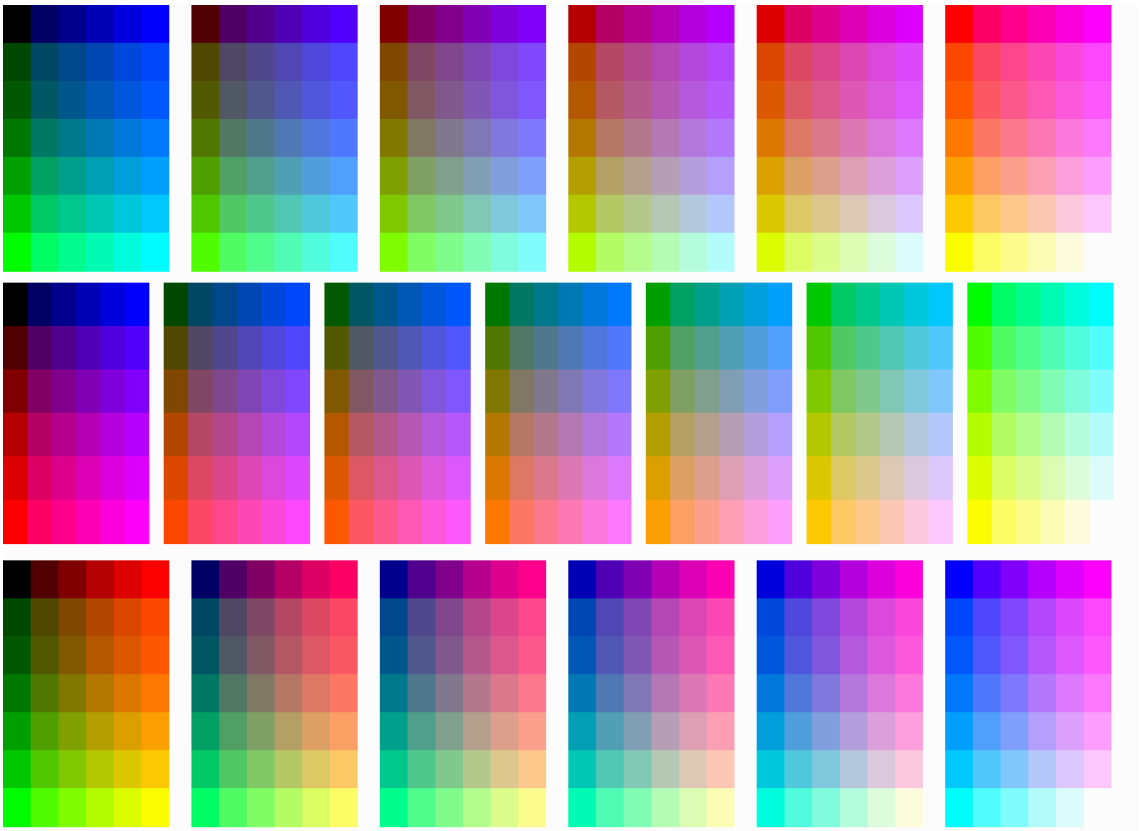
6. Implementar Bézier racionales y NURBS. Aplicar a los mismos problemas que en los ejercicios 2 y 3. Comentar los resultados.

4.6 Bibliografía recomendada

El tratamiento de este tema es un poco superficial en los textos clásicos de Computación Gráfica. Al mismo tiempo, el gran desarrollo que ha tenido hace que los resultados recientes figuren casi exclusivamente en artículos en revistas o en libros específicos del tema. De todas maneras, existen libros dedicados exclusivamente a curvas y superficies para CAGD que comienzan desde un nivel adecuadamente introductorio y presentan todo lo necesario para conocer el material clásico sobre curvas y superficies de Bézier y B-Splines. La recomendación básica es el libro de Farin [30], el cual es exhaustivo y presenta los temas de relevancia de una manera clara y definitiva. Otro libro recomendable es el de Bartels, Barski y Beatty [6], que sin ser tan riguroso y preciso presenta algunos temas avanzados desarrollados por los autores, y que no es posible encontrar en otros textos.

Por razones de espacio hemos omitido de este Capítulo las curvas racionales y los β-Splines. Recomendamos a los interesados que consulten [26]. Un buen material introductorio al tema de las curvas racionales puede leerse en [70], donde se presenta la interpretación proyectiva de las curvas de Bézier racionales y se deriva el sistema de coordenadas baricéntrico asociado a las bases de Bernstein racionales. El mismo autor presenta en [71] una introducción a las curvas y superficies NURBS. Los β-Splines son un desarrollo relativamente reciente. La discusión acerca de la continuidad paramétrica *versus* la continuidad geométrica está presentada en [5]. Los β-Splines son comparados con los métodos clásicos en [3], y la derivación de la ecuación del control local para los parámetros de forma está en [4].

5 El Color en Computación Gráfica



5.1 Introducción

El descenso en el costo de los monitores determina que en la actualidad prácticamente en todas las aplicaciones de Computación Gráfica se utiliza intensivamente el color, como medio de representación (por ejemplo, asociar una escala de colores con temperaturas o altitudes), con fines "cosméticos", para mejorar la apariencia y legibilidad de una interfase gráfica, o para dar realismo a la representación de objetos en una escena tridimensional sintética. El color es un elemento indispensable en las aplicaciones en las que se busca una representación con realismo, porque la simulación de los fenómenos ópticos que ocurren en los objetos reales (reflexión, refracción, etc.) se realiza en forma adecuada a través de una ecuación de iluminación (ver Capítulo 7). Dicha ecuación se basa implícitamente en un modelo de color, cromático o monocromático (escala de grises).

El tema del color es rico y complejo, involucrando conceptos de física, fisiología y psicología, además de los aspectos computacionales que son de nuestro interés. En este Capítulo dedicaremos las primeras Secciones a presentar los elementos fundamentales de la teoría del color, para luego introducirnos en los espacios cromáticos, es decir, espacios que permiten una representación sistemática de los colores, y por último a la representación del color en las tarjetas gráficas.

5.2 Aspectos Físicos del Color

El color es una sensación fisiológica producida por la acción de ondas electromagnéticas específicas en los receptores nerviosos de la retina. Es decir, su origen primario es en una distribución de energía electromagnetica en el espectro visible. Sin embargo, es una magnitud relativa, dado que dicho espectro en general no depende de la radiación lumínica propia de los objetos. En general el color de un objeto no depende sólo de éste, sino también de las fuentes de luz que lo iluminan y de otras consideraciones que veremos en detalle en el Capítulo 7:

Iluminantes $\longrightarrow$ Objetos $\longrightarrow$ Observador

El observador, al mismo tiempo, impone sus características propias, dado que el estímulo producido por una misma distribución espectral de energía no siempre produce la misma percepción visual:

Estímulo $\longrightarrow$ Sensación $\longrightarrow$ Percepción

Así, pese a estar originado en una magnitud física objetiva (la longitud de onda de las radiaciones del estímulo) no es posible encontrar una definición intrínseca del color de un objeto, sino que todas las descripciones deben ser relativas a una condición ambiental y fisiológica dada. Una de las características menos problemáticas de comprender en el color es su componente acromática o *intensidad luminosa* (también llamada *luminancia* o luminosidad). La luz acromática es la producida, por ejemplo, en un televisor blanco y negro, en el cual cada punto visible tiene un único atributo (su luminosidad, variando entre 0 = negro y 1 = blanco). Un televisor tiene un continuo de niveles entre 0 y 1, mientras que un monitor de computadora tiene una cantidad limitada (por ejemplo, 2, 16, 256) y una impresora de puntos o laser corriente tiene solo 2.

¿Cuántos atributos debe tener un sistema de reproducción de color cromático? La evidencia experimental al respecto indica que con tres atributos es suficiente. Esta característica se expresa en la *primera ley de Grassman*: cualquier color puede representarse como combinación de tres atributos primarios. Estos atributos pueden ser distintos (como distintas bases en un espacio vectorial), siendo los más comunes en computación gráfica los tres niveles de los colores primarios

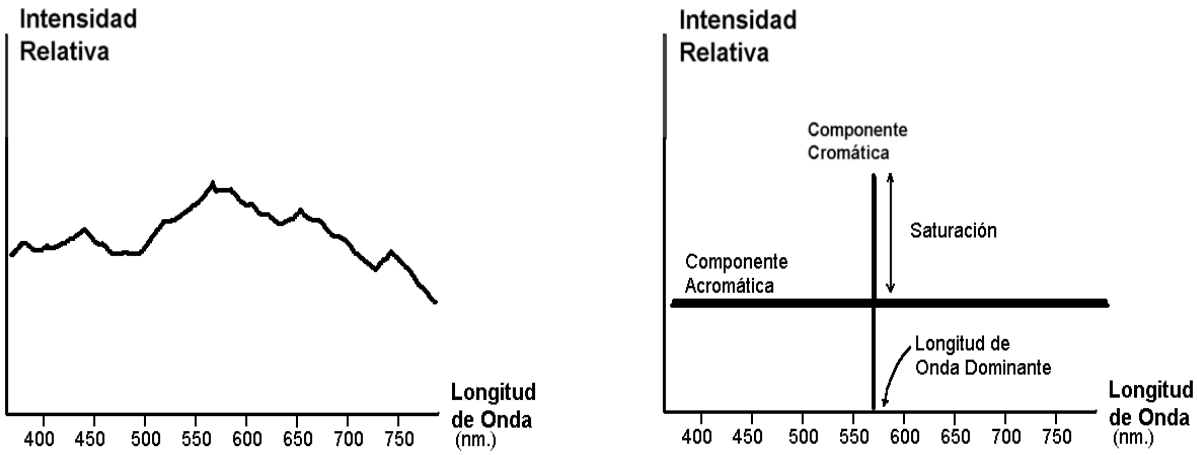


Figura 5.1 *Distribución espectral de un estímulo cromático y su equivalente metamérico.*

rojo, verde y azul, o *espacio cromático* RGB. En las Artes Plásticas se asumen otros colores primarios y también otros nombres referirse a los primarios.

En algunas ramas de la física (así como en TV color), sin embargo, los atributos considerados son la luminosidad, la crominancia y la saturación. La crominancia distingue la característica de "color" propiamente dicho en una determinada distribución (a qué "color puro" se parece el color), mientras que la saturación se refiere a la pureza del color. En términos físicos la luminosidad es análoga a la amplitud de energía de la onda luminosa (o a la cantidad de fotones) por unidad de tiempo. La crominancia es análoga a la longitud de onda dominante y la saturación a la relación entre la energía de la longitud de onda dominante y la energía total.

5.3 Aspectos Fisiológicos del Color

Muchas posibles distribuciones espectrales dan origen a sensaciones cromáticas idénticas. Si bien por un lado es posible que estímulos idénticos produzcan percepciones distintas en algunos casos, es también posible observar que estímulos radicalmente diferentes producen sistematicamente la misma percepción. Esta característica del aparato visual humano se denomina *metamerismo*, y es la que en definitiva determina que las sensaciones cromáticas puedan definirse con solo tres parámetros.

Dada una distribución espectral de energía lumínica en las frecuencias visibles, el sistema visual humano detecta la energía total (luminosidad), la longitud de onda dominante (crominancia) y la relación entre la energía de esta última con la energía total (saturación), y en base a ésto se elabora la sensación visual. Si el estímulo visual no tiene una longitud de onda dominante, entonces se percibe como luz acromática (blanca). De esa forma, es posible decir que cualquier estímulo luminoso tiene una componente acromática y una componente cromática pura. La crominancia es, entonces, la longitud de onda de la componente cromática del estímulo (excepto en algunos casos que veremos más adelante), y la saturación es la relación de energías entre la componente cromática y la acromática (ver Figura 5.1).

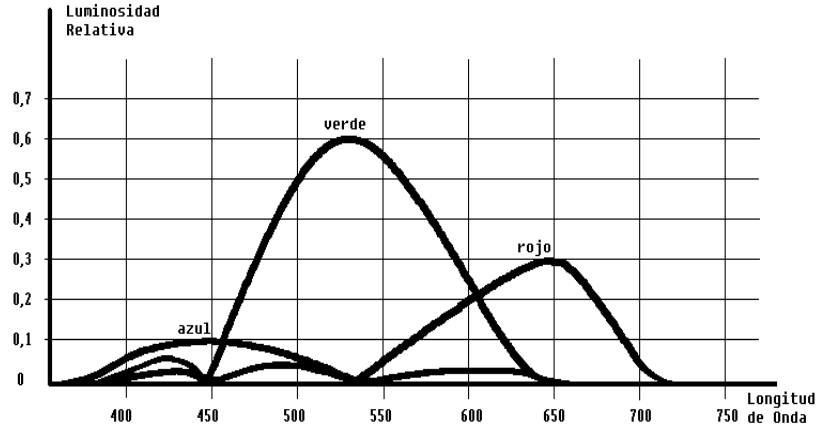


Figura 5.2 *Sensibilidad relativa de los receptores de la retina.*

Los aspectos físicos del color mencionados más arriba y los aspectos fisiológicos humanos se relacionan a través de la teoría triestímulo del color. Según esta teoría, existen en la retina tres tipos de receptores en los conos (células nerviosas sensibles a la luz) con sensibilidades centradas a los 650nm. (rojo), 530nm. (verde) y 425nm. (azul) respectivamente, y cuya sensibilidad relativa responde a las curvas de la Figura 5.2, obtenidas experimentalmente.

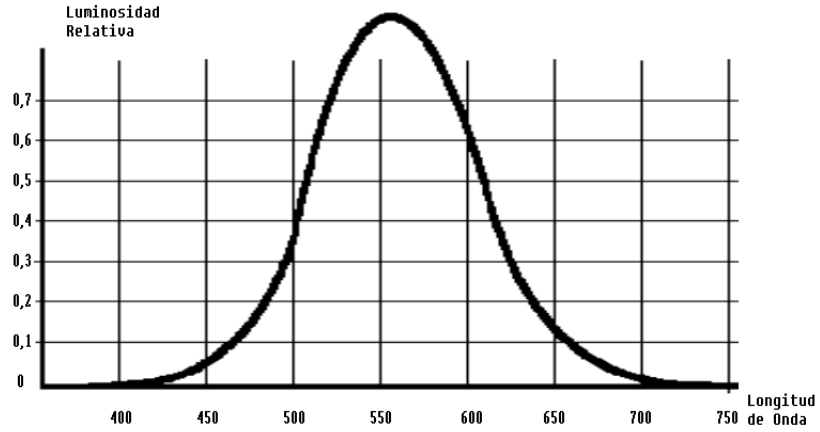


Figura 5.3 *Sensibilidad espectral del aparato visual humano.*

La suma de las sensibilidades de cada receptor corresponde a la curva de sensibilidad total del aparato visual humano (ver Figura 5.3). Lo importante del caso es que cualquier estímulo cromático puede sintetizarse con una combinación de iluminantes rojo, verde, y azul (llamados por ende los colores iluminantes primarios).

Este modelo de síntesis cromática falla, sin embargo, en el caso de los colores complementarios a los primarios (cyan, magenta, y amarillo, respectivamente), cuando estos últimos se encuentran

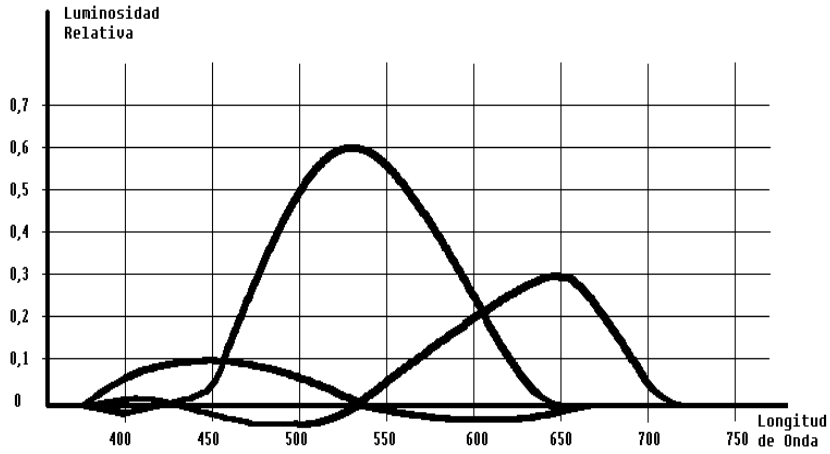


Figura 5.4 *Refinamiento de las curvas de sensibilidad relativa, teniendo en cuenta la respuesta "negativa" de los receptores.*

muy saturados. Esto quiere decir que un objeto de color magenta muy puro produce un estímulo cromático que no puede sintetizarse exactamente con luz roja y azul; con dichos iluminantes solamente se podrá producir un estímulo cromático de la misma longitud de onda dominante, pero con saturación menor. Sin embargo, si el objeto magenta es iluminado con luz levemente saturada con verde, entonces la síntesis puede efectuarse. Esto es equivalente a *restar* luz verde a la mezcla. Del mismo modo un amarillo saturado podrá sintetizarse sumando rojo y verde, y restando una pequeña proporción de azul. Evidentemente la resta de luces no puede realizarse con los dispositivos actuales (televisores o monitores), por lo que algunos colores muy saturados de la gama de los complementarios a los iluminantes primarios no pueden reproducirse con total fidelidad con dichos dispositivos. Experimentalmente, sin embargo, esta deficiencia demuestra ser de una importancia menor.

Conociendo la proporción de sumas y restas necesarias para sintetizar cualquier color, (según la teoría triestímulo) la respuesta de los receptores especializados en la retina contiene una parte *negativa* cuando la longitud de onda del estímulo corresponde aproximadamente a la longitud de onda de su color complementario. Por ejemplo, los receptores del rojo tienen un máximo a los 650nm. y un máximo negativo aproximadamente a los 470nm. (correspondiente al color denominado *cyan*). Esto puede deberse a que los receptores específicos realicen una asimilación anabólica en un caso y catabólica en el otro. Las curvas que intentan reproducir este fenómeno se muestran en la Figura 5.4 y fueron obtenidas por medio del análisis de la sensibilidad de sujetos videntes normales y con problemas de percepción visual en uno o más de los receptores (daltónicos y otros cinco casos posibles).

5.4 El Diagrama CIEXY de Cromaticidad

Es posible suponer la existencia de una fuente luminosa cuya densidad de energía corresponda a las curvas de sensibilidad de cada uno de los receptores. Por ejemplo, una fuente luminosa que radie energía con un máximo alrededor de los 650nm. y con un máximo negativo alrededor de los 470nm. coincide con la curva de sensibilidad del receptor para el rojo. Dicho color se denomina

iluminante X. Del mismo modo las curvas de los receptores verde y azul se corresponden con los iluminantes Y, y Z, respectivamente. Es importante tener en cuenta, sin embargo, que estos colores no se pueden crear físicamente, y por lo tanto son "invisibles". Podemos sin embargo pensar en ellos como "primarios super-saturados".

Los iluminantes X, Y y Z son tales que cada uno posee una función de distribución de densidad de energía relativa en función de la longitud de onda. Si suponemos el plano de los estímulos visuales con luminosidad constante X+Y+Z=1, obtenemos en el mismo todos los estímulos cromáticos posibles a una luminosidad total dada. Esto es, de los tres parámetros podemos variar dos: la crominancia y la saturación. La combinación de ambos se denomina *cromaticidad*. Es posible, entonces, construir diagramas planos con todas las cromaticidades posibles, denominados precisamente *diagramas de cromaticidad*. Fueron estandarizados en 1930 por la CIE (Comision International de l'Eclariage) y existen varios tipos, siendo el más común el XY, que representa el plano X+Y+Z=1, aunque el XZ tambien es utilizado.

Podemos observar la forma que asume el diagrama de cromaticidad CIEXY en la Figura 5.5. El interior de la zona demarcada contiene todas las cromaticidades visibles por el ojo humano. Los colores "puros" o con máxima saturación, que corresponde a una única longitud de onda, se ubican en la parte curva de la frontera. La parte recta inferior de la misma corresponde a la línea del magenta saturado (que no corresponden a una longitud de onda única). A medida que nos acercamos al centro los colores se vuelven menos saturados.

En la Figura 5.6 podemos ver otros detalles del diagrama. En la frontera del mismo está marcada la longitud de onda asociada a un estímulo monocromático. Así, por ejemplo, el punto marcado con 580nm. corresponde a una radiación amarilla saturada. También está marcada en el interior la curva de cromaticidad de la radiación de un cuerpo negro en función de su temperatura (denominada también curva de la *temperatura cromática*). Dicha función, conocida como ley de Plank, predice que un objeto radia fotones cuya distribución espectral es función de la temperatura. Esta ley es la que permite, entre otras cosas, determinar la temperatura en la superficie de las estrellas por la distribución cromática de la luz que radian.

Por ejemplo, alrededor de los 1000K el color de la radiación es un rojo intenso. Luego pasa por el anaranjado (temperatura cromática de una vela) y el amarillo (temperatura de las lámparas), cada vez con menos saturación, hasta llegar a un "blanco" aproximado a los 6500K. A mayores temperaturas el color cromático se va tornando azulado, hasta llegar al color de la radiación de un cuerpo a "temperatura infinita", la cual es una radiación de color azul verdoso poco saturado.

En el punto X=Y=Z=$\frac{1}{3}$ se encuentra el denominado *iluminante* E, con igual energía a toda longitud de onda. Cerca del punto E se encuentra el iluminante D que corresponde aproximadamente al color de la luz solar directa al mediodía, es decir, a la radiación de un cuerpo negro a 6500K –la temperatura cromática del sol– filtrada por la atmósfera. Por debajo de ambos se encuentra el iluminante C, que corresponde al color de la luz diurna (luz solar directa más la reflejada por la atmósfera).

Cada punto X, Y que yace dentro del área visible del diagrama representa una cromaticidad única y diferente de las demás. La segunda ley de Grassman expresa que *variar escalarmente un estímulo cromático no afecta su cromaticidad*. Esto es: si aumentamos o disminuimos la intensidad de un estímulo, la cromaticidad del mismo, es decir, sus coordenadas en el diagrama XY no varían. Esto equivale a decir que la intensidad luminosa L es "perpendicular" al diagrama CIEXY y que por lo tanto todo estímulo cromático puede representarse en un espacio LXY (ver Figura 5.7).

La tercera ley de Grassman nos permite establecer la cromaticidad resultante de la suma de dos fuentes luminosas de cromaticidad conocida. En efecto, sean F_1 y F_2 dos fuentes luminosas con sus respectivas intensidades y cromaticidades. El estímulo resultante $F_1 + F_2$ se puede encontrar

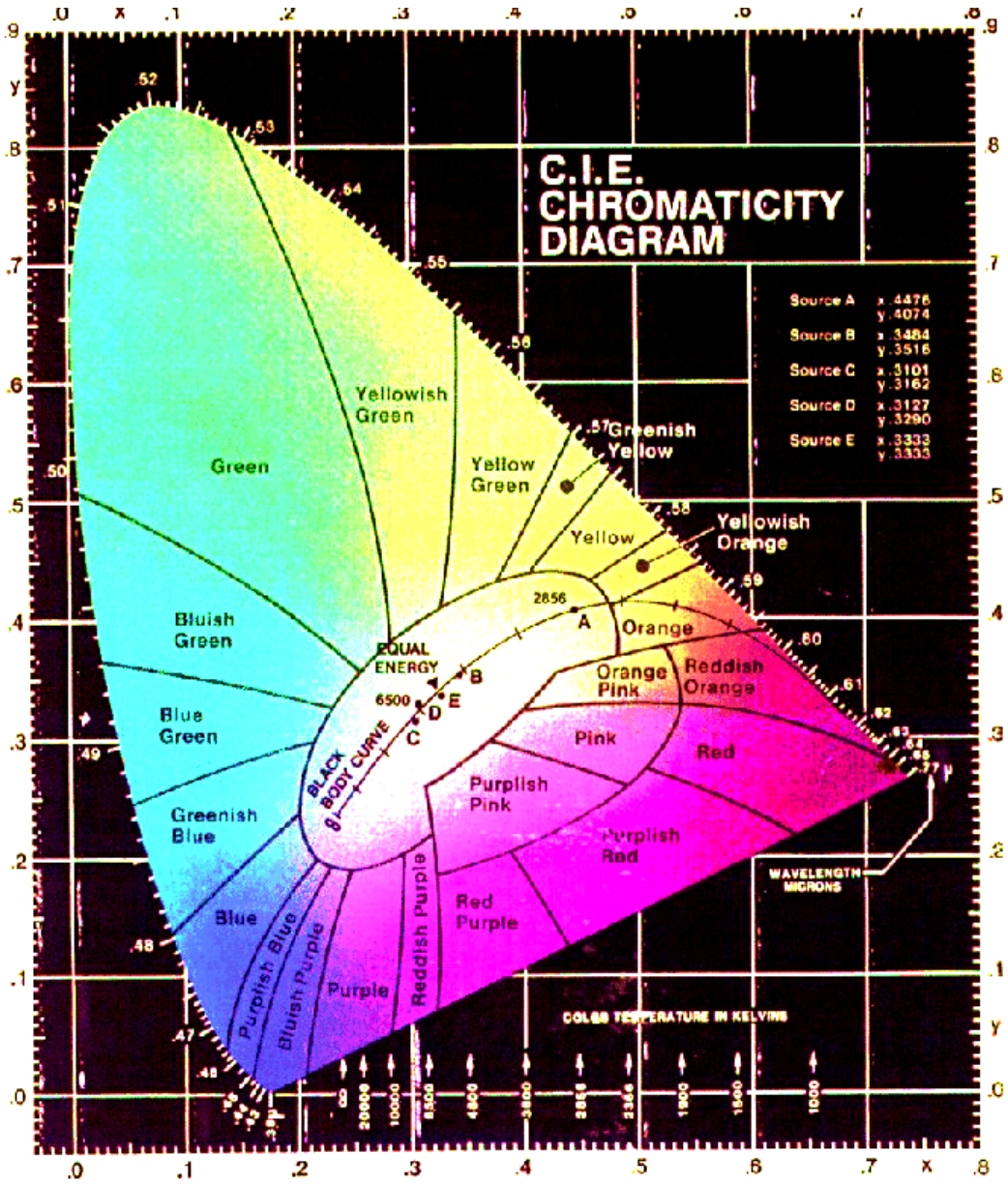


Figura 5.5 *Diagrama CIEXY de cromaticidad.*

por medio de su suma vectorial (ver Figura 5.8). Dicha suma vectorial proyectada sobre el plano X+Y+Z=1 representa la cromaticidad de la suma de ambas fuentes. Esto quiere decir que la cromaticidad resultante yacerá sobre la recta que una a las cromaticidades de los distintos componentes de la suma, estando más cerca de uno o de otro según sean las intensidades respectivas (ver Figura 5.9).

Esto equivale a *interpolar* cromaticidades. La *extrapolación* de colores es tambien factible. Supongamos que hemos estimado la cromaticidad de un estímulo en el punto X y que lo suponemos

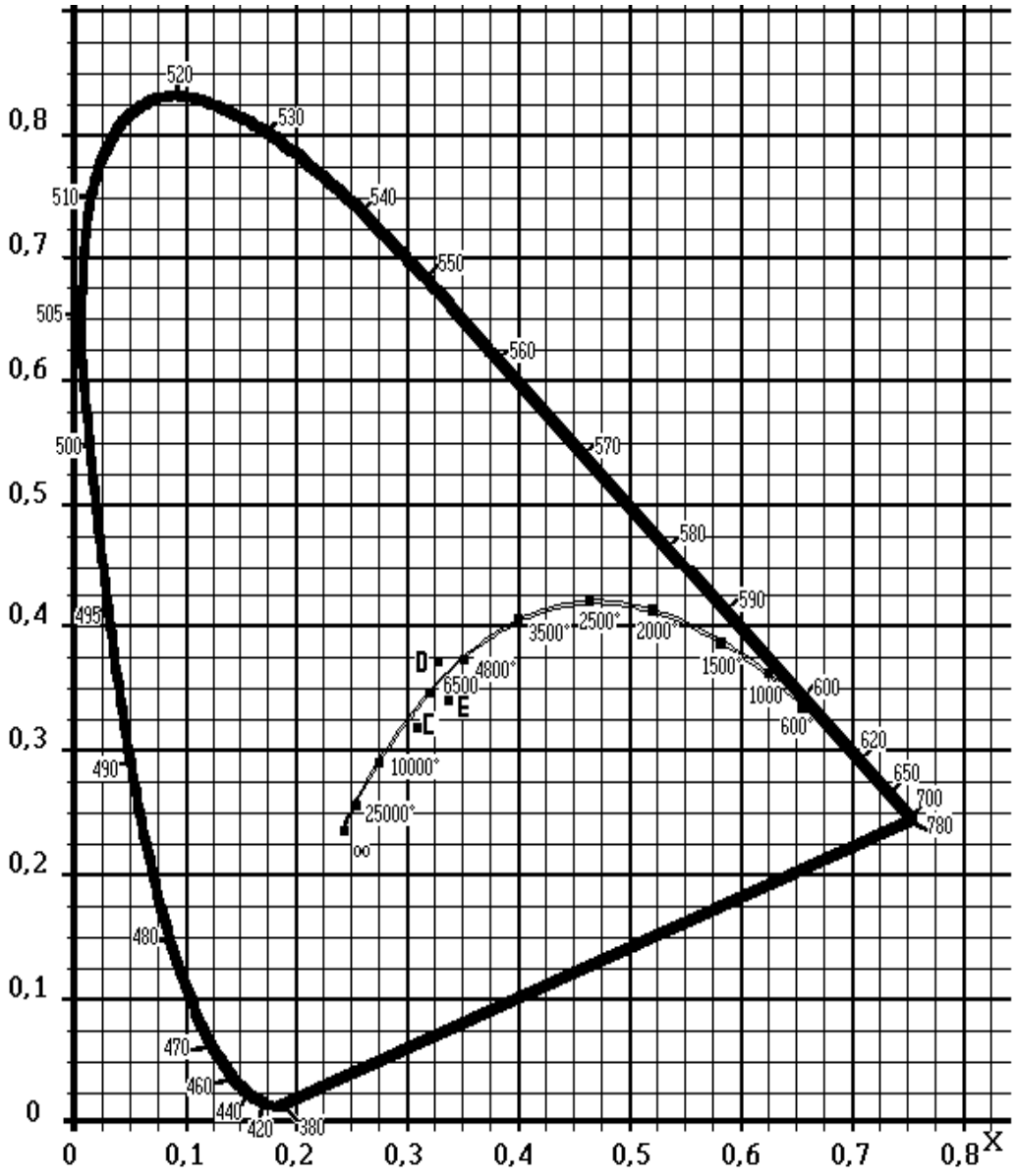


Figura 5.6 *Escalas útiles en el diagrama CIEXY de cromaticidad.*

iluminado por el iluminante C (al cual consideramos entonces como luz blanca). Entonces podemos estimar la longitud de onda dominante del color X trazando una recta desde C, pasando por X y prolongándola hasta intersectar la frontera del diagrama (ver Figura 5.10). La razón de la distancia de X a C con respecto a la longitud total del segmento es también una buena estimación de la saturación del color X. Es importante destacar que esta determinación depende de la condición de iluminación en que ocurre el estímulo, es decir, si el iluminante tiene una temperatura cromática diferente, entonces la percepción de colores complementarios se modifica.

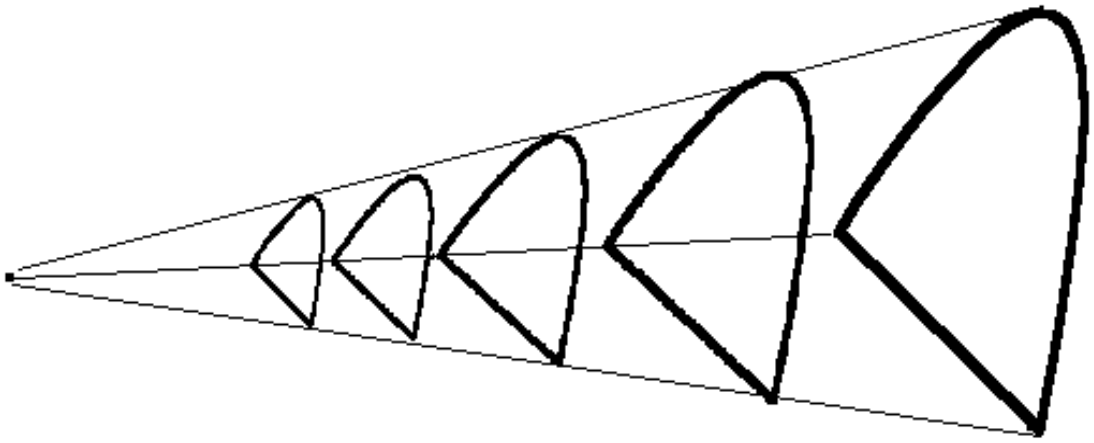


Figura 5.7 *Espacio cromático LXY.*

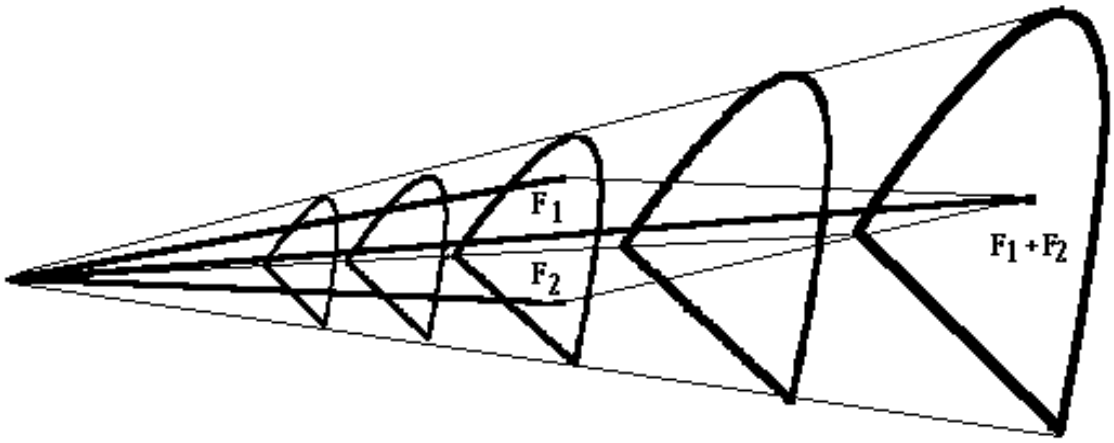


Figura 5.8 *Resultante de la suma de dos fuentes luminosas.*

La representación de gamas cromáticas es tambien sencilla en el diagrama CIEXY. La *gama cromática* es el conjunto de todas las cromaticidades representables por un método de reproducción. Por ejemplo, los monitores o televisores tienen tres iluminantes primarios que corresponden a los colores del fósforo de los tres haces electrónicos. La gama cromática de tales dispositivos es, por tanto, el interior del triángulo determinado por los mismos. Un monitor típico tiene fósforos que producen luz saturada a 650nm., 530nm., y 425nm. La gama cromática resultante se muestra en la Figura 5.11. Este procedimiento se puede extender a casos con más colores primarios, aunque al trabajar con tintas la gama cromática resultante puede no ser convexa, debido a que no se cumple la linealidad de la combinación de colores.

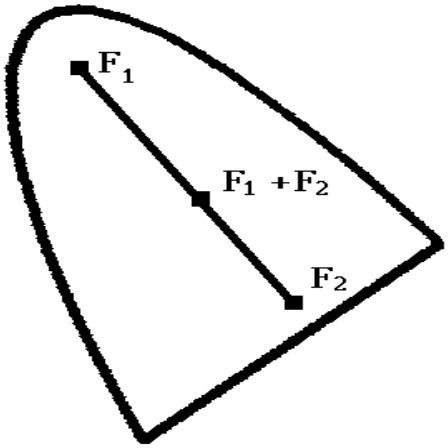


Figura 5.9 *Proyección de la resultante de la suma de dos fuentes luminosas en el plano X+Y+Z=1.*

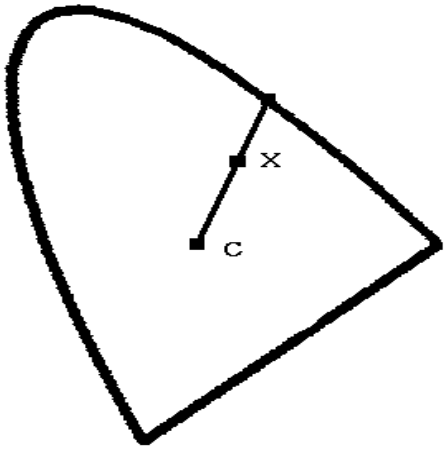


Figura 5.10 *Estimación de la longitud de onda dominante (crominancia) y de la saturación de un estímulo X dado.*

5.5 Espacios Cromáticos

Los colores primarios de los monitores son entonces el rojo (C=650nm.; Y=0.3), el verde (C=530nm.; Y=0.59), y el azul (C=425nm.; Y=0.11). La gama cromática que se puede obtener con dichos primarios incluye todos los colores visibles excepto algunos amarillos, cyan y magentas muy saturados. En las tarjetas gráficas, cada pixel tiene asignada memoria para representar los tres primarios con una cantidad fija de niveles posibles, por ejemplo 64. De esta forma podemos obtener directamente la representación de cualquier color como suma de rojo, verde y azul, en el denominado *espacio cromático* RGB (ver Figura 5.12).

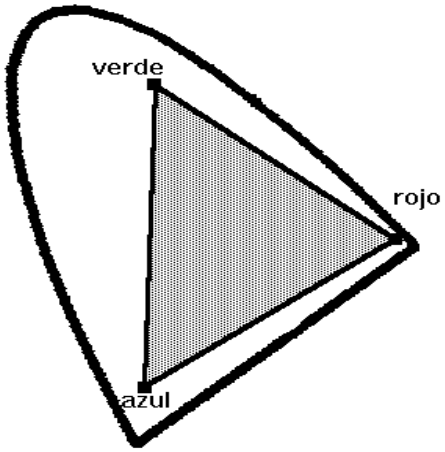


Figura 5.11 *Gama cromática de un monitor típico.*

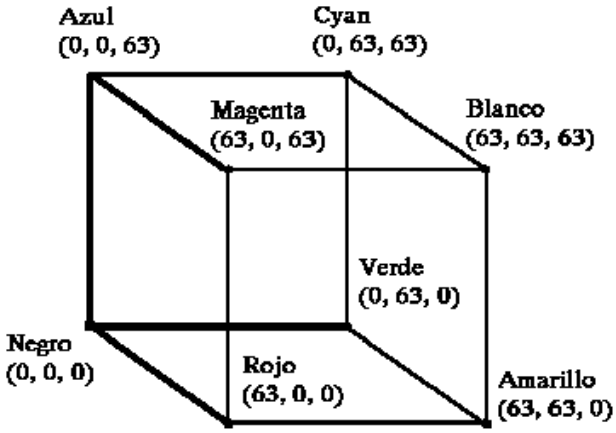


Figura 5.12 *Espacio cromático RGB.*

Supongamos que un dispositivo tiene 64 niveles para cada primario (de 0 a 63). La gama representable incluye a $64^3 = 256$K colores posibles. Los colores neutros (grises) se ubican en la diagonal entre el blanco y el negro, correspondiendo a valores de la forma (i, i, i). Los colores saturados se ubican en la frontera mostrada en la Figura 5.13. Los restantes colores son no saturados, y asumen alguna de las seis permutaciones de la forma $(i, i + a, i + a + b)$, donde $0 \leq i \leq 63$, $0 \leq a \leq 63 - i$ y $0 \leq b \leq 63 - i - a$. Por lo tanto, en el espacio RGB, los colores pueden descomponerse en un gris (i, i, i) sumado a un color saturado $(0, a, a + b)$.

El espacio cromático RGB es adecuado para su representación en el hardware, pero el usuario, dado un color (i, j, k), no puede conocer intuitivamente el color resultante, su crominancia, luminancia y saturación. Tampoco es directo cuál es el color complementario, cuál es el color saturado de la crominancia, etc.

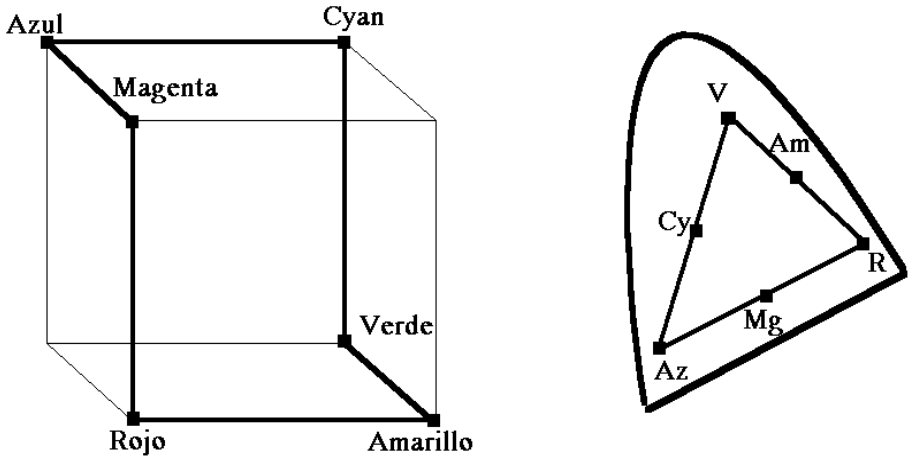


Figura 5.13 *Representación de los colores saturados del espacio RGB en el espacio LXY.*

Una representación similar a la RGB se utiliza en las impresoras color, donde los primarios son los complementarios a RGB, es decir CMY. El pasaje de RGB a CMY es sencillo: R es K-C, G es K-M y B es K-Y y viceversa, donde K es el negro. Esta conversión no se utiliza en la práctica por dos motivos. Primero, la gama de grises no es estable con respecto a los pigmentos. Esto quiere decir que a distintas intensidades dominan distintos pigmentos. Segundo, la mezcla de pigmentos no tiene las propiedades de la mezcla de luces. Esto quiere decir que la gama cromática puede no ser convexa, el "blanco" en el monitor puede no coincidir con el color del papel. Por ello las impresoras utilizan pigmento negro para estabilizar la combinación de pigmentos, denominándose espacio CYMK al modelo de color resultante.

Dado que la estructuración psicológica de los estímulos cromáticos parece realizarse en función de la crominancia, luminancia y saturación, resulta natural utilizar a dichas dimensiones para representarlos. Esto da origen al sistema CSV. C (de crominancia) es un ángulo que generalmente está medido en una escala de 0 a 360 grados, con origen en el rojo y medido en sentido antihorario. Por ejemplo un verde tiene crominancia 120 grados. El color complementario de C tiene crominancia C + 180 grados. S (de saturación) es una magnitud de 0 a 1, donde 0 representa un color neutro (un gris) y 1 representa un color saturado (ver Figura 5.14). V (de "valor" o luminancia equivalente) es una magnitud de 0 a 1, donde 0 es negro en cualquier condición de los otros parámetros, y 1 es la máxima intensidad.

Por ejemplo, el verde que en RGB se representa $(0, n-1, 0)$, en se representa $(120, 1, 1)$. De esta forma, todos los colores representables se ubican en un cono, cuyo vértice inferior es el negro y el círculo superior contiene todas las cromaticidades. El eje del cono es la escala de grises.

Para pasar del modelo CSV al RGB es necesario aplicar un algoritmo sencillo que aprovecha las simetrías en dicho espacio. Si el color en CSV tiene saturación cero, entonces devuelve el valor equivalente de L en cada uno de los primarios RGB (es decir, un gris). Si el color tiene saturación mayor que cero, entonces, como vimos, es posible descomponerlo en un primario principal, uno secundario y uno opuesto. La determinación de cuál de los primarios RGB cumple cada uno de esos roles se determina según el "sexto" del círculo de crominancias en que está el color. Así, una crominancia de 70 grados, por ejemplo, tiene al verde como principal, al rojo como secundario y al azul como opuesto, etc. En total son seis casos.

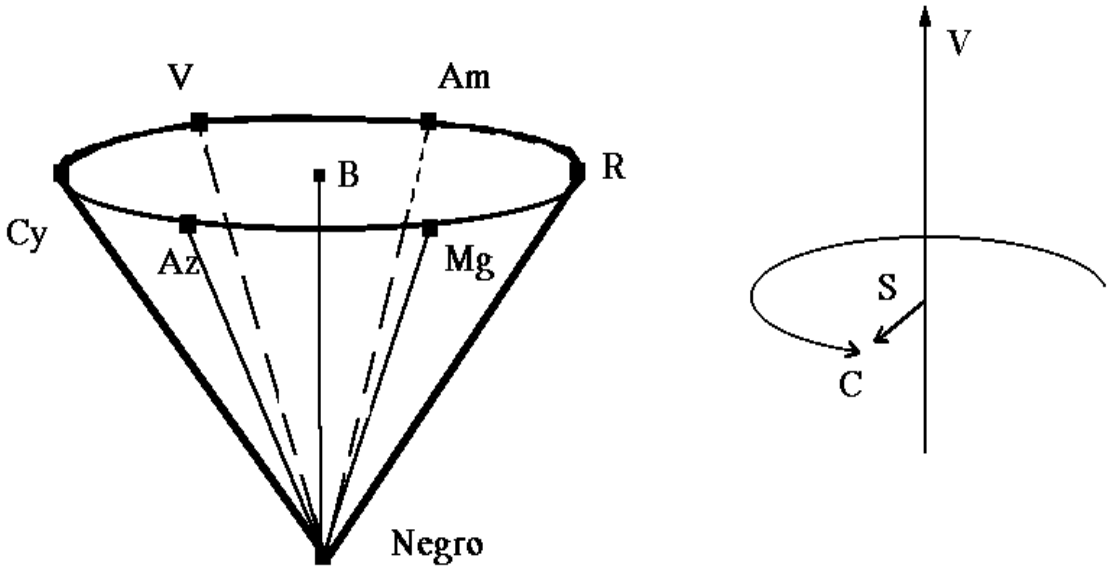


Figura 5.14 *Espacio cromático CSV.*

La intensidad del primario principal es el valor equivalente de V, mientras que la intensidad del color opuesto es dicho valor multiplicado por (1-S). Es decir que si la saturación es máxima, la intensidad del primario opuesto es cero. La intensidad del secundario depende también de lo cerca o lejos que el color esté del "sexto" del círculo correspondiente. Así, la intensidad del rojo para una crominancia de 70 grados es mayor que para una crominancia de 110 grados.

El procedimiento mostrado en la Figura 5.15 efectúa dichos cálculos, recibiendo como entrada los valores de `c` como un ángulo de 0 a 360 grados, y de `v` y `s` como escalares entre 0 y 1. El pasaje de los valores `r, g, b` de salida se hace por referencia para facilitar el uso del procedimiento, aunque lo correcto hubiera sido definir tipos de datos para color en CSV y en RGB y escribir el algoritmo como función.

Es posible encontrar el procedimiento que transforma un color de RGB a CSV por medio de la misma relación. El valor es la transformación a [0,1] de la intensidad del primario en RGB más intenso (el principal). La saturación es el cociente entre el menos intenso (el opuesto) y el más intenso, y la crominancia está determinada por el principal y el secundario. Por ejemplo, si el principal es azul y el secundario es rojo, entonces el ángulo base de la crominancia es 240 grados. A medida que el rojo es más importante, se suma a dicha base un ángulo de 0 a 60 grados.

5.6 Representación de Color en Computación Gráfica

El color en las tarjetas gráficas, como vimos, se representa por medio del espacio cromático RGB (ver la Sección 2.1). Esto significa que el color de cada pixel se representa por medio de una terna de valores de las componentes en rojo, verde y azul, respectivamente, que tiene dicho color. Si cada pixel tiene asignada memoria para sus componentes RGB, se trata del modo *true color*. En

```
procedure csvrgb(c,s,v:real; var r,g,b:integer);
   var  x,y,z,i,vv:integer;
                f:real;
   begin
    l:=255*l;                              % luminancia de [0,1] a [0..255]
    if s=0 then begin                      % color no saturado
      r:= g:= b:= trunc(v);
      end
    else begin
      i:=trunc(c/60);                      % determinar el sexto del circulo
      f:=c/60-i;                           % cerca del comienzo o del final
      x:=trunc(v*(1-(s*f)));             % comienzo
      y:=trunc(v*(1-(s*(1-f))));           % final
      z:=trunc(v*(1-s));                   % opuesto
      vv:=trunc(v);                        % principal
      case i of                            % cual simetria
        0: begin r:=vv; g:=y; b:=z; end;
        1: begin r:=x; g:=vv; b:=z; end;
        2: begin r:=z; g:=vv; b:=y; end;
        3: begin r:=z; g:=x; b:=vv; end;
        4: begin r:=y; g:=z; b:=vv; end;
        5: begin r:=vv; g:=z; b:=x; end;
      end;
     end;
    end;
```

Figura 5.15 *Transformación del espacio CSV a RGB.*

cambio, si el pixel guarda un índice a una entrada en una tabla de colores donde está definido el color del cual está pintado el pixel, estamos en modos gráficos más económicos.

En dichos modos, el seteado del índice del color de un pixel se efectúa por medio de la sentencia `putpixel(x,y,c)`, mientras que asignar una entrada en la tabla de colores se realiza por medio de la sentencia `setrgbpalette(c,r,g,b)`. En dicho modelo se utiliza la sentencia `putpixel(x,y,c)` para acceder al buffer de pantalla y setear el pixel en la posición `x, y` con el índice de color `c`, con `x, y, c` de tipo `word`. En los modos gráficos VGA y super VGA, los parámetros `r, g, b` son de tipo `word`, pero se truncan los dos bit menos significativos, dado que el rango efectivo de cada componente es de 0 a 63. Por lo tanto es conveniente utilizar una aritmética dentro de dicho rango para representar los colores, y multiplicar por 4 en el momento de la llamada.

Los modos gráficos VGA y SVGA permiten definir en general 256 colores simultáneos (la "paleta" gráfica) de entre 256K colores posibles. Estas posibilidades pueden ser, en algunos casos, poco satisfactorias, no solo porque la paleta sea limitada, sino porque los colores son definibles con poca precisión en algunos casos. Si bien el ojo humano detecta aproximadamente 350.000 colores diferentes (y es capaz de distinguir aproximadamente 50.000 en forma simultánea), esta sensibilidad no es uniforme en todo el espacio cromático, sino que es mucho mayor en ciertas áreas (por ejemplo en el eje naranja-violeta) y mucho menor en otras (por ejemplo en el eje magenta-verde).

Esto quiere decir que el ojo detecta a simple vista una gran diferencia entre los colores (63,63,0) y (63,63,1) o (63,62,0), por ejemplo, pero no detecta diferencias entre (0,63,0) y (0,62,0) o (0,62,1) y probablemente tampoco con (5,63,5)! De los 256K colores definibles, miles de ellos son idénticamente percibidos, mientras que otros no se representan con una fidelidad adecuada. En otras palabras, el espacio RGB es una forma muy ineficiente de representar colores porque la información está codificada de una manera muy "despareja" con respecto a la capacidad del ojo.

En los modos gráficos *true color* el problema se soluciona con un costo muy grande (3 bytes por pixel es mucho más de lo necesario). Sin embargo, hay personas con visión cromática muy sensible que siguen encontrando diferencias de matiz entre colores contiguos en la gama del amarillo-anaranjado y del violáceo. Probablemente la mejor solución hubiera sido contar con tecnología CSV en las tarjetas gráficas, dado que la conversión al RGB del monitor se puede hacer dentro de la controladora de video.

5.7 Paletas Estáticas y Dinámicas

En muchas circunstancias la capacidad de manejo de colores en las tarjetas gráficas está restringida a una paleta de 256 colores simultáneos. Esto puede deberse a varios factores. Por ejemplo, puede ser necesaria la mayor resolución posible, y sin una cantidad de memoria adecuada para el *frame buffer* puede no ser suficiente para soportar el modelo *true color*. Puede ocurrir también por limitaciones tecnológicas (tarjetas o monitores obsoletos, falta de *drivers*, etc.). La limitación en la cantidad de colores simultáneos se sobrelleva, en general, con un esquema cromático que utiliza una paleta con los 256 colores más significativos de la imagen. Estos 256 colores son obtenidos durante la generación de la misma (generalmente con histogramas y técnicas de separación), por lo que las paletas se denominan *dinámicas*.

Sin embargo, en un sistema gráfico de propósito general el esquema dinámico de paletas puede ser inadecuado cuando se desea manejar dos o más imágenes desarrolladas independientemente, porque cada una de ellas reclamará lo que considera que son los colores más adecuados. El sistema tiene que llegar a una solución de compromiso, sacrificando algunos de los colores de cada una de las imágenes. Esto produce como resultado un deterioro impredecible en la calidad gráfica. Al mismo tiempo, una paleta dinámica resulta inaceptable en aplicaciones interactivas, porque al modificar cualquier propiedad de la escena (el agregado de un nuevo objeto, los atributos de un objeto ya dibujado, las condiciones de iluminación, etc.) se requiere el recálculo de los histogramas de la escena completa, y el redibujado completo de la misma, lo cual insume un tiempo muy grande.

En las paletas *estáticas*, por su parte existe un esquema cromático predefinido. Es decir, se define de antemano un conjunto de colores adecuados para graficar cualquier escena, en cualquier condición de iluminación y bajo cualquier algoritmo. Los resultados gráficos tienden a ser de menor calidad. Sin embargo, la calidad de las imágenes, una vez graficadas, no se deteriora con el ulterior agregado de otras imágenes. Al mismo tiempo, la referencia al índice de color que corresponde a cada pixel puede calcularse a partir del color que debería corresponder al mismo. De esa forma, la imagen es graficada al mismo tiempo que es computada, sin un costo adicional debido al manejo de color.

Un esquema estático de color ubica el índice de color con el que corresponde colorear un pixel en función del color reclamado por el modelo de iluminación, y los colores más cercanos disponibles en la paleta, probablemente por medio de interpolación. En el Capítulo 7 estudiaremos cómo funcionan los modelos de iluminación. Supongamos ahora que para un punto p de cada cara en la escena se computan ecuaciones que determinan la componente del color para cada primario RGB. Cada pixel de cada cara reclamará un color determinado, es decir, una terna de reales (R(p), G(p), B(p)). Esta terna debe transformarse a una terna (R,G,B) dentro de la aritmética de la tarjeta

```
procedure inicializar_paleta();
var rojo : array[0..5] of integer = (0, 20, 32, 45, 55, 63);
   verde : array[0..6] of integer = (0, 15, 22, 30, 40, 50, 63);
    azul : array[0..5] of integer = (0, 25, 35, 45, 55, 63);
   r,g,b : integer;
begin
 for r := 0 to 5 do
  for g := 0 to 6 do
   for b := 0 to 5 do
     setrgbpalette(42*r+6*g+b, 4*rojo[r], 4*verde[g], 4*azul[b]);
end;
```

Figura 5.16 *Inicialización de la paleta estática.*

gráfica. Al mismo tiempo, los valores de (R,G,B) deben estar asociados a uno de los índices de color disponibles.

Una forma de diseñar paletas estáticas proviene de las siguentes consideraciones. Normalmente las tarjetas representan la intensidad de cada primario con una precisión de 6 bit en una escala entera de 0 a 63, lo cual permite 256K combinaciones de colores. De dichos 256K colores, es necesario elegir los 256 más representativos para un uso general. Más aún, deben ser elegidos de manera tal que cualquier aplicación gráfica que reclama un color no existente en la paleta, pueda encontrar rápidamente el índice de un color perteneciente a la paleta que sea el más adecuado para reemplazarlo. Es necesario, entonces, elegir un subconjunto de las 64 intensidades permitidas para cada primario, de modo tal que el producto de la cantidad de elementos en cada conjunto sea menor o igual que 256. En los modos graficos de 64K colores (conocidos como *hi-color*), se utiliza una partición en $32 \times 64 \times 32$, asignándole mayor resolución cromática al primario verde, dada la gran sensibilidad del ojo al mismo. Pero en nuestro caso, combinaciones como $8 \times 8 \times 4$ quedan eliminadas, porque 4 intensidades posibles para un primario es un valor demasiado pequeño, aún para el primario azul. La combinación que empíricamente resultó ideal fue $6 \times 7 \times 6$, ya que su producto es 252, es decir, desaprovecha solamente 4 índices de color, y representa una buena solución de compromiso.

De esa forma, se eligen 6 intensidades permitidas para los primarios rojo y azul, y 7 para el primario verde (que es para el cual el ojo humano es más sensitivo). De esa manera, el espacio RGB de la tarjeta queda "cuantizado" en 150 prismas rectangulares, y todos los colores representables que caen dentro de un mismo prisma se coercionan al valor del vértice más cercano. La determinación de los 6x7x6 valores se debe realizar en forma experimental, teniendo en cuenta la corrección-γ del monitor utilizado. En un determinado monitor (un NEC MultiSync 3D) para una posición adecuada en las perillas de brillo y contraste, los resultados elegidos fueron rojo = (0, 20, 32, 45, 55, 63), verde = (0, 15, 22, 30, 40, 50, 63) y azul = (0, 25, 35, 45, 55, 63). Debemos recordar que en este modo gráfico, es posible dar un valor entero de 0 a 63 a la intensidad en cada primario (ver Capítulo 2).

La estructura de la cuantización del espacio RGB elegida resulta ser muy práctica en el momento de encontrar el color con el cual pintar un pixel. Antes de ejecutar la graficación, es decir, como paso de inicialización de la interfase, se almacena la paleta en la tabla de colores de la pantalla. Esto se efectúa con el procedimiento mostrado en la Figura 5.16.

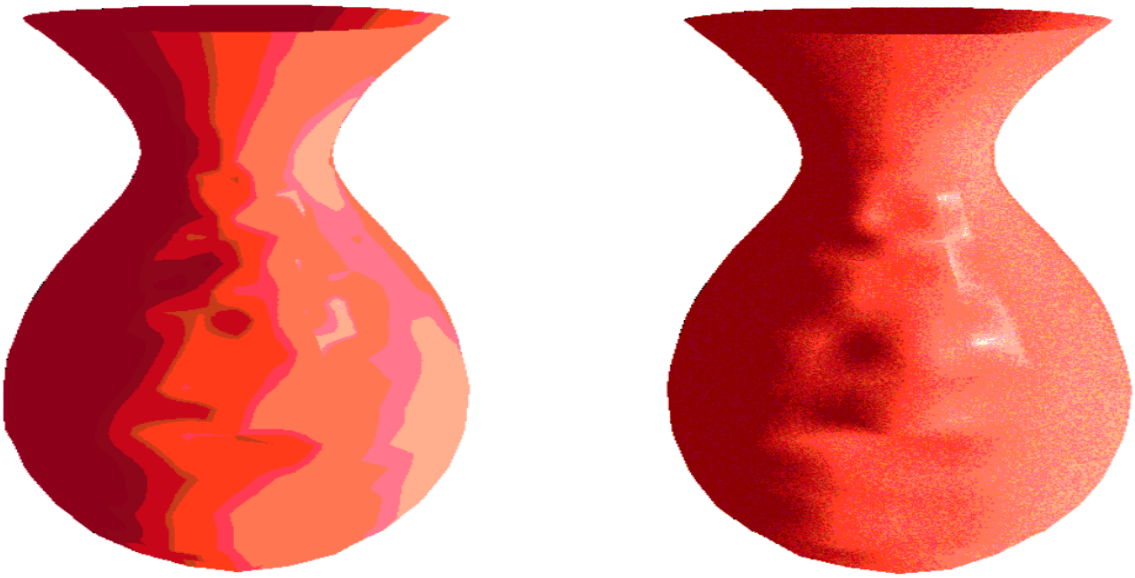


Figura 5.17 *Paleta estática sin y con* dithering *aleatorio.*

Cuando es necesario graficar un pixel de un color R, G, B arbitrario, se buscan los valores r, g, b tales que `rojo[r], verde[g], azul[b]` sean los valores más cercanos, y luego se grafica el pixel con el índice de color `42*r+6*g+b`. Por ejemplo, si el modelo de iluminación reclama un color (35,42,48), se ubican `r=2, g=4, b=3` y se grafica el pixel con el índice de color 111. Con este esquema se produce el efecto de "bandas de Mach" cuando se pintan áreas contiguas con colores parecidos. Este efecto es producido por la capacidad del ojo de amplificar localmente pequeñas variaciones cromáticas, lo cual le permite, entre otras cosas, reconocer los bordes y las formas de los objetos. Sin embargo, el efecto que producen las bandas de Mach al utilizar esta paleta de colores es indeseado (ver Figura 5.17). Pero a diferencia de lo que sucede con las paletas dinámicas, en nuestro esquema es posible utilizar una técnica de *dithering* aleatorio, es decir, se puede "perturbar" aleatoriamente un color dado cambiándolo por alguno de sus vecinos más próximos en la paleta. Esto es así porque en nuestro modelo, dado un determinado color de la paleta, es posible ubicar rápidamente a sus colores vecinos, los cuales, además, son muy similares.

Una forma muy económica de producir este efecto consiste en perturbar para cada primario la cuantización elegida. Si para el primario R se reclama un valor R, el cual está comprendido entre `rojo[r]` y `rojo[r+1]` se perturbará la elección del valor `r` o `r+1` en la generación del índice de color asociado al pixel (lo propio se efectúa con los otros dos primarios). Para ello se genera un número aleatorio *rnd* uniformemente distribuido entre 0 y 1. Si $rnd \geq \frac{R-\texttt{rojo[r]}}{\texttt{rojo[r+1]}-\texttt{rojo[r]}}$ se utiliza `r+1`, y en caso contrario se utiliza `r`. En el ejemplo mencionado más arriba, la elección para la cuantización del rojo está circunscripta a los valores predefinidos 32 o 45. Como 35 es más cercano a 32 que a 45, la probabilidad de que se utilice dicho valor es mayor a la de utilizar 45. De esa manera, se transforma el *aliasing* cromático producido por la baja frecuencia de muestreo en un ruido uniforme (ver Figura 5.17).

5.8 Ejercicios

1. Implementar tipos de datos para representar colores en los espacios RGB y CSV en precisión real, y en el espacio RGB con la precisión de la tarjeta gráfica. Implementar los algoritmos de conversion entre espacios cromáticos.

2. Determinar experimentalmente en qué cromaticidades el ojo es menos sensible o más sensible a cambios cromáticos. Determinar aproximadamente la distancia entre diferencias apenas perceptibles, es decir, cuánto es necesario alterar la definición de un color hasta que se distinga que hay una diferencia.

3. Implementar la paleta estática descripta en este capítulo, y graficar un rectángulo pintando todas las cromaticidades para una luminosidad dada (por ejemplo, recorrer los colores del arco iris de izquierda a derecha, y las saturaciones de arriba abajo). Pauta: asignando C a la coordenada x y S a la coordenada y del rectangulo (con las debidas escalas), se genera una secuencia bivariada CSV.

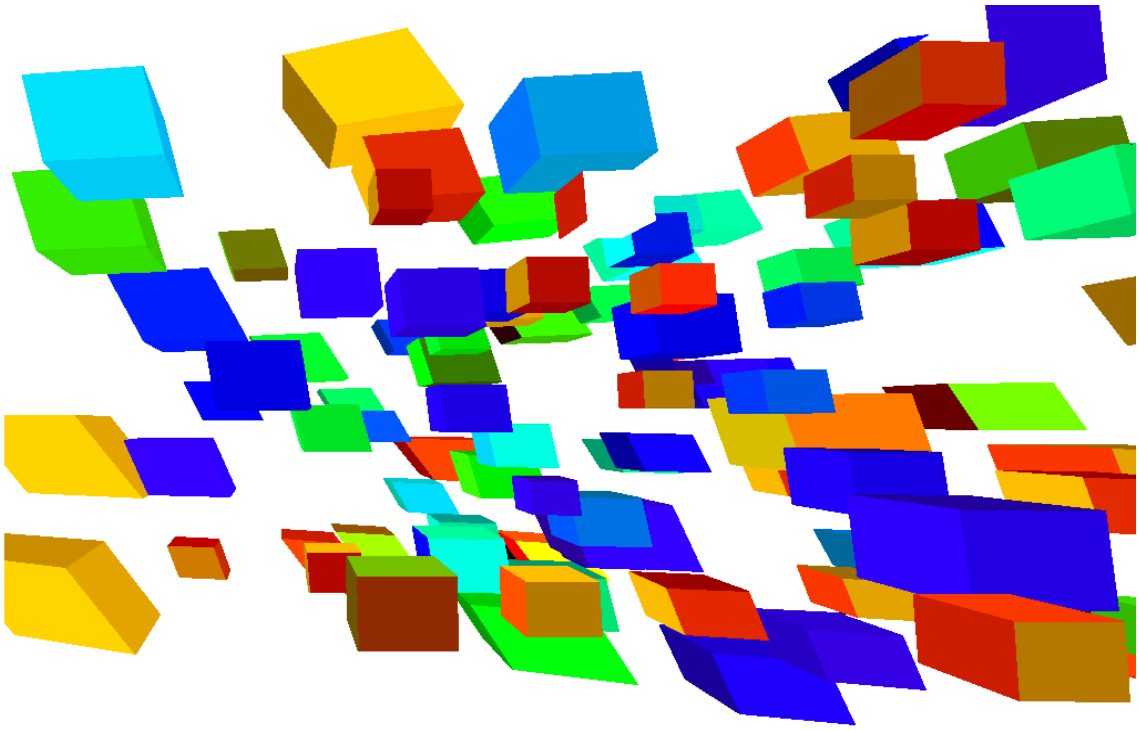
5.9 Bibliografía recomendada

Los aspectos físicos y fisiológicos del color así como los elementos de la teoría de color, las determinaciones experimentales y los diagramas cromáticos pueden consultarse en [17] o en [65]. Una referencia adecuada para colorimetría, espacios cromáticos CIE, sensibilidad del ojo, etc. es [81]. La descripción del manejo de colores en el hardware, la representación de los espacios cromáticos y los algoritmos para convertir colores de una representación a otra figuran en el Capítulo 13 del libro de Foley et. al. [33]. En [31] se describe la relación entre el espacio CLS y los nombres asociados a los colores. Pueden conocerse y compararse los diversos espacios cromáticos y sus ventajas en el trabajo de Schwarz et. al [75].

Es interesante consultar algunas técnicas para el diseño de secuencias de colores, porque muchos de los problemas discutidos en este Capítulo deben resolverse también en esos casos. Recomendamos consultar los trabajos de P. Robertson [72, 73] y M. Stone [76, 77]. En [47] se discute el problema del muestreo en los espacios cromáticos. También se pueden conocer los detalles para la calibración de monitores en el trabajo de W. Cowan [22].

La paleta estática aquí descripta fue presentada en [25]. Hay otros acercamientos posibles, por ejemplo el presentado por P. Heckbert [49] o el de M. Gervautz [39]. La construcción de paletas dinámicas es una técnica compleja, donde cada autor presenta ideas de un modo diferente. Recomendamos consultar el trabajo de A. Voloboj [82]. Otros principios relacionados con la elección de colores pueden consultarse en [64].

6 Computación Gráfica en Tres Dimensiones



6.1 Objetivos de la Computación Gráfica Tridimensional

Como mencionáramos en la Introducción, la Computación Gráfica tridimensional representó el primer cambio de paradigma en la disciplina. Se pasó del objetivo de la obtención de "gráficos" al objetivo de representar un "mundo virtual" como si fuese visto desde una ventana determinada por el monitor de la computadora. Por lo tanto, la idea esencial en este Capítulo es comenzar el estudio de las técnicas que permiten representar gráficamente las imágenes con un grado de realismo que permita la percepción o visualización de objetos tridimensionales.

La búsqueda del realismo tridimensional produjo durante la década del '70 el desarrollo de un conjunto de técnicas que permiten, gradualmente, una representación más adecuada de entidades gráficas en un mundo tridimensional virtual en el que ocurren efectos de iluminación, atmósfera, interacciones, etc. (consultar en [2] un panorama de dichas técnicas y las referencias bibliográficas a los trabajos originales). Podemos citar, por ejemplo:

Proyección perspectiva: Es una técnica geométrica que representa la tercera dimensión (profundidad) como un factor de escala que afecta a las otras dos dimensiones, en función de la posición de un observador o "cámara". Se basa en un modelo simplificado del comportamiento geométrico de la luz en el ojo. Fue desarrollada en el Renacimiento por pintores como Durero, que buscaban salir del modelo artístico medieval.

Uso del color: También es una técnica desarrollada por pintores. Es otra de las maneras de representar profundidad, alterando el color de los objetos alejados en función de la misma (por ejemplo, haciendo que se vean más grises y oscuros).

Eliminación de las líneas y caras ocultas: Es una técnica indispensable, dado que las partes "no visibles" de las entidades gráficas molestan y confunden si son graficadas. Sin embargo, dada la enorme dificultad para encontrar algoritmos eficientes de cara oculta, los sistemas gráficos tridimensionales se valen usualmente de técnicas *ad hoc* y simplificaciones.

Modelos de iluminación y sombreado: Aquí la palabra "sombreado" se refiere al inglés *shading* que representa la alteración de los tonos cromáticos para dar idea de mayor o menor iluminación, y no a *shadowing* que representa la proyección de sombras de un objeto sobre otro. Los modelos de iluminación plantean la existencia de fuentes de luz ubicadas en el espacio, y de un modelo de reflexión por parte de los objetos. Esto determina que el color de los mismos varía en función de su posición y de la ubicación de la cámara.

Visión estereoscópica: Se producen dos perspectivas, levemente desplazadas, de modo que los objetos sean percibidos por cada ojo como si estuviesen "detrás" de la pantalla. En la mayoría de las personas ésto produce que el sistema visual reconstruya la percepción de una situación tridimensional (como también sucede con los estereogramas de punto aleatorio).

Modelos cinéticos y animación: Es la culminación del modelo paradigmático de la Computación Gráfica 3D con realismo. Se programan características cinéticas para los objetos de modo que se vean con la ilusión de un movimiento. Esto también ayuda a la interpretación y comprensión adecuada de la escena.

6.2 Transformaciones y Coordenadas Homogeneas

Generalizando lo visto en 2D, un punto en el espacio 3D homogéneo será un vector columna

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Las transformaciones en 3D son muy similares a las transformaciones en 2D. Esencialmente hay que agregar una fila y una columna a las matrices para representar la nueva coordenada de nuestras entidades. De esa manera tenemos que una matriz de escalamiento es

$$p' = E \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} e_x & 0 & 0 & 0 \\ 0 & e_y & 0 & 0 \\ 0 & 0 & e_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Ahora debemos tener en cuenta que nuestro espacio puede rotar alrededor de tres ejes (la rotación en 2D es implícitamente rotar el espacio bidimensional del plano xy alrededor del eje z). La rotación un ángulo θ alrededor del eje x se representa como

$$p' = R_x \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sen\theta & 0 \\ 0 & sen\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

La rotación un ángulo θ alrededor del eje y es:

$$p' = R_y \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & -sen\theta & 0 \\ 0 & 1 & 0 & 0 \\ sen\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Por último, la rotación un ángulo θ alrededor del eje z es:

$$p' = R_z \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta & 0 & 0 \\ sen\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Al igual que en 2D, es posible representar la traslación $x' = t_x + x, y' = t_y + y, z' = t_z + z$ con una matriz de transformación:

$$p' = T \cdot p, \text{ es decir, } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Toda otra transformación afín puede representarse en términos de una composición de estas transformaciones básicas. Por ejemplo la transformación T que rota por θ alrededor de un eje paralelo al eje x se representa por una matriz que se obtiene con los siguientes pasos (ver Figura 6.1):

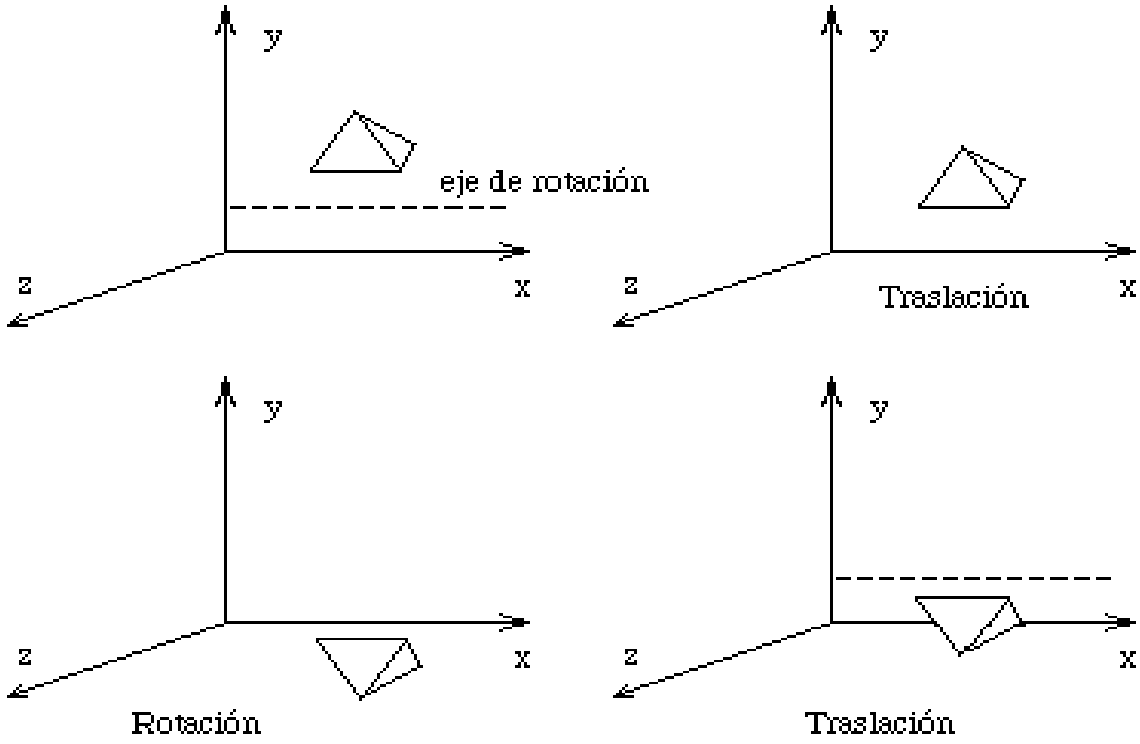


Figura 6.1 *Rotación alrededor de un eje paralelo al eje x.*

1. Se trasladan los objetos con una matriz de traslación T_t de modo que el eje de rotación sea el eje $x = 0$.
2. Se rota un ángulo θ con respecto al nuevo eje x con una matriz T_r.
3. Por último se trasladan los objetos a su posición original con $T_t^{-1} = T_{-t}$.
4. La transformación buscada es $T = T_{-t} \cdot T_r \cdot T_t$.

Si debe hacerse una rotación alrededor por un ángulo α alrededor de un eje arbitrario, entonces la secuencia es un poco más compleja:

1. Se trasladan los objetos con una matriz de traslación T_t de modo que el eje de rotación pase por el nuevo origen de coordenadas.
2. Se rotan los objetos un ángulo θ_x con respecto al nuevo eje x con una matriz T_{rx} de modo que el eje de rotación pertenezca al nuevo plano xz.
3. Se rota un ángulo θ_y con respecto al nuevo eje y con una matriz T_{ry} de modo que el eje de rotación se transforme al eje z.
4. Rotar un ángulo α alrededor del nuevo eje z (que, como vimos, coincide con el eje de rotación) con una matriz T_{rz}
5. Se rota un ángulo θ_{-y} con respecto al nuevo eje y con una matriz $T_{ry}^{-1} = T_{-ry}$.

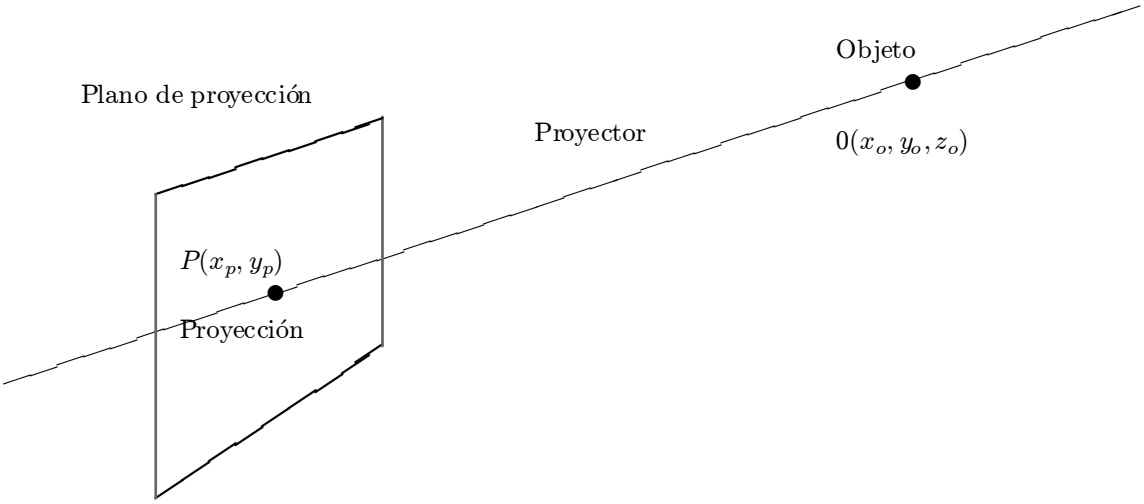


Figura 6.2 *Esquema de una proyección geométrica planar.*

6. Se rota un ángulo θ_{-x} con respecto al nuevo eje x con una matriz $T_{rx}^{-1} = T_{-rx}$.
7. Por último se trasladan los objetos a su posición original con $T_t^{-1} = T_{-t}$.
8. La transformación buscada es $T = T_{-t} \cdot T_{-rx} \cdot T_{-ry} \cdot T_{rz} \cdot T_{ry} \cdot T_{rx} \cdot T_t$.

6.3 Proyecciónes y Perspectiva

Una *proyección* abstractamente es una operación en la cual se reduce la cantidad de dimensiones de una entidad. En Computación Gráfica se utilizan normalmente las *proyecciones geométricas* para representar en 2D los objetos cuyo modelo es de 3D. Una proyección geométrica consiste en hacer pasar *ejes proyectores* por el objeto a proyectar, encontrando su intersección con una entidad de proyección, normalmente una variedad diferenciable o superficie (ver Figura 6.2).

Si la proyección se realiza sobre un plano, entonces la proyección geométrica se denomina *planar*. Si además los ejes proyectores son paralelos entre sí, entonces la proyección se denomina *paralela* (ortogonal u oblícua). En cambio, si los proyectores convergen sobre un foco, la proyección se denomina *perspectiva* (ver Figura 6.3).

Las proyecciones ortográficas son muy utilizadas en el dibujo técnico, dado que representan la tercera dimensión conservando las distancias. Esto hace que los diagramas mantengan una escala uniforme en todas sus dimensiones y que por lo tanto sean más fáciles de interpretar y utilizar (por ejemplo en planos). El sistema multivista consiste en hacer pasar ejes paralelos a los ejes del sistema de coordenadas entre las entidades a proyectar y planos paralelos a los planos del sistema de coordenadas.

Como veremos, esto equivale a *ignorar* la coordenada respectiva. Por ejemplo, hacer pasar ejes paralelos al eje z por un plano paralelo al plano xy, conserva las componentes en x e y de las

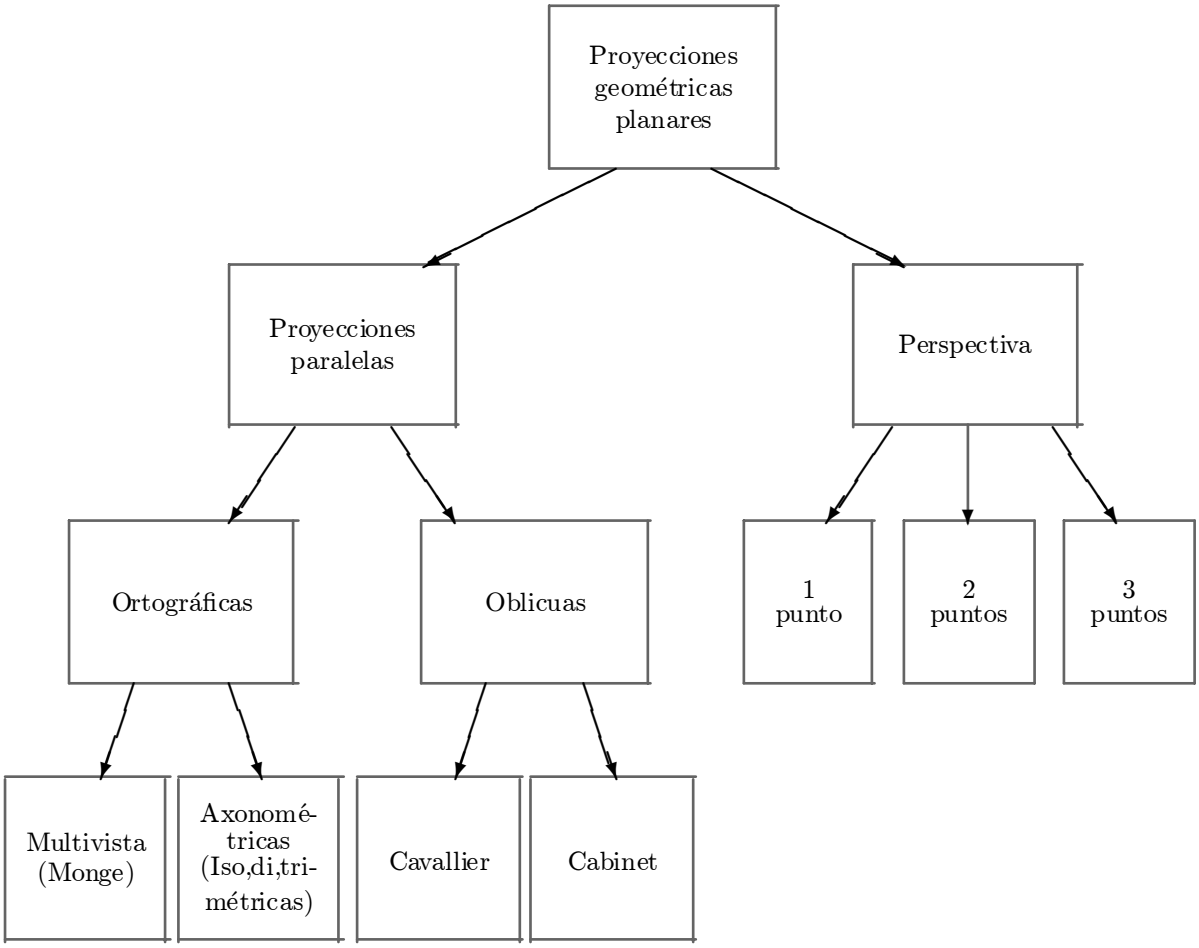


Figura 6.3 *Distintas clases de proyecciones geométricas planares.*

entidades proyectadas, pero descarta la componente z (ver Figura 6.4). Es decir,

$$x_p = x_o,\ y_p = y_o,$$

y por lo tanto la operación de proyección se puede representar por medio de una matriz (denominada matriz de proyección):

$$p = p_r \cdot o, \text{ es decir, } \begin{bmatrix} x_p \\ y_p \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}.$$

Normalmente se adopta la convención de que la proyección se efectúa sobre el plano del window (probablemente en NCS), por lo que es necesario pasar de las coordenadas homogenéas a las coordenadas del window:

$$x = \frac{x_p}{h}, \qquad y = \frac{y_p}{h},$$

y por último al viewport por medio de la transformación de windowing adecuada.

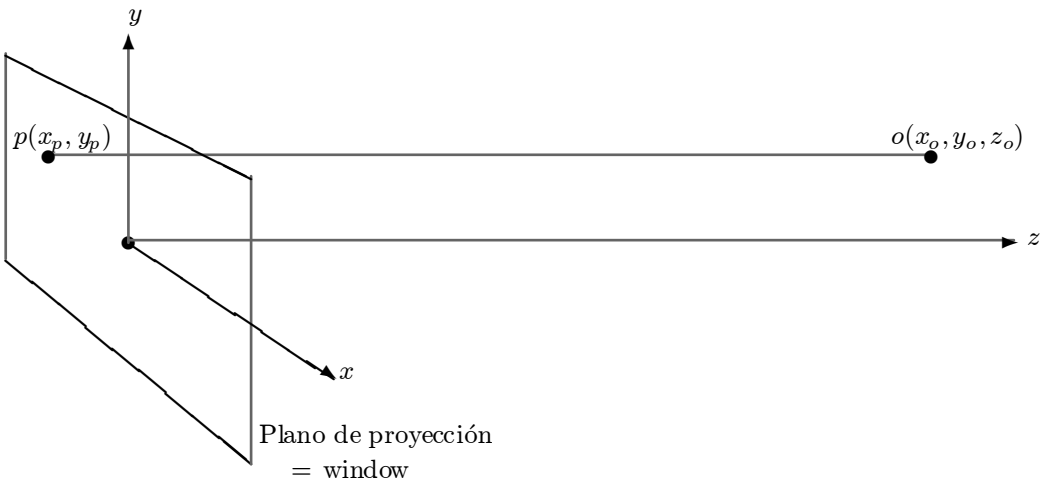


Figura 6.4 *Proyección ortográfica.*

Las proyecciones axonométricas (isométrica, dimétrica y trimétrica), son similares a las ortográficas, pero posicionan el plano de proyección en forma alabeada a los planos del sistema de coordenadas. Como los ejes proyectores son perpendiculares al plano de proyección, entonces los ejes dejan de ser paralelos a los ejes del sistema de coordenadas.

De esa manera, ubicando un plano de proyección en un lugar adecuado, es posible obtener en una única proyección toda la información necesaria, algo que con las proyecciones ortográficas es difícil y requiere varias vistas. Las proyecciones oblicuas, en cambio, hacen pasar ejes proyectores en forma oblicua por el plano de proyección, el cual sigue siendo paralelo a los planos del sistema de coordenadas. En ambos casos, se respeta la uniformidad de distancias en los tres ejes, aunque cada uno de ellos tiene una escala diferente.

Para la Computación Gráfica, la proyección perspectiva es de mayor utilidad porque, si bien de una manera limitada, produce un efecto geométrico similar al efecto visual en el ojo humano. La perspectiva consiste en hacer pasar los ejes proyectores entre el objeto a proyectar y un punto distinguido, denominado *foco* en el dibujo técnico, y *observador, punto de vista* o *cámara* en Computación Gráfica.

En la literatura es común encontrar planteos muy diversos (e innecesariamente complejos) para describir la perspectiva en términos de una transformación. Esto es así dado que es posible ubicar el plano de proyección y el foco en diversos lugares del espacio del mundo. Nosotros aquí emplearemos una técnica que simplifica enormemente la matriz resultante y los cálculos involucrados. Suponiendo que el plano de proyección es el plano xy, y que el observador está en $(0, 0, -1)$ mirando hacia la dirección positiva del eje z (ver Figura 6.5), entonces es posible plantear la siguiente semejanza de triángulos:

$$x_p = \frac{x_o}{z_o + 1}.$$

Lo mismo se efectúa para la coordenada y:

$$y_p = \frac{y_o}{z_o + 1}.$$

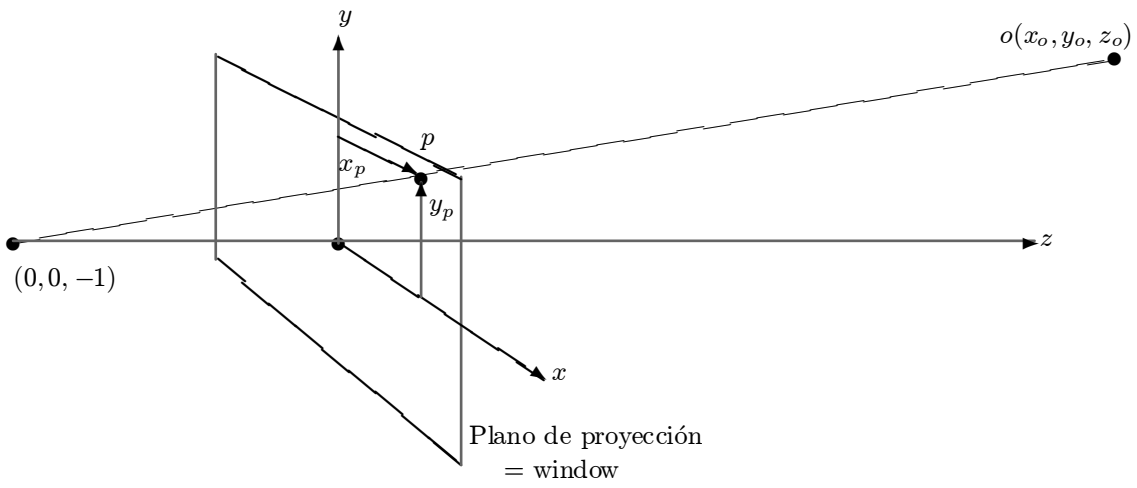


Figura 6.5 *Elección particular del plano de proyección y del foco para la proyección perspectiva.*

Una forma de combinar la división en la perspectiva con el pasaje de coordenadas homogéneas a comunes es utilizar la siguiente matriz de perspectiva:

$$p = p_r \cdot o, \text{ es decir, } \begin{bmatrix} x_p \\ y_p \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix},$$

y luego se pasa de las coordenadas homogeneas a las coordenadas del window:

$$x = \frac{x_p}{h} = \frac{x_o}{z_o + 1}, \qquad y = \frac{y_p}{h} = \frac{y_o}{z_o + 1}.$$

En el uso de la perspectiva es necesario realizar algunas consideraciones prácticas, dado que luego deben ejecutarse los algoritmos de cara oculta. Normalmente se realiza primero la transformación perspectiva sin proyección (es decir, sin eliminar la coordenada z) con una matriz

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Esta matriz puede premultiplicarse a todas las demás matrices de transformación, lográndose el pasaje del espacio del mundo (en el cual el área visible es un sólido con forma de pirámide truncada denominado "*frustum*") a un espacio en el cual el área visible tiene forma cúbica (ver Figura 6.12).

Como veremos en la Sección 6.5, luego de la perspectiva, el *clipping* es más sencillo porque los planos de *clipping* del nuevo espacio son paralelos a los ejes. Es necesario efectuar *clipping* con el plano $z = 0$ a fin de eliminar las partes que quedan "por delante" del plano de proyección, lo cual produciría un efecto inadecuado.

Al retener la coordenada z de los puntos transformados, es posible utilizar esta información para los algoritmos de cara oculta. Por ejemplo, los algoritmos basados en prioridades utilizan el

valor de dicha coordenada para ordenar las caras y de esa manera obtener el resultado deseado. Una vez efectuadas las manipulaciones necesarias, en el momento en el que se debe graficar una entidad, se efectúa una proyección paralela (es decir, se descarta la coordenada z).

La premultiplicación de todas las matrices involucradas en la tubería de procesos gráficos en 3D (ver Figura 6.13) tiene la siguiente forma:

$$\begin{bmatrix} e_x & r & r & t_x \\ r & e_y & r & t_y \\ r & r & e_z & t_z \\ p_x & p_y & p_z & e_h \end{bmatrix},$$

donde los factores e_x, e_y y e_z son factores de escala para las respectivas coordenadas, mientras que e_h es un factor de escala global que altera la coordenada homogénea. Todos los factores r producen rotación o inclinación de algún tipo. Los factores t_x, t_y y t_z producen traslación según las coordenadas respectivas, y los factores p_x, p_y y p_z, por último, producen una perspectiva (sin proyección) en la cual la escala es afectada inversamente por las coordenadas respectivas. Dependiendo de cuántos de estos factores sean no nulos, es el tipo de perspectiva (1 punto de fuga, o 2 puntos, o 3, ver Figura 6.3).

6.4 Primitivas y Estructuras Gráficas

La manera de estructurar las entidades gráficas en 3D sigue los mismos lineamientos generales que en 2D. Es posible realizar modelos de objetos relativamente complejos (mobiliarios, edificios, etc.) utilizando solamente instancias de cubos. La única diferencia, de gran importancia, es que es necesario observar una metodología más disciplinada para describir los objetos primitivos.

Como ya mencionáramos, la eliminación de líneas y caras ocultas es indispensable para lograr el realismo, y según veremos al final de este capítulo, los algoritmos que realizan estas tareas requieren una rápida y correcta identificación de determinadas características geométricas de las entidades gráficas. Lo mismo sucede con los modelos de iluminación y sombreado. De esa manera, en el momento de graficar una entidad, puede ser necesario ubicar el normal de sus caras, las aristas y vértices que concurren a las mismas, etc. Si todas estas determinaciones debieran realizarse por búsqueda, el tiempo de ejecución sería prohibitivo.

Por lo tanto, se recurre a describir las primitivas según un orden arbitrario pero estricto. Por ejemplo, dado el cubo de la Figura 6.6, numeramos sus vértices de modo que su ubicación en su propio espacio intrínseco esté relacionado con el índice del vértice. Luego buscamos que cada cara contenga la secuencia de vértices (ordenada por ejemplo en sentido antihorario si es vista desde afuera). Esto permite encontrar rápidamente la secuencia de aristas para dibujar la cara, y también permite encontrar el vector normal, dado que el mismo es el producto vectorial de dos aristas cualesquiera. Por ejemplo, el normal a la cara 1 es $(v7 - v5) \times (v8 - v7)$. Otra posibilidad es precalcular los normales y utilizar las matrices de transformación corriente para transformar los normales. Por último, puede ser de gran utilidad poder determinar cuáles son las caras que concurren a un vértice sin tener que recorrer la estructura.

En la Figura 6.7 observamos las declaraciones de tipos para una representación como la mencionada. Al igual que en el Capítulo 3, las entidades estructuradas se representan por medio de un registro variante, que puede apuntar tanto a un cubo o, recursivamente, a un (sub)objeto estructurado.

Dado un objeto estructurado en 3D, el algoritmo para graficarlo se basa en una recursión, cuyo caso base ocurre cuando el objeto es un cubo. En dicho caso, se premultiplica la matriz de instancia

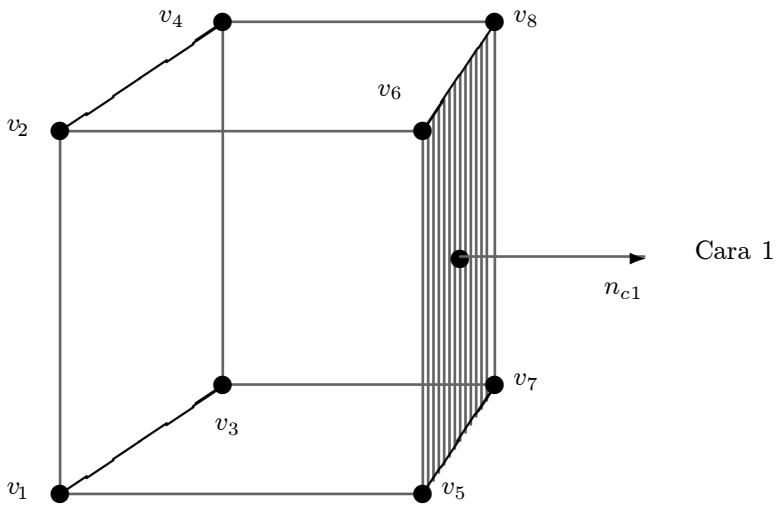


Figura 6.6 *Descripción ordenada de un cubo.*

```
type
      cubo =  array [1..8] of punto;
    transf = array [0..3] of array [0..3] of real;
 entidades = (cubs,defis);
      defi = array[0..9] of ^objeto;
    objeto = record
               t:^transf;
               case tipo:entidades of
                      defis:(d:^defi);
                       cubs:(c:^cubo);
             end;
```

Figura 6.7 *Declaraciones de tipos para representar objetos estructurados en 3D.*

del cubo (dónde debe aparecer, con qué tamaño, rotación, etc.) por la matriz de transformación "corriente", recibida por parámetro. Esta última transformación actúa como pila, según veremos. Con la matriz producto obtenida, se transforman los ocho vértices del cubo, y se grafican las doce aristas del mismo, siguiendo el orden correspondiente a su definición (por ejemplo, la mostrada en la Figura 6.6).

Si el objeto no es un cubo, entonces está estructurado a partir de subobjetos. En dicho caso, la parte recursiva del algoritmo aplica la matriz de instancia del objeto a cada subobjeto, y se llama recursivamente para cada subobjeto con la nueva matriz, la cual hace de matriz "corriente" de todos ellos. El procedimiento debe necesariamente terminar en el graficado de cubos (ver Figura 6.8).

```
procedure graf_cub(o:objeto;at:transf);
var t,t1:transf;
    cu,c:cubo;
    i:integer;
begin
  t:=o.t^;
  cu:=o.c^;
  matprod(t,at,t1);                    {preproducto por matriz corriente}
  for i:=1 to 8 do begin
   c[i].x:=cu[i].x*t1[0][0]+cu[i].y*t1[0][1]+      {transforma un vertice}
           cu[i].z*t1[0][2]+cu[i].w*t1[0][3];
  ...                                              {idem con los otros 7}
  end;
  linea(c[1],c[2]);    linea(c[2],c[3]);           {grafica las 12 aristas}
  ...
end;

procedure graf_obj(o:objeto;at:transf);
var t,t1:transf;
    d:defi;
    i:integer;
begin
  case o.tipo of
   cubs : graf_cub(o,at);     {si es cubo lo grafica con matriz corriente}
   defis: begin
       t:=o.t^;               {si es estructura}
       d:=o.d^;
       i:=0;
       while d[i]<>nil do begin          {para cada subobjeto}
            matprod(t,at,t1);            {nueva matriz corriente}
            graf_obj(d[i]^,t1);          {llama recursivamente}
            i:=i+1;
            end;
       end;
   end;
end;
```

Figura 6.8 *Algoritmo base para graficar un cubo, y recursivo para graficar un objeto estructurado en 3D.*

Para inicializar una estructura como la mostrada, debemos primero inicializar el cubo, dándole coordenadas a cada uno de sus vértices. De acuerdo a la Figura 6.6 utilizamos la asignación

```
v1 = (-1, -1, -1,  1)          v2 = (-1,  1, -1,  1)
v3 = (-1, -1,  1,  1)          v4 = (-1,  1,  1,  1)
v5 = ( 1, -1, -1,  1)          v6 = ( 1,  1, -1,  1)
v7 = ( 1, -1,  1,  1)          v8 = ( 1,  1,  1,  1).
```

```
procedure graficar;
var          cu : cubo;
   at,t1 ... t8 : transf;
      o1 ... o8 : objeto;
    mesa,escena : defi;
begin
 {matriz corriente = transformacion global}
 at[0][0]:=1;   at[0][1]:=0;   at[0][2]:=0;  at[0][3]:=0;
 ...  {otros 12 coeficientes}

 cu[1].x:=-1; cu[1].y:=-1; cu[1].z:=-1; cu[1].w:=1; {primer vertice}
 ...                                                {otros 7 vertices}

 t1[0][0]:=10;   t1[0][1]:=0;   t1[0][2]:=0;  t1[0][3]:=0;
 ...
 o1.t:=@t1;      o1.c:=@cu;

 {similar a lo visto en 2D}

 graf_obj(o8,at);     {llama al algoritmo recursivo}
end;
```

Figura 6.9 *Inicialización de estructuras.*

De esa manera, el algoritmo que grafica el cubo debe graficar las aristas que van del vértice 1 al 2, del 2 al 3, etc. También es necesario inicializar la matriz "corriente" con la tranformación "del mundo", la cual puede ser en principio la matriz identidad. Una vez establecidos estos datos, debemos comenzar a asociar a cada objeto con su matriz y definición correspondiente, la cual, como en 2D, es una lista de punteros a objetos, cada uno con su transformación y definición, y así recursivamente hasta llegar a los cubos. Una vez inicializada la estructura, se llama al algoritmo de graficación pasándole la referencia al objeto "tope" de la estructura, y con la matriz "corriente" (ver Figura 6.9).

Podemos ver en la Figura 6.10 el resultado de definir una escena como objeto estructurado. Primero se definió la mesa como cinco instancias de cubos, cada uno con su transformación correspondiente. Luego se definió la escena como cuatro instancias de mesa, cada una con su transformación.

6.5 Clipping y Transformación de Viewing

El clipping en 3D es más mandatorio que en 2D, dado que debemos eliminar lo que está detrás del plano de proyección para que el modelo de "camara" tenga sentido. Al mismo tiempo, suele agregarse un plano adicional con un z muy lejano, para no perder tiempo graficando objetos que no influyen en el resultado final de la escena. Como vimos, el primer paso antes del clipping es aplicar la transformación perspectiva (sin proyectar), para transformar el frustum o área visible (ver Figura 6.11) en una "caja" rectangular delimitada por planos paralelos y ortogonales.

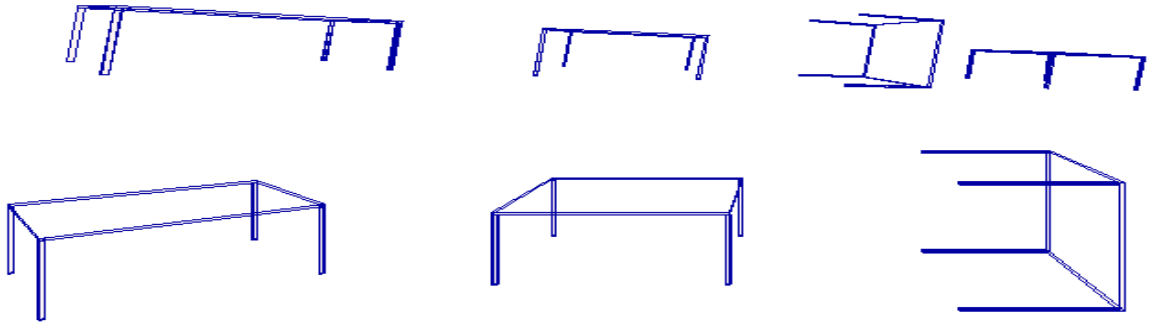


Figura 6.10 *Una escena 3D.*

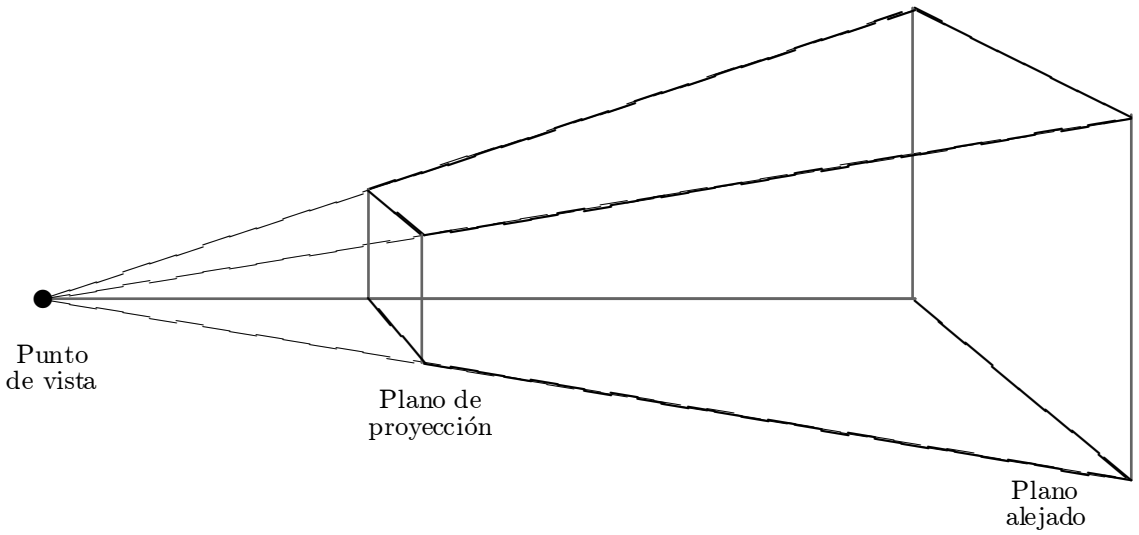


Figura 6.11 *El "frustum" o área visible.*

Entonces se introduce un nuevo espacio "del frustum", el cual está definido como el espacio rectangular que se obtiene luego de aplicar al mismo la transformación perspectiva (sin proyectar). La característica más importante de este nuevo espacio es que los planos de clipping dejan de ser oblicuos y pasan a ser paralelos (ver Figura 6.12).

Un algoritmo de clipping en dicha área procede de forma similar al Cohen-Sutherland en 2D, salvo que tenemos 27 áreas en el espacio, de las cuales solo una es visible, y las mismas se etiquetan con una séxtupla de booleanos. Las ventajas de proceder de esta manera son varias (pero es

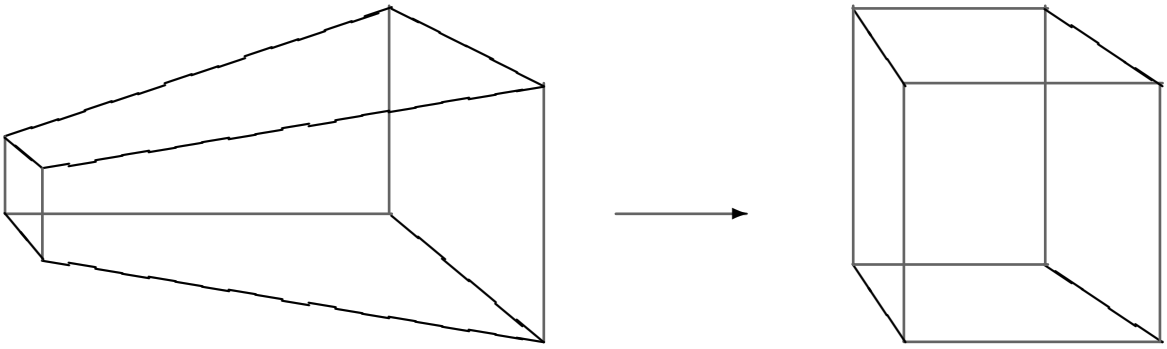


Figura 6.12 *El espacio del "frustum", luego de la transformación perspectiva.*

necesario recorrer algún camino para poder asimilar su importancia, por lo que recomendamos al lector que releea esta Sección luego de haber leído la Sección siguiente):

1. El pasaje de coordenadas homogéneas a comunes y la perspectiva se "enganchan" en una única operación que involucra solamente dos divisiones por cada punto procesado.
2. La matriz de transformación perspectiva puede premultiplicarse a todas las demás transformaciones corrientes en una única matriz de 4×4 que procesa todos los puntos del modelo de la escena de una manera eficiente.
3. Luego de la perspectiva, el clipping es mucho más sencillo porque solo hay que modificar levemente el algoritmo de Cohen-Sutherland ya implementado.
4. La eliminación de caras ocultas también es más sencilla porque si hay dos o más partes de la escena que "caen" en el mismo lugar de la pantalla, esto se puede descubrir porque tienen el mismo valor de x e y luego de la perspectiva.
5. Luego del clipping y de la cara oculta, la proyección es ahora una proyección paralela, la cual en definitiva es descartar la coordenada z.

Podemos en este punto intercalar una nueva transformación y un nuevo espacio, los cuales serán redundantes pero simplifican enormemente la descripción de determinados efectos, en particular al realizar animaciones. Cuando definimos la posición del observador o "camara" y su plano de proyección asociado, utilizamos un caso particular en el cual se simplificaban los aspectos matemáticos de la transformación perspectiva. Particularmente, se eligió en la Sección anterior que el observador esté en $(0, 0, -1)$, y que el plano de proyección sea el plano $z = 0$.

Sin embargo, un observador puede moverse en el espacio, y/o modificar la dirección hacia la cual está observando la escena. Por lo tanto, podemos considerar la existencia de un espacio del observador, en el cual el mismo está siempre en $(0, 0, -1)$ y el plano de proyección es el plano $z = 0$. Los cambios en la posición del observador se representarían con una transformación de viewing que modifica el espacio del mundo de modo tal que el punto $(0, 0, -1)$ del mundo pase a ser la posición del observador, y que el plano $z = 0$ pase a ser el plano de proyección.

Esto se podría representar *moviendo la escena*, por lo cual estrictamente no es necesaria una nueva transformación y un nuevo espacio. Sin embargo, mover la escena para acomodarse al

movimiento del observador es artificial, y destruye la información de la *verdadera* posición de los objetos y el observador dentro de la escena. Por otra parte, modificar la escena y modificar la posición del observador son operaciones conceptualmente distintas y es un error incorporarlas en una única matriz. Y si tuviésemos *dos* transformaciones del mundo, es fácil ver que la segunda sería *la inversa* de la transformación de viewing (mover el observador a la derecha es equivalente a mover la escena hacia la izquierda, rotar su plano de proyección un ángulo α alrededor de un eje es equivalente a rotar la escena $-\alpha$ alrededor del mismo eje, etc.

Podemos ver en la Figura 6.13 la integración de todos los procesos gráficos 3D en una única tubería. Los elementos restantes, correspondientes a los modelos de iluminación y sombreado que se verán en el Capítulo 7, se integran en la misma a la altura de los algoritmos básicos.

6.6 Cara Oculta

La eliminación de las partes de una escena que no deben verse es una tarea esencial en un sistema gráfico tridimensional con pretensiones de realismo. Pero por otra parte no existen soluciones generales que funciones con costos tolerables. Por ello existe una gran diversidad de métodos y técnicas *ad hoc* que se utilizan solos o en combinación y para casos particulares determinados. Entre las técnicas relativamente sencillas podemos mencionar las siguientes:

Orientación de caras: Es una de las primeras en utilizarse, y suele acompañar a otros métodos para reducir la complejidad. Cada cara tiene una orientación que determina cuál es la parte "exterior" de un objeto. Esto queda determinado por la dirección del normal de la cara (recordar el cuidado que tuvimos para definir los normales del cubo en la Figura 6.6) Por lo tanto, si una cara es mirada "desde atrás" es porque no debe ser visible. Este test puede realizarse de manera sencilla y rápida, dado que es el resultado del signo del producto escalar entre el normal a la cara y un vector que va de un vértice de la misma al observador. Si el signo es positivo, el normal apunta "hacia el mismo lado" que el observador, y la cara es visible. Este test se denomina "backface culling" (filtrado de las caras posteriores). Estrictamente, el método es correcto solo si la escena está compuesta por un único objeto cóncavo y de caras planas, aunque en la práctica se aplica *antes* de utilizar otros métodos más complejos, para reducir el cómputo en un factor de dos o tres.

Algoritmo del pintor: Debe su nombre a que se basa en una técnica utilizada por los pintores (en este punto debemos reconocer que la Computación Gráfica le debe varias cosas a los pintores renacentistas). Por ejemplo, Leonardo no puede tener a Mona sentada frente a un paisaje durante el tiempo necesario. Entonces va hacia el paisaje y lo pinta con toda tranquilidad, y luego va a la casa de Mona y la retrata *encima* del lienzo pintado con el paisaje. En términos de lo estudiado aquí, lo que Leonardo ha hecho es ordenar las entidades por su coordenada z y luego pintarlas de atrás hacia adelante. Este método tiene varias desventajas. Primero, es de complejidad $O(n^2)$, donde n es el número de caras. Segundo, no siempre es posible decidir cuál cara está delante. Tercero, es muy ineficiente porque se pintan pixels varias veces.

z-buffer: Es un algoritmo de gran popularidad por ser muy sencillo e implementable en hardware. Consiste en tener un buffer donde se almacena la coordenada z de cada pixel. Si durante la graficación se accede a un pixel con una profundidad menor que la almacenada, entonces se pinta el pixel y se actualiza el buffer (ver Figura 6.14).

Las desventajas del método es que utiliza una gran cantidad de memoria (usualmente cuatro veces más que el buffer de pantalla) y que es muy ineficiente porque un pixel puede ser pintado varias veces.

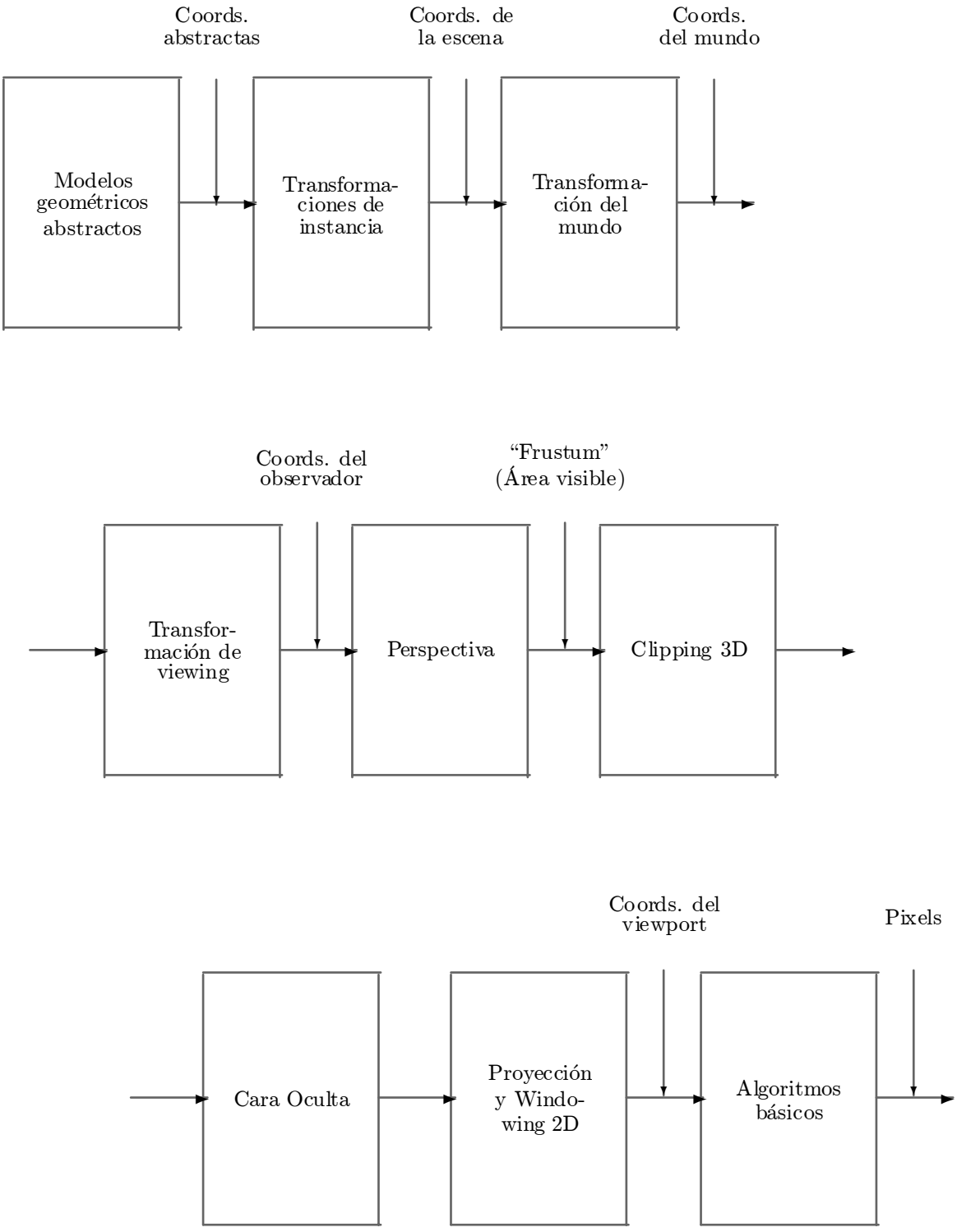


Figura 6.13 *La "tubería" de procesos gráficos en 3D.*

```
procedure zbuffer;
...
 zbuffer:=infinito;
 for cada-poligono do
   for cada-pixel do
     if pixel.z < zbuffer[pixel.x][pixel.y] then begin
           putpixel(pixel.x,pixel.y,poligono.color);
           zbuffer[pixel.x][pixel.y]:=pixel.z
     end;
 end;
```

Figura 6.14 *Esquema del algoritmo de z-buffer.*

Ray tracing: Para cada pixel se envía un "rayo" desde el observador hacia la escena. Si el rayo intersecta más de un objeto, se debe pintar con el color del objeto más cercano. Si no intersecta ningún objeto, entonces se pinta con un color "de fondo" (ver Figura 6.15). Estos algoritmos son sencillos y eficientes, y pueden aplicarse recursivamente para computar un modelo de iluminación híbrido pero de buenos resultados visuales, aunque las entidades gráficas representables deben ser de una geometría limitada [41]. Por dicha razón los consideraremos con cierto detalle en la Sección 7.5.

6.6.1 Eliminación de líneas ocultas en superficies funcionales

Este algoritmo fue desarollado por Wright [86] y es muy importante no solo por su aplicabilidad, sino porque es el primero en proponer un orden *front to back* como solución al problema de las entidades ocultas. El método está pensado primariamente para graficar funciones de dos variables, pero puede utilizarse también para representar arreglos bidimensionales de valores, lo cual es de gran utilidad en Computación Gráfica. Cada fila de dicha matriz representa el valor de $y = f(x, z)$ a un valor constante de z.

Es posible determinar cuál de los cuatro vértices de la matriz es el más cercano al observador, y se recorren las filas en orden, comenzando por dicho vértice. Al tiempo que se grafican las filas, se actualizan dos arreglos donde se almacena, para cada `x` de pantalla, el máximo y mínimo valor de `y`. Los valores de estos arreglos determinan la "silueta" de lo ya dibujado, la cual es un área dentro de la cual no debe dibujarse (ver Figura 6.16).

Es muy importante observar que para que el algoritmo funcione correctamente, las filas deben dibujarse de adelante hacia atrás. La implementación de este algoritmo es sencilla, dado que solamente debe modificarse el algoritmo de discretización de segmentos de recta para incluír la condición de no invadir la silueta de lo ya dibujado. Para mejorar el aspecto de lo graficado, normalmente se dibujan también las líneas a x constante (es decir, la poligonal que une los valores de las columnas). Una estrategia posible para hacerlo es "marcar" los valores que fueron visibles en la recorrida por filas y luego unirlos. Esto, sin embargo, tiene algunos problemas, dado que entre dos valores visibles puede haber una porción no visible. Otra solución es ejecutar el algoritmo por filas y por columnas utilizando siluetas independientes.

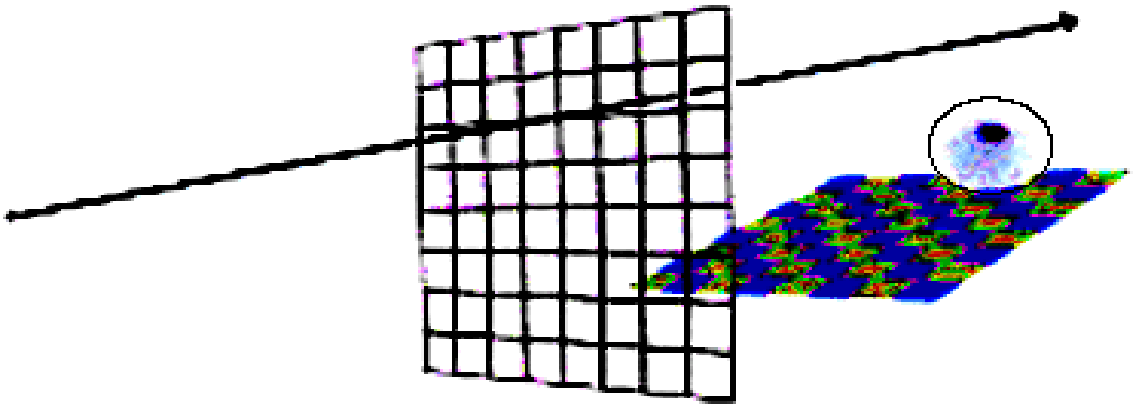
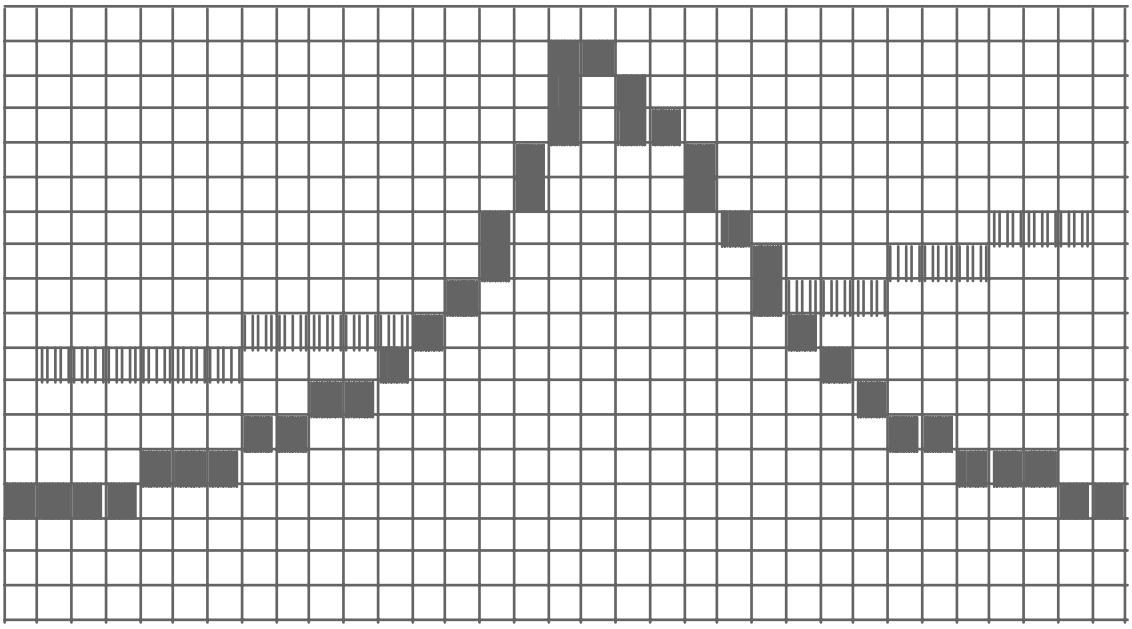


Figura 6.15 *Ray tracing.*



maxy	4	8	8	8	8	8	8	9	9	9	9	9	9	10	12	14	17	17	16	15	14	12	11	10	10	10	11	11	11	12	12	12	4
miny	4	4	4	4	5	5	5	6	6	7	7	8	9	10	12	14	17	17	16	15	14	12	11	9	8	7	6	6	5	5	5	5	4

Figura 6.16 *Estado intermedio durante el cómputo del algoritmo de Wright para superficies funcionales.*

6.6.2 Clasificación de los métodos generales

Las consideraciones de espacio impiden referirnos aquí a los métodos importantes de cara oculta con la profundidad necesaria. Por lo tanto intentaremos dar una breve reseña de los mismos, clasificándolos por la naturaleza de los algoritmos utilizados. Los lectores interesados en profundizar los conceptos pueden consultar la bibliografía recomendada.

Métodos en 3D: Los denominamos de esta manera porque efectúan el cómputo en el espacio del mundo, es decir, en tres dimensiones y con precisión `real`. Por lo tanto, el resultado es independiente del dispositivo, y se pueden utilizar para dispositivos de vectores. Tienen, sin embargo, la desventaja del elevado costo computacional. Además, estrictamente computan líneas ocultas, no pudiendo pintarse las caras.

Roberts: Utiliza una representación biparamétrica de cada arista. Uno de los parámetros permite recorrer un punto de vértice a vértice, y el otro parámetro va de dicho punto al observador. Cada objeto de la escena se representa con una colección convexa de caras orientadas. De esa manera, es posible utilizar las técnicas de programación lineal para resolver si la formulación biparamétrica tiene una solución *dentro* de alguno de los objetos convexos o no. Si así sucede, entonces parte de la arista es no visible.

Appel: Representa cada cara como una colección orientada de aristas. Mientras se recorre cada arista, se verifica que la proyección de la misma no intersecte la proyección de otra arista. Si ocurre una intersección con una arista orientada positivamente y que está por delante, entonces se incrementa en uno un contador de "visibilidad cualitativa". Si la misma está orientada negativamente, entonces se decrementa. La recorrida de la arista se realiza pintando mientras la visibilidad cualitativa sea cero.

Métodos en 2.5D: También denominados métodos de prioridades, dado que trabajan en el espacio del mundo en x e y, pero establecen una jerarquía cualitativa en z. Todo método en el que se ordenen los objetos por profundidad antes de graficarlos cae en esta categoría. Por lo tanto, tienen un costo $O(n^2)$.

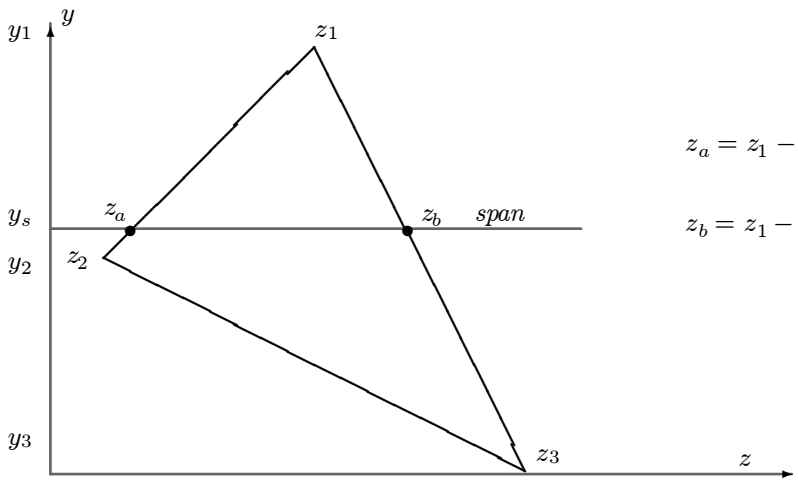
Newell et. al.: Es un caso particular del algoritmo del pintor, en el cual se realiza un esquema de ordenación similar al bubble-sort o burbujeo. El algoritmo recorre la lista de caras comparándolas de a pares, llevando hacia abajo en la lista a la que aparezca detrás. Una vez que ubicó la más alejada del observador, la grafica e itera con las caras restantes. Esta comparación de la profundidad entre caras es bastante problemática. En algunos casos se puede determinar (si el z de todos los vértices de una está por delante o por detrás del z de todos los vértices de la otra), pero en general es bastante difícil, y en el caso del "solapamiento cíclico" de caras no hay una solución automática.

Schumacker: Irónicamente, fue el desarrollador de muchos métodos que actualmente se utilizan en los video juegos, simuladores de carreras, por ejemplo. La idea principal es que las escenas pueden descomponerse en "clusters" o racimos de caras, de modo tal que las caras de un cluster que está por delante de otro pueden dibujarse sin efectuar la comparación por profundidad. De esa manera se puede encontrar una asignación estática de prioridades a los objetos. Por ejemplo, en un simulador de carreras, el cielo y el paisaje de fondo están siempre detras de la pista, la pista está siempre detrás de los demás automóviles, los cuales están siempre detrás del volante del conductor. También puede encontrarse una prioridad dinámica precomputada, en función de la posición del observador. Por último, es posible encontrar una asignación de prioridades entre las caras de cada cluster. Este tipo de métodos produce resultados aptos para la animación en tiempo real, pero la asignación de prioridades debe hacerse en forma estática y compilarse dentro del programa.

Wolfenstein: Comenzó a utilizarse en las implementaciones de video-juegos de la serie *Doom*, *Hexxen* y similares. Por lo tanto es muy reciente y no está debidamente documentado.

Las escenas en estos juegos suelen ser polígonos con texturas, las cuales tienen un significado indispensable para el juego. El algoritmo consiste en trazar un rayo desde el observador a un punto dado de cada polígono, para determinar el punto de entrada al mapa de texturas. De esa manera, si bien se comete un error geométrico (más importante cuanto más próximo es el polígono al observador), es posible determinar rápidamente el orden de graficación, y acelerar el uso de mapas de texturas.

Métodos en 2D: Trabajan en el espacio de la imagen, recorriendo por línea de barrido. Por dicha razón se los denomina métodos scan-line. Son los más eficientes en tiempo, aunque la precisión depende del dispositivo gráfico de salida. Además trabajan bien en relación con los algoritmos de iluminación y sombreado. Podemos citar trabajos de varios autores (Wilye, Romney, Watkins), pero todos coinciden en procesar la escena por líneas a y constante. En cada línea es posible determinar las caras *activas*, es decir, aquellas que tienen por lo menos un pixel con dicha coordenada y. Para todas las caras activas se realiza la conversión scan (como la vista en la Sección 2.6 pero para solo una línea, denominada *span*). Durante dicha conversión es posible computar el z de cada uno de los pixel del comienzo y final del span (ver Figura 6.17). Por lo tanto, es posible determinar de una manera sencilla cuándo un span está por delante de otro, simplemente teniendo en cuenta que son segmentos de recta y que su intersección es fácil de calcular (ver Figura 6.18).



$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

Figura 6.17 *Determinación del valor de z de cada pixel en un span.*

Métodos de subdivisión recursiva: La esencia de algunos métodos sencillos y eficientes como el clipping de Cohen-Sutherland está en poder descomponer recursivamente el problema en casos triviales. La idea de recursión se puede extrapolar a la solución del problema de la cara oculta. De esa manera, existen algoritmos que se basan en una subdivisión recursiva del espacio del mundo, o del espacio de la imagen. Entre los primeros se cuentan el algoritmo BSP (Binary Space Partition) que ordena un conjunto de caras por subdivisión binaria. Se toma una cara del conjunto, y las restantes pasan a estar orientadas positiva o negativamente. Cada uno de dichos subconjuntos es luego procesado recursivamente, de una manera similar al *quicksort*. Otra técnica de subdivisión recursiva del espacio del mundo es utilizar *octoárboles*, los cuales permiten subdividir recursivamente la escena hasta un nivel de resolución ("*voxels*") manejable.

Entre los algoritmos de subdivisión recursiva del espacio de la imagen podemos mencionar al de Warnock, que interroga por el contenido del viewport. Si todo el viewport contiene un único objeto, o ningún objeto, entonces se lo pinta con el color del objeto o con el color de

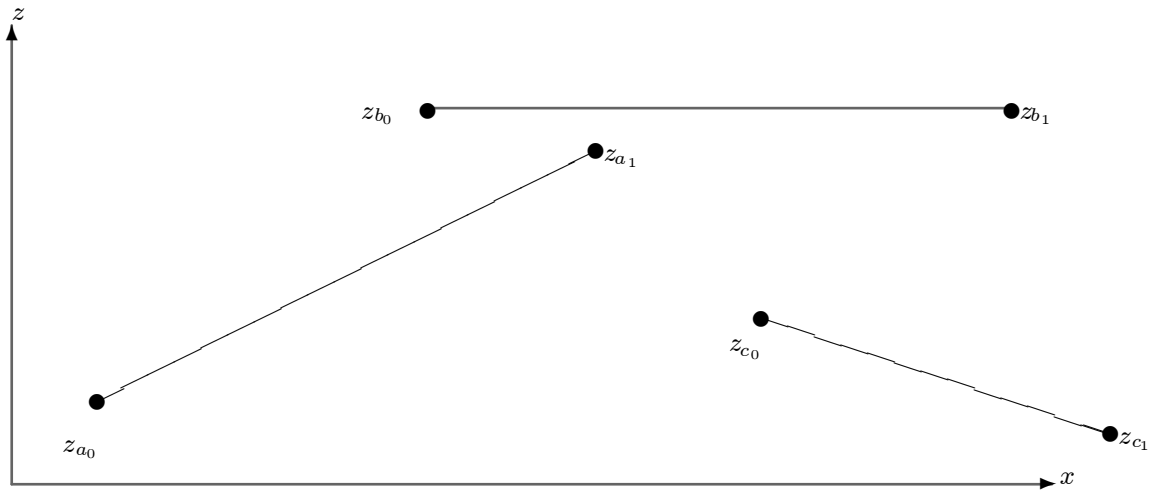


Figura 6.18 *Decidiendo cuándo un span está delante de otro.*

fondo, respectivamente. En caso contrario, el viewport se subdivide en cuatro, y a cada uno de los sub-viewports se lo procesa recursivamente con el mismo algoritmo.

6.6.3 El algoritmo SPIII

El algoritmo que describiremos ahora se basa en una combinación de las técnicas de prioridades, áreas de invisibilidad, y scan-line. Sin embargo, es de implementación relativamente sencilla (con respecto a los otros métodos), de ejecución eficiente, y posee pocos casos particulares en los que fracasa. Su descripción puede hacerse en dos líneas conceptuales:

1. Ordenar los polígonos por profundidad.
2. Dibujarlos de adelante hacia atrás, manteniendo en una estructura las áreas de invisibilidad.

Podemos ver en la Figura 6.19 el resultado de dibujar dos triángulos con este método.

Una técnica práctica para efectuar el ordenamiento entre caras es utilizar una variante del *radix-sort* [57] en función de las profundidades de las caras. Sea n la cantidad de caras a ordenar, entonces cada cara tiene un z mínimo, es decir, la profundidad del vértice más cercano al plano de proyección. Es posible dimensionar un arreglo de n lugares para distribuir las caras en el mismo. Cada lugar del arreglo está en correspondencia con la profundidad z del vértice más cercano de una cara según la ecuación

$$I = \frac{n(z - minz)}{maxz - minz},$$

donde $minz$ y $maxz$ son las profundidades del vértice más cercano de la cara más cercana y de la más lejana, respectivamente (ver Figura 6.20).

Idealmente, si las caras están uniformemente distribuidas en profundidad, es posible que corresponda una sola cara a cada lugar del arreglo. En la práctica pueden existir lugares vacíos y

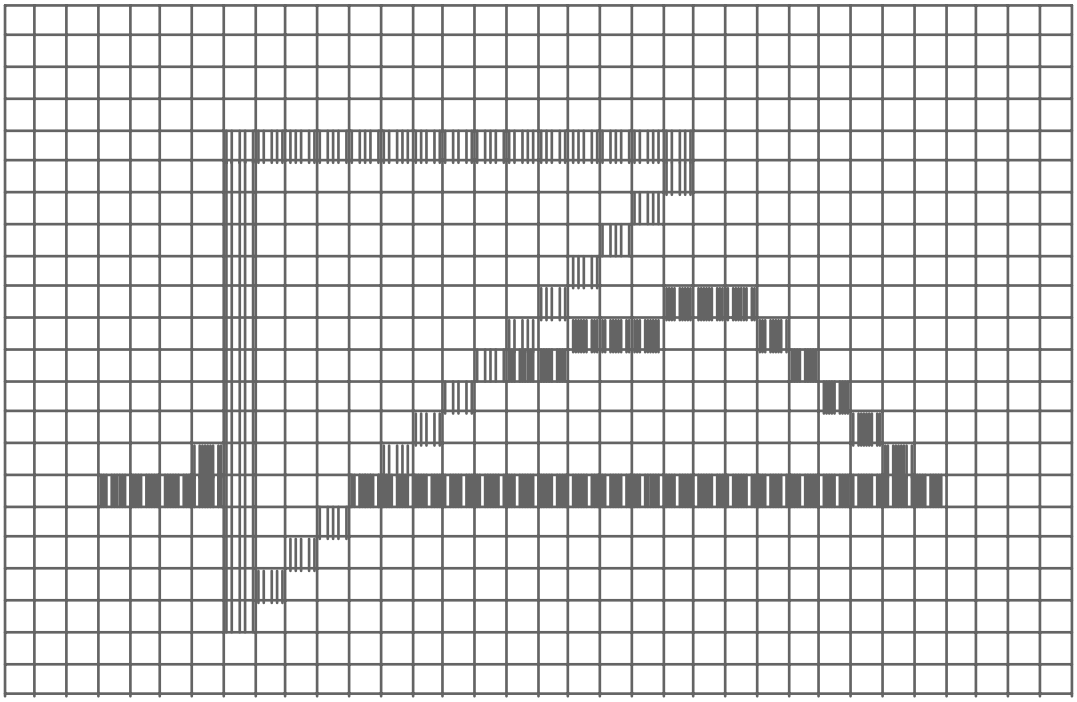


Figura 6.19 *Resultado de graficar dos triángulos con el algoritmo SPIII de cara oculta.*

lugares con varias caras. Lo importante es que las mismas pueden procesarse por orden, y en el caso de tener varias caras en una misma profundidad, de acuerdo al tipo de escena y de los objetos que se dibujan (superficies funcionales como las que veremos en el Capítulo 8, por ejemplo), normalmente no hay que ordenarlas entre sí. De todas maneras, en caso de tener que ordenarlas, se trataría siempre de un subconjunto muy pequeño del conjunto total de caras, y la complejidad del ordenamiento no se alejaría de $O(n)$ [27].

Con respecto al dibujado de las caras con áreas de invisibilidad, la idea se basa en el algoritmo de Wright para superficies funcionales, pero acomodada para trabajar en forma similar a los algoritmos scan-line. Esto se hace así dado que es posible utilizar el mismo cómputo para implementar los algoritmos de iluminación y sombreado que estudiaremos en el Capítulo 7. Esta es una de las razones principales de la practicidad y eficacia del algoritmo SPIII, dado que los demás algoritmos de cara oculta no están pensados para facilitar el sombreado de las caras.

Las áreas de invisibilidad se computan manejando una lista de pares para cada línea de barrido. Dichos pares contienen el menor y mayor x de cada cara dibujada en una línea y dada (ver Figura 6.21). Durante la conversión scan de cada polígono, entonces, es necesario chequear que para un dado y, el x del pixel a pintar no esté comprendido dentro del área de invisibilidad detrminado por cada uno de los pares existentes en dicho y, específicamente, que no sea menor que el mayor x y mayor que el menor x de alguno de los pares.

Como puede verse en la Figura 6.21, cuando las áreas de invisibilidad de dos polígonos se solapan, es posible utilizar un único par. Esta técnica produce una disminución notable en el tiempo de ejecución, dado que la cantidad de tests de invisibilidad se reduce considerablemente [27].

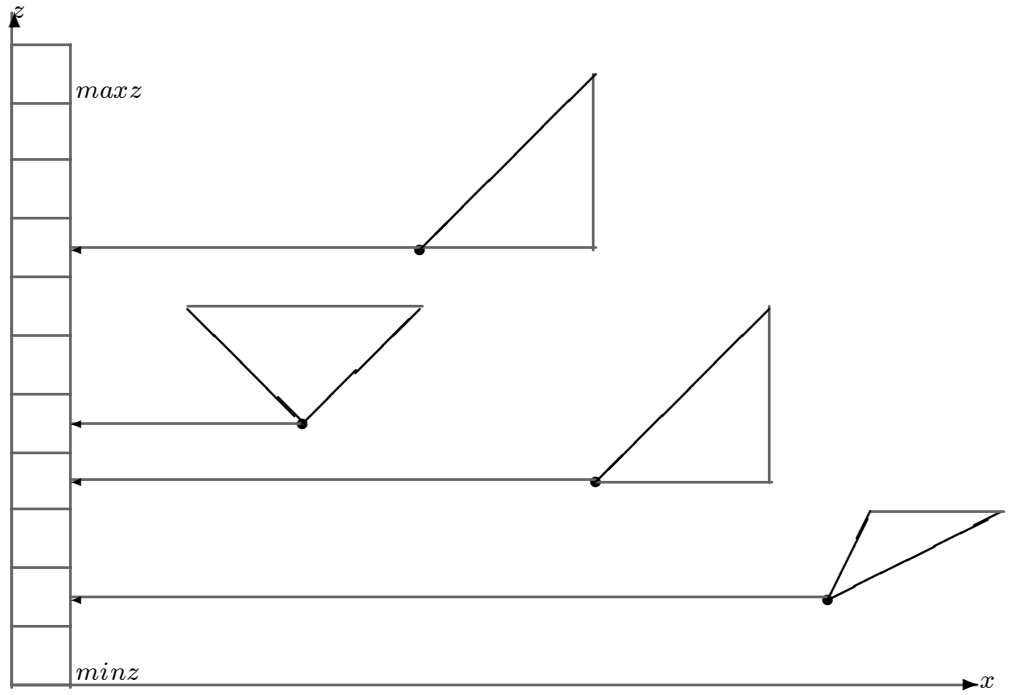


Figura 6.20 *Ordenamiento de caras en el algoritmo SPIII.*

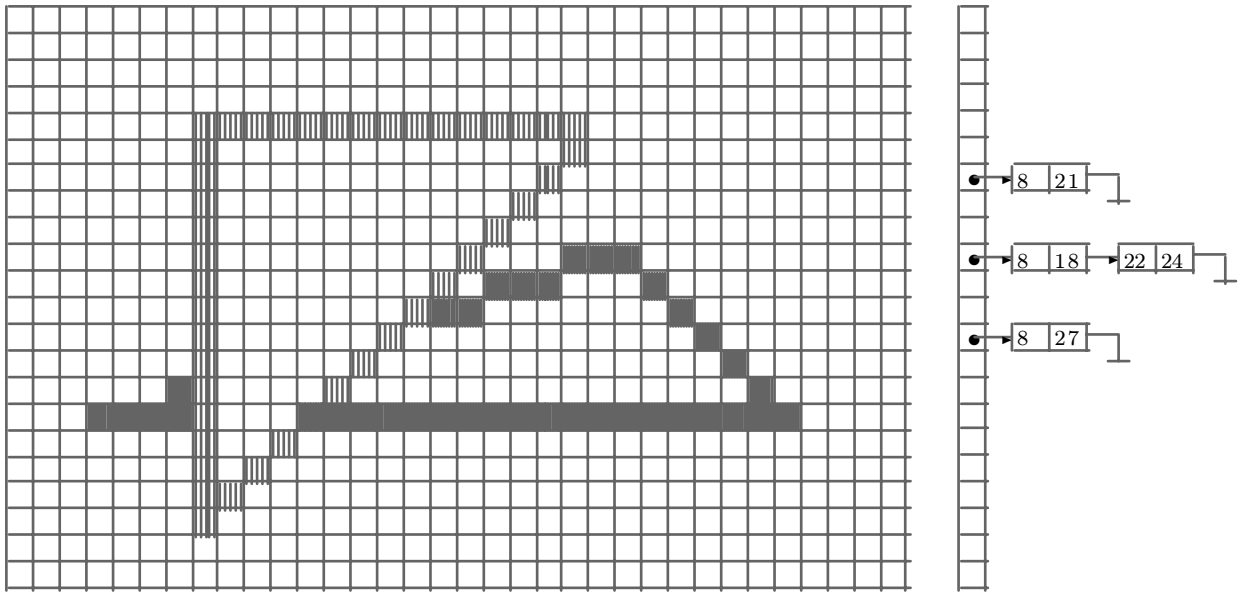


Figura 6.21 *Mantenimiento de las áreas de invisibilidad (se muestran solo algunas).*

Podemos ver en la Figura 6.22 el resultado de aplicar este algoritmo a una compleja estructura de caras.

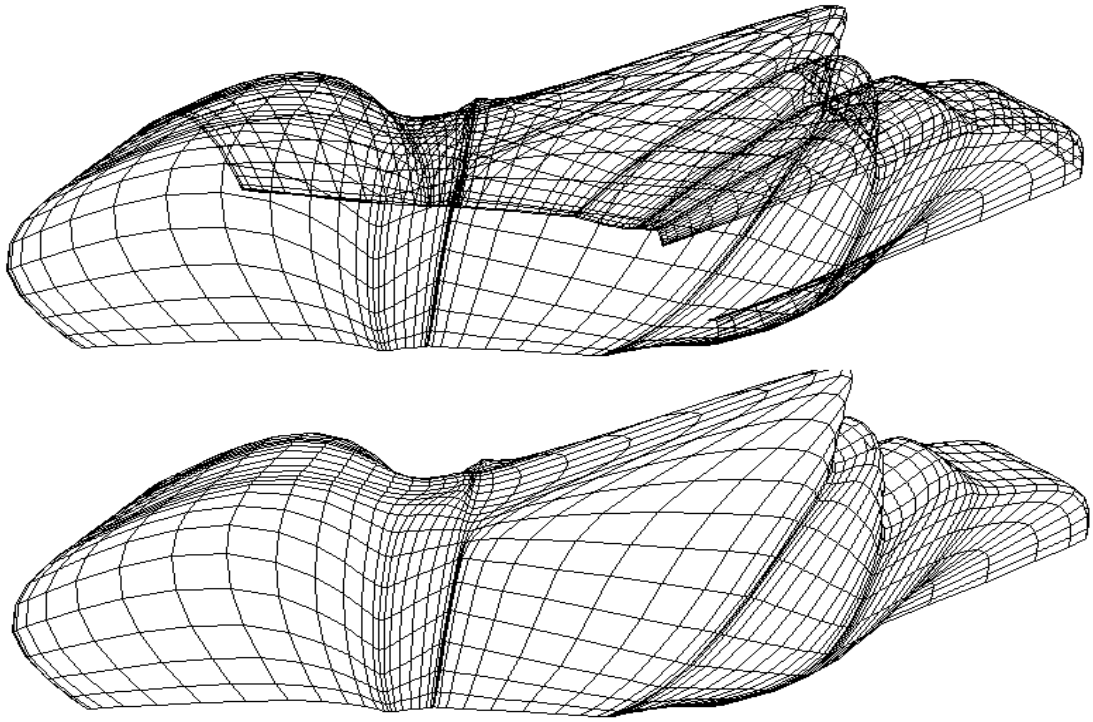


Figura 6.22 *Algoritmo SPIII aplicado a una estructura de caras.*

6.7 Ejercicios

1. Implementar un objeto estructurado en 3D y graficarlo con proyección paralela para varias transformaciones y posiciones del objeto.
2. Repetir el ejercicio anterior pero con proyección perspectiva.
3. Dado un objeto en 3D, comparar el resultado de graficarlo con distintas escalas, graficarlo acercándolo al plano de proyección, y graficarlo con el observador alejándose del plano de proyección.
4. Repetir el ejercicio anterior utilizando backface culling. Recordar aplicar primero la perspectiva, luego la eliminación de caras ocultas y por último la proyección.
5. Repetir el ejercicio 3 pero utilizando el algoritmo del pintor y efectuando la conversión scan de las caras.
6. Implementar el algoritmo de Wright para superficies funcionales.
7. Implementar el algoritmo SPIII.

6.8 Bibliografía recomendada

El manejo de las transformaciones en 3D y las coordenadas homogéneas puede consultarse en el Capítulo 5 del libro de Foley et. al. [33] y el Capítulo 22 del libro de Newmann y Sproull [66]. La descripción de los modelos estructurados de entidades gráficas puede consultarse en el capítulo 22 del Newmann-Sproull [66], y algunos aspectos avanzados en el Capítulo 3 del libro de Giloi [40] y el Capítuo 7 del Foley.

El tema de las proyecciones y la transformación de viewing está tratada de una forma compleja y algo confusa en el Capítulo 23 del Newman-Sproull. Un tratamiento más amplio figura en el Capítulo 6 del Foley, aunque tampoco se llega a una formulación sencilla. La descripción definitiva puede verse en [10]. Aquellos interesados en conocer en detalle los aspectos matemáticos de las proyecciones planares pueden consultar [19].

La interrelación entre las coordenadas homogéneas, la perspectiva y el clipping son bastante más profundas de lo expuesto en este Capítulo. Una discusión adecuada, junto con la explicación del costo computacional y los problemas en la representación numérica de estas operaciones puede leerse en [14].

La descripción y caracterización de algoritmos de línea y cara oculta se basa en el trabajo de Sutherland et. al. [78], el cual pese a su antiguedad sigue vigente y es de lectura muy recomendable. La descripción de las técnicas más primitivas y algunos métodos más modernos puede consultarse en el Capítulo 15 del Foley. También puede encontrarse en el Capítulo 5 del Giloi la descripción de algunos algoritmos competitivos y su comparación con los tradicionales. El funcionamiento del algoritmo de Wolfenstein me fue comunicado por Gustavo Patow. Los detalles de implementación, resultados, y técnicas asociadas al algoritmo SPIII pueden consultarse en [27].

7 Modelos de Iluminación y Sombreado

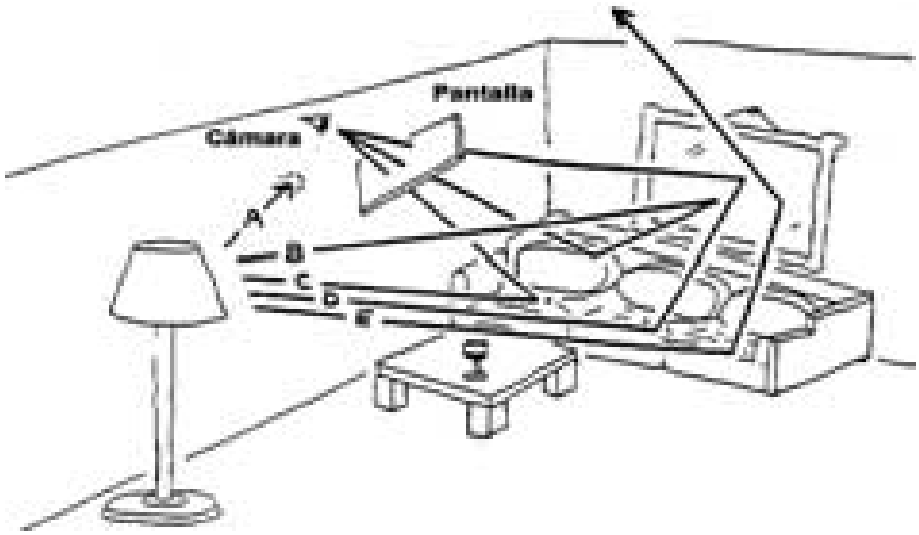


Figura 7.1 *Una escena sintética y sus condiciones de iluminación*

7.1 El Realismo en Computación Gráfica

Como ya mencionáramos en Capítulos anteriores, la búsqueda del realismo constituyó un cambio de paradigma en los objetivos de la Computación Gráfica durante la década del 70. El objetivo pasó a ser la creación de imágenes de escenas tridimensionales que parecieran *reales*. Una definición de realismo podría desembocar en un debate. El sentido que se le da en nuestra área está relacionado con la mayor fidelidad de la representación de atributos en los modelos computacionales subyacentes al proceso de generación de la imagen.

Por lo tanto es posible hablar de modelos "más" o "menos" realistas en función de la mayor o menor cantidad de características representadas, fundamentalmente en los modelos de interacción entre la luz y los objetos, y de la fidelidad con la que se representan dichas características. En un extremo están los modelos *fotorrealistas*, que buscan competir en fidelidad con los resultados fotográficos (ver [2]), mientras que en el otro están los modelos tan sencillos como la representación del *wireframe* o "armazón de alambre" que estamos manejando hasta ahora en este libro, modelos en los cuales los objetos son representados solamente con algunos vértices seleccionados y las aristas que los unen.

El realismo de una imagen tridimensional depende de la correcta simulación de los efectos de iluminación y sombreado. Se utiliza un modelo de iluminación para calcular las intensidades y colores de cada pixel de la imagen, mientras que las técnicas de sombreado permiten reducir la cantidad de pixels para los que se debe computar el modelo de iluminación. El modelo de iluminación representa una simulación de los fenómenos que, modificando o filtrando la distribución de energía de la luz incidente, dan lugar a los colores en las imágenes. Por ejemplo, en la Figura 7.1 se ilustra el efecto de querer simular una determinada condición de iluminación para una escena sintética, vista desde un determinado lugar. Los seis fenómenos fundamentales derivados de la interacción entre luces y objetos son reflexión, transmisión, absorción, difracción, refracción e interferencia. Aunque todos ellos se han modelado en computación gráfica, el más importante visualmente es la reflexión.

Los modelos de reflexión que se estudiaron en la década del 70 no buscaron simular las ecuaciones integrales que reproducen exactamente muchos efectos visuales (basadas en la teoría de ondas electromagnéticas y en las ecuaciones termodinámicas), sino que buscaron modelos empíricos que fueran un buen compromiso entre las restricciones en tiempo y equipo disponible y la simulación del comportamiento de la luz en la realidad. Ya aparecía el sombreado de objetos a fines de los '60 (en [87] y [83]) computándose con funciones inversamente proporcionales a la distancia al foco luminoso. Ahora se ignora casi completamente el efecto de la distancia en el cálculo de iluminación excepto en la simulación de niebla. Probablemente el primer modelo empírico se debe a Bouknight (en [16]), donde se evaluaba la intensidad de cada faceta de un objeto de acuerdo a su orientación respecto a la fuente de luz, en base a la ley de Lambert, y también incluía un término de luz ambiente para iluminar las partes no enfrentadas con fuentes de luz.

El modelo de Phong [69], que con el tiempo llegó a convertirse en el estándar, divide la reflectividad en componentes difusa y especular (brillo). Este es un modelo de reflexión especular local, es decir, no tiene en cuenta las interreflexiones entre objetos de la escena. Sin embargo, es elegante y produce resultados satisfactorios a la vista. El paso más importante en esta línea fue dado por Whitted [85], quien utiliza el modelo de Phong localmente pero exitende el modelo por medio de la técnica de trazado de rayos (ray tracing) utilizada por Appel en 1968 para determinar la cara oculta (ver Sección 4.6). Dichos rayos permiten computar la interreflexion entre objetos, y por lo tanto un modelo de iluminación no local.

Desde entonces, el tema del realismo ha experimentado un desarrollo notable. Al mismo tiempo han surgido nuevos modelos que retoman la idea de simular los fenómenos ópticos desde el punto de vista del electromagnetismo (por ejemplo, la ecuación del rendering de Kajiya) o del equilibrio termodinámico (por ejemplo, la radiosidad, de Greenberg et. al.). En este Capítulo presentaremos los elementos fundamentales de los modelos de locales de iluminación y de las técnicas de sombreado. Esto permite concluir con el modelo de rendering denominado *scan-line*, el cual es la base de este texto. También describiremos brevemente los modelos de iluminación no locales, los cuales dan origen a dos modelos de rendering: el trazado de rayos o ray tracing, y la radiosidad. Por último, describiremos algunas técnicas que permiten mejorar la representación de diversas cualidades por medio de mapas de atributos.

7.2 Modelos de Iluminación

El objetivo de un modelo de iluminación es representar los fenómenos óptico-físicos que determinan el color con que debe graficarse una superficie en un punto dado. Pueden tenerse en cuenta distintos factores, y darle diferente importancia a cada uno. Por lo tanto existen varios modelos de iluminación, cada uno expresado mediante a una ecuación de iluminación que determina el color de un punto en función de la geometría, posición y orientación de la superficie a la que pertenece, el material que la compone, las fuentes de luz que la iluminan, y la interacción luminosa con otros objetos.

Es importante tener en cuenta que estas ecuaciones normalmente no provienen de soluciones simplificadas a las ecuaciones de la óptica o de la propagación de la radiación electromagnética. Por el contrario, tienen más bien una base empírica que la justifica por los resultados obtenidos en la práctica. Esto es así porque los modelos físico-matemáticos de la propagación luminosa son de una complejidad en principio inmanejable, aún para escenas sencillas. Por otro lado, la computación de los caminos luminosos posibles en una escena es un problema similar pero mucho más complejo que el de la eliminación de las partes ocultas al observador, que ya estudiáramos en Capítulos anteriores. Por lo tanto, es necesario contar con un modelo de iluminación, aunque sea una aproximación muy cruda, que permita comenzar a recorrer el camino ascendente del realismo.

Una de las primeras simplificaciones consiste en omitir del modelo la interacción luminosa entre objetos, la cual, sin duda, es la parte más compleja del cálculo de una ecuación de iluminación. Este tipo de modelos se denominan *locales*, dado que resuelven la ecuación de iluminación de un punto solamente a través de la información geométrica local a dicho punto. Los modelos que estudiaremos en esta Sección son de estas características, dejando para la Sección 7.4 una breve reseña de las ecuaciones de iluminación basadas en modelos físicos, para la Sección 7.5 el tratamiento de modelos no locales basados en aplicaciones recursivas ray tracing y para el próximo Capítulo una introducción a los modelos de refracción.

Los modelos de iluminación que presentaremos en esta Sección representan la combinación de diversos factores, como por ejemplo la radiación luminosa de cada objeto, la reflexión de luz proveniente de fuentes de iluminación, etc. Una caracterización fiel de las mismas desde el punto de vista de la teoría del color significaría representar completamente la interacción para cada frecuencia visible del espectro. En la práctica esto es innecesario, porque como vimos en el Capítulo 5, el ojo transforma dicha información espectral a una representación cromática simplificada. Por lo tanto, es solo necesario computar el modelo de iluminación en ciertas frecuencias espectrales, por ejemplo en el modelo RGB.

El modelo de iluminación más sencillo es considerar que los objetos radian su propia energía luminosa, sin interactuar con fuentes de luz. Por lo tanto, cada objeto se muestra con un color constante propio

$$I = k_i,$$

donde I es la intensidad resultante, y k_i es la intensidad luminosa propia del objeto i. En el modelo RGB debemos computar tres veces esta ecuación, una para cada primario, donde tendríamos tres intensidades resultantes y tres intensidades propias de cada objeto, que determinan específicamente su color. Este modelo es obviamente muy simplificado, pero tiene la ventaja de que se evalúa solo una vez por objeto.

Para poder comprender de qué manera se representan los demás factores en un modelo de iluminación en su correspondiente ecuación, necesitamos previamente conocer y clasificar los mecanismos por medio de los cuales es transportada la energía luminosa. En particular, nos concentraremos primero en la reflexión.

Físicamente existen dos posibilidades cuando un fotón incide sobre un objeto: el mismo es absorbido y por lo tanto su energía se disipa en forma de calor, o bien es vuelto a emitir. Este último fenómeno reconoce otras dos posibilidades: o bien el fotón no encuentra un lugar en el espacio de energías del objeto, y por lo tanto "rebota" balísticamente, o bien es absorbido temporalmente excitando la órbita de un electrón en un centro de color, para luego ser reemitido.

Estos tres fenómenos se denominan, respectivamente, absorción, reflexión especular y reflexión difusa, y la energía luminosa incidente total debe repartirse entre los tres. Los adjetivos "especular" y "difuso" se refieren a que en el primer caso los fotones literalmente "rebotan" y por lo tanto la geometría del rayo luminoso reflejado puede conocerse a partir de la geometría del rayo incidente y de la superficie del objeto, mientras que en el segundo caso dicha relación se pierde, debiendo utilizarse factores estadísticos para simularlo.

Por su parte, las fuentes de luz pueden producir un haz luminoso geométricamente coherente, en cuyo caso hablamos de "iluminación puntual" dado que la dirección de un rayo luminoso en un punto puede computarse como la dirección entre la fuente luminosa y dicho punto. Sin embargo, al existir fenómenos de reflexión difusa, es posible que la coherencia geométrica se rompa, y que por lo tanto la energía luminosa en un punto deba computarse con una función más compleja. Una aproximación a este fenómeno es considerar que en todo punto y en toda dirección existe una intensidad luminosa determinada, a la cual se denomina "iluminación ambiente".

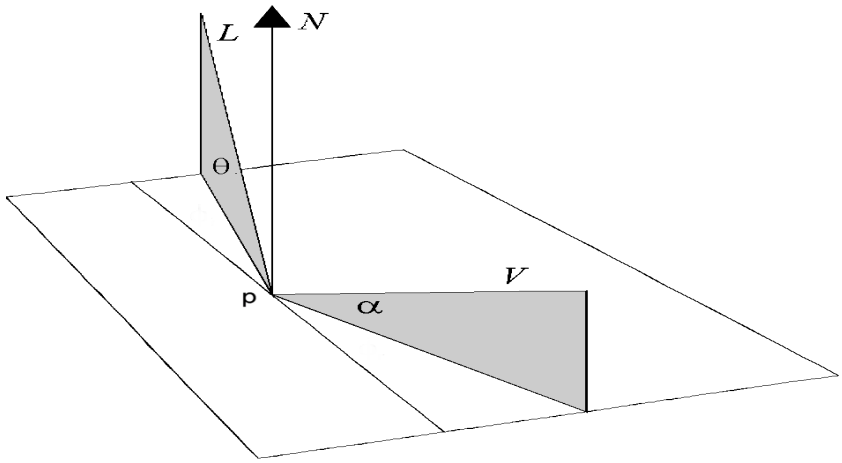


Figura 7.2 *Vectores asociados al modelo de reflexión difusa.*

Tenemos por lo tanto que considerar cuatro mecanismos de transporte: la reflexión especular de la iluminación puntual, la reflexión difusa de la iluminación puntual, la reflexión especular de la iluminación ambiente, y la reflexión difusa de la iluminación ambiente. Normalmente los dos primeros mecanismos son fácilmente calculables, mientras que los otros dos no, y por lo tanto se los aproxima con un único "factor ambiente". De esa manera tenemos

$$I = I_a k_a,$$

donde I_a es la intensidad de la iluminación ambiente (en cada primario) y $k_a \in [0,1]$ es el coeficiente de reflexión ambiente del material (también en cada primario).

Este modelo es también muy sencillo, aunque es una grosera simplificación de los fenómenos de iluminación ambiente. Produce escenas donde los objetos se ven antinaturalmente uniformes en brillo, pero sensibles a los cambios de iluminación ambiente. Por ejemplo, un objeto amarillo ($k_a =< 0.8, 0.8, 0 >$) se verá rojo si es iluminado con luz ambiente roja ($I_a =< 1, 0, 0 >$), amarillo si la luz es blanca ($I_a =< 1, 1, 1 >$), y *negro* si la luz es azul ($I_a =< 0, 0, 1 >$).

Uno de los primeros modelos de iluminación implementados fue el de Wylie, en 1967. El modelo sombreaba los puntos de la superficie de un objeto usando una intensidad (monocromática) inversamente proporcional a la distancia del punto a la fuente de luz. Superficies planas, entonces, reciben distintas intensidades. Otro desarrollo fue la consideración de la orientación de los puntos en la superficie con respecto a la fuente de luz. Bouknight [16] agregó en 1970 un término difuso Lambertiano (dependencia por el coseno del ángulo) para evaluar la intensidad de objetos construidos con polígonos o caras planas.

Consideremos ahora qué sucede cuando un objeto reflector difuso es iluminado con luz puntual. Esto es lo que sucede con objetos como la tiza, los cuales reflejan la luz incidente en todas las direcciones por igual. Por lo tanto, la intensidad luminosa reflejada depende exclusivamente de la intensidad luminosa incidente. Sea el punto p el lugar de incidencia, y sea N el versor normal al plano tangente de la superficie del objeto en p (ver Figura 7.2).

Para el punto p, la intensidad luminosa reflejada es un factor constante por la energía incidente. Esta última sólo depende del ángulo θ que forman la dirección L que va de p a la fuente de luz

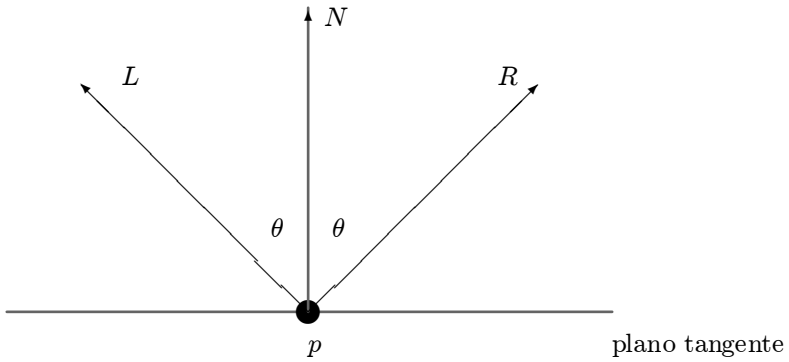


Figura 7.3 *Geometría de la reflexión especular.*

(llamado vector de incidencia) y el normal N a la superficie en p (ecuación de Lambert):

$$I = I_p k_d (\cos\theta),$$

donde I_p es la intensidad de la luz incidente en el punto p, k_d es el coeficiente de reflexión difusa para el material que constituye el objeto, y $\theta \in [-\frac{\pi}{2}..\frac{\pi}{2}]$ es el ángulo entre el vector normal y el vector de incidencia. Si $|N| = |L| = |V| = 1$, entonces es posible utilizar las identidades vectoriales (ver Apéndice 2) y expresar la ecuación de Lambert como

$$I = I_L k_d \ (N \cdot L).$$

Si el iluminante es lejano, entonces el vector de incidencia L puede considerarse constante, lo cual se denomina también "luz direccional".

También puede agregarse al modelo una función de atenuación similar a la propuesta por Wylie. En general el modelo de inversa al cuadrado de la distancia da variaciones muy abruptas, por lo que se utiliza una función

$$f_{att} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right),$$

donde las constantes c_1, c_2, c_3, determinadas por el usuario, simulan algún efecto atmosférico. La ecuación de iluminación es, entonces:

$$I = I_a k_a + f_{att} I_L k_d \ (N \cdot L).$$

Este modelo resultó ser poco satisfactorio, porque las caras "invisibles" desde la fuente de luz no reflejan energía luminosa, lo que le da una apariencia muy artificial a las escenas, Por ello, Bouknigth también incluyó un término ambiente para modelar la iluminación de las partes de un objeto que son visibles al observador pero invisibles para la fuente de luz.

Si el modelo de iluminación fuese el de un objeto especular puro, el 100% de la energía luminosa incidente se reflejaría en la dirección R (ver Figura 7.3). En cambio, si el objeto es reflector Lambertiano puro, el coseno del ángulo θ entre el normal N y la dirección V hacia la cual se encuentra el observador no se aplica para la energía reflejada. Es decir, la *intensidad* luminosa de un reflector Lambertiano puro, vista desde cualquier dirección, es constante. Esto se debe a que, si bien la *energía* luminosa radiada hacia el observador depende de la ubicación del observador,

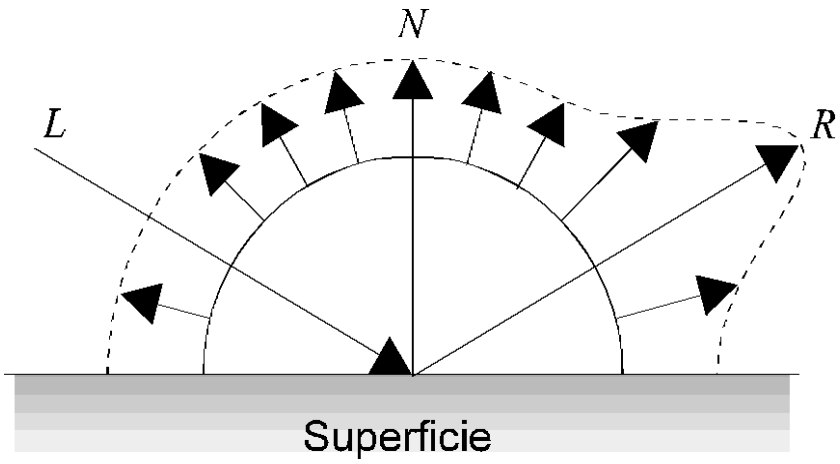


Figura 7.4 *Una superficie normalmente refleja una combinación difusa y especular.*

también el ángulo sólido subtendido por un diferencial de área alrededor del observador depende de dicha dirección. Como la energía es la integral de la intensidad por unidad de área, al variar la dirección del observador, varía la energía pero también varía el área en forma idéntica, con lo cual ambos factores se cancelan y la intensidad permanece constante.

En la práctica es fácil ver que ningún reflector es Lambertiano puro. Los objetos tienden a reflejar mayor intensidad hacia el ángulo de la reflexión especular predicha por la ley de Snell (ver Figura 7.4). Phong Bui-Tuong propuso un modelo de reflexión en el cual es posible separar una componente difusa (Lambertiana) de una especular, cada una con su coeficiente específico. De esa manera, es posible determinar una ecuación de iluminación en la cual interviene también el ángulo que forman el normal y la dirección del observador.

El modelo de Phong es el que más se usa en Computación Gráfica. En él la reflectividad de una superficie está dada por la contribución de dos factores. El primero es esencialmente un factor Lambertiano, que depende de un coeficiente definido para el material, y del coseno del ángulo θ de incidencia. El segundo es un factor que tiene en cuenta la reflexión especular, y depende de un coeficiente del material y del coseno del ángulo α entre la dirección de reflexión ideal R y la dirección V al observador.

Pero la observación crucial de Phong es que en la reflexión especular, el "brillo" que se produce aparece del mismo color que la luz (con excepción de los metales), y en general no depende del color de la superficie. Por lo tanto nuestras ecuaciones deben ahora tener en cuenta la distribución espectral (el color) del iluminante y del objeto por separado. Designaremos, entonces, con I_λ a la intensidad resultante del modelo a cierta longitud de onda λ, (que normalmente es alguno de los primarios R, G, B), $I_{a\lambda}$ es la intensidad de la iluminación ambiente a dicha longitud de onda, y $I_{p\lambda}$ es la intensidad del iluminante puntual. También designaremos con O_λ al color del objeto, de modo que $k_a O_\lambda$ corresponde al coeficiente de reflexión ambiente a una determinada longitud de onda.

Otro problema es que la ecuación de la refexión especular predicha por la ley de Snell implica que toda la energía se refleja especularmente en una única dirección (ver Figura 7.3):

$$R = 2N(N \cdot L) - L.$$

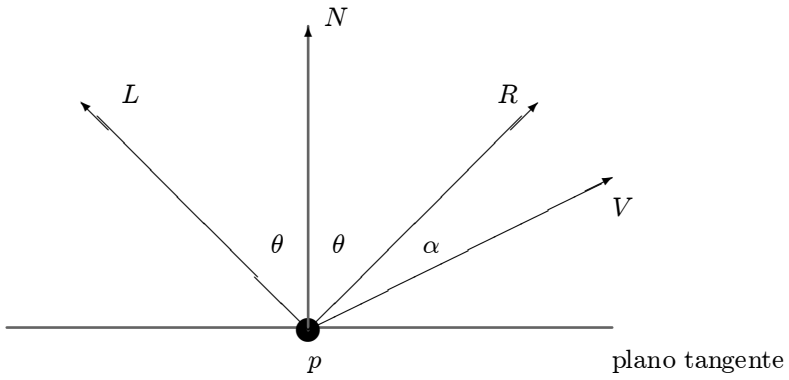


Figura 7.5 *El ángulo entre el rayo reflejado ideal y la dirección del observador.*

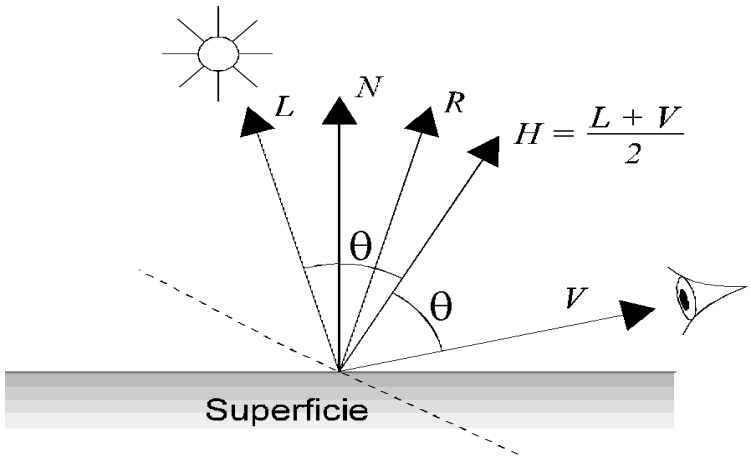


Figura 7.6 *Interpretación geométrica del vector H.*

Phong propone entonces una función empírica para determinar en forma efectiva y sencilla la distribución geométrica de la energía luminosa reflejada en forma especular. Dicha función parte de considerar el ángulo α entre el rayo reflejado ideal R y la dirección del observador V (ver Figura 7.5).

Utilizar el coseno de dicho ángulo determina funciones de reflexión especular que "decaen" lentamente a medida que la dirección de observación se aparta de la del rayo reflejado. Esto es característico de superficies poco pulidas, como el papel común, la madera, y algunas telas. Pero en superficies más pulidas, dicho decaimiento es más abrupto, o dicho de otra forma, la influencia de la reflexión especular es intensa pero muy localizada. Por dicha razón, la función de *glossiness* o pulimiento propuesta por Phong es $\cos^n \alpha$, donde n puede ser un entero entre 1 y 100:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} \left[k_d O_{d\lambda} \cos\theta + W(\theta) \cos^n \alpha \right],$$

donde

- $I_{a\lambda}k_aO_{d\lambda}$: Iluminación ambiente.
- $f_{att}I_{p\lambda}k_dO_{d\lambda}\cos\theta$: Término de reflexión difusa.
- $f_{att}I_{p\lambda}W(\theta)\cos^n\alpha$: Término de Phong (iluminación especular no ideal).
- n: factor de *glossiness*.

Como antes, si L, V, R y N están normalizados, entonces $\cos\theta = (N \cdot L)$ y $\cos\alpha = (R \cdot V)$. $W(\theta)$ es utilizada para modelar funciones de reflectividad que varían con el ángulo de incidencia, aunque normalmente una constante k_s (coeficiente de reflexión especular) $k_s \in [0..1]$ da resultados aceptables.

De esa manera, la ecuación de iluminación de Phong, en función de los coeficientes (cromáticos) de reflexión ambiente, difusa y especular, y de la intensidad (cromática) de la iluminación ambiente y puntual, es

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}[k_dO_{d\lambda}\ (N \cdot L) + k_s(R \cdot V)^n] \tag{7.1}$$

Es importante observar que k_s no depende de la longitud de onda, haciendo que el color del brillo sea el mismo que el de la fuente de luz (dado por $I_{p\lambda}$). Además, según se propuso, el exponente n controla el "brillo" del material. Sin embargo, el efecto que resulta al variar n es simular que la fuente de luz varía su tamaño. Reduciendo n hace que la fuente luminosa parezca mayor; pero **no** reduce el brillo del objeto (ver Figura 7.8 [84]).

Blinn [8] propone una manera de simplificar el costo computacional de la ecuación 7.1. Básicamente propone un vector H el cual está a mitad de camino entre R y V (ver Figura 7.6):

$$H = \frac{(L+V)}{|L+V|}.$$

De esa manera, la componente especular de la ecuación de Phong es

$$(N \cdot H)^n.$$

Como el cálculo de dicha potencia no debe realizarse necesariamente con gran precisión, es posible aproximarlo satisfactoriamente con tablas o funciones polinomiales de bajo orden. De esa manera, el modelo de Phong y su ecuación 7.1, si bien parte de una base empírica, logra resultados bastante adecuados en tiempos razonables.

En la Figura 7.7 podemos observar los diferentes efectos que produce, para un iluminante y un objeto dado, el cambio de los coeficientes de iluminación ambiente y reflexión difusa, para un factor de glossiness y un factor de reflexión especular fijos. En la Figura 7.8 podemos observar los diferentes efectos que produce en el mismo objeto el cambio de los coeficientes de reflexión especular y factor de glossiness, para factores de reflexión difusa e iluminación ambiente fijos. Una última mejora que mencionamos al modelo de Phong proviene de considerar que las fuentes luminosas pueden no radiar energía uniformemente en todas las direcciones. Para ello puede elevarse el coseno del ángulo *de incidencia* a una cierta potencia, simulando luces reflectoras, o bien utilizar una función umbral para simular haces muy concentrados y *spots*.

La ecuación de Phong es también adecuada para modelar varios iluminantes, debiendo computarse separadamente la contribución especular y difusa de cada uno:

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + \sum_{i=luces} f^i_{att}I^i_{p\lambda}\left[k_dO_{d\lambda}\ (N \cdot L^i) + W(\theta)\ (R^i \cdot V)^n\right].$$

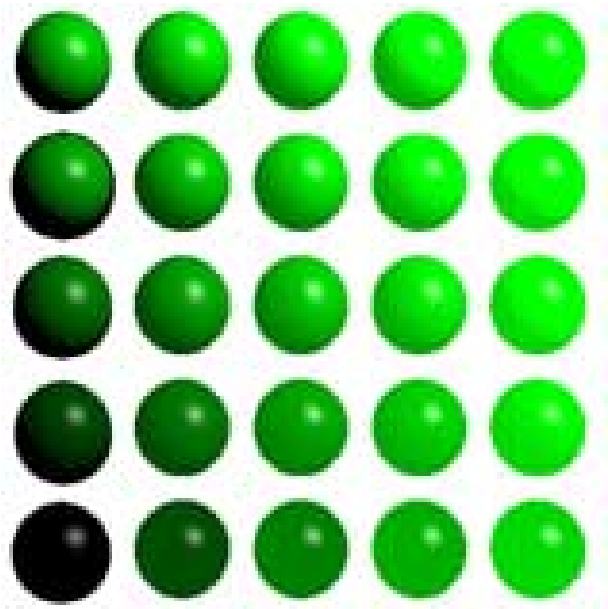


Figura 7.7 *Modelo de iluminación de Phong. Se varía de izquierda a derecha el coeficiente de iluminación ambiente de 0 a 1, y de arriba abajo el coeficiente de reflexión difusa de 0 a 1.*

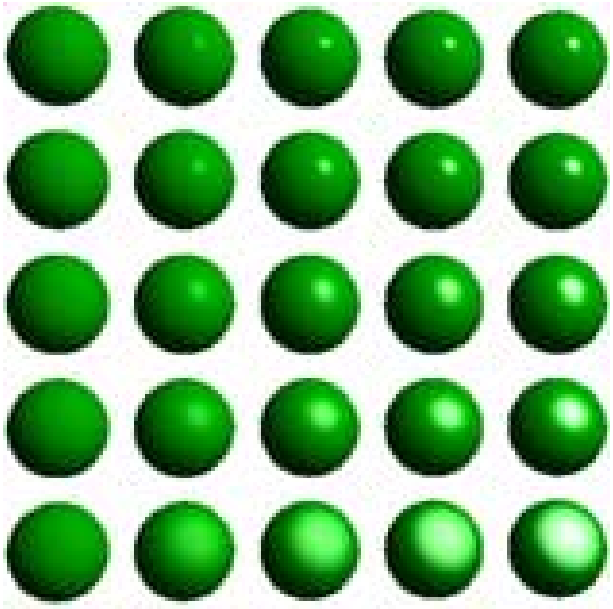


Figura 7.8 *Modelo de iluminación de Phong. Se varía de izquierda a derecha el coeficiente de reflexión especular de 0 a 1, y de arriba abajo el coeficiente de glossiness de 1 a 50.*

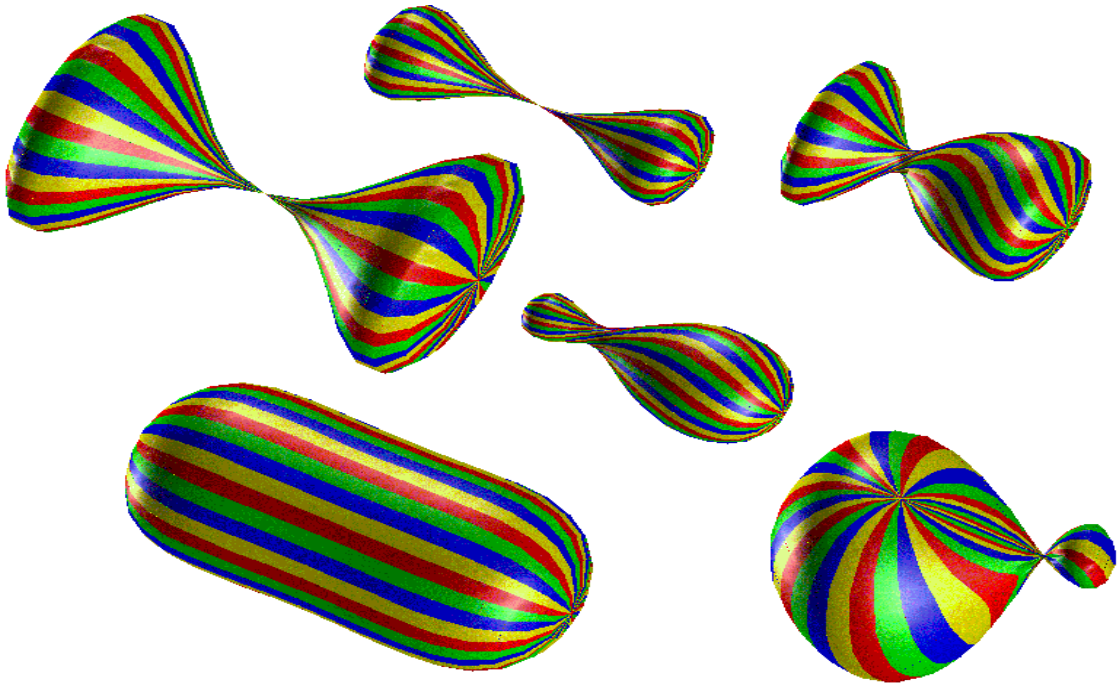


Figura 7.9 *Modelo de iluminación de Phong con iluminantes múltiples.*

Podemos ver en la Figura 7.9 el resultado de computar un objeto con varias fuentes de luz. Sin embargo, la interreflexión no puede determinarse, por lo cual el modelo sigue siendo *local*.

7.3 Sombreado de Polígonos

Si bien los resultados de la Sección anterior pueden ser satisfactorios, en la práctica se demuestra que no es necesario tomar una muestra del modelo de iluminación local para cada pixel de la escena. Normalmente este cómputo es redundante y muy ineficiente. Por lo tanto, en esta Sección estudiaremos algunas técnicas que permiten acelerar el dibujado de escenas con realismo por medio de técnicas de *sombreado*, es decir, por medio de interpolaciones entre los valores de las muestras.

La manera más rápida de pintar un polígono, es por medio de la técnica de "sombreado constante". El modelo de iluminación es utilizado para calcular una única intensidad, con la cual se "pinta" todo el polígono. Si la fuente luz está muy alejada, entonces $N \cdot L$ es constante en toda la cara del polígono. Si además el observador está alejado, entonces también $N \cdot V$ es constante. Por lo tanto, muestreamos la ecuación para todo el polígono: (en el centro o en un vértice). La intensidad resultante es utilizada durante la conversión scan del polígono para setear cada uno de sus pixels (ver Figura 7.11). El algoritmo para graficar una escena con este método puede verse en la Figura 7.10.

Si alguna de las dos suposiciones deja de ser verdadera, es decir, o bien la fuente de luz o el observador está próximo al polígono, entonces el esquema comienza a ser más y más artificial. El color de cada polígono dependerá de dónde se efectúe la muestra, dado que con un mismo normal,

```
procedure sombreado_constante(e:escena);
...
begin
 para cada objeto o en e
   para cada cara de o
     computar un normal n
     computar el modelo de iluminacion para n
     efectuar la conversion-scan de o
end
```

Figura 7.10 *Algoritmo de sombreado constante de polígonos.*

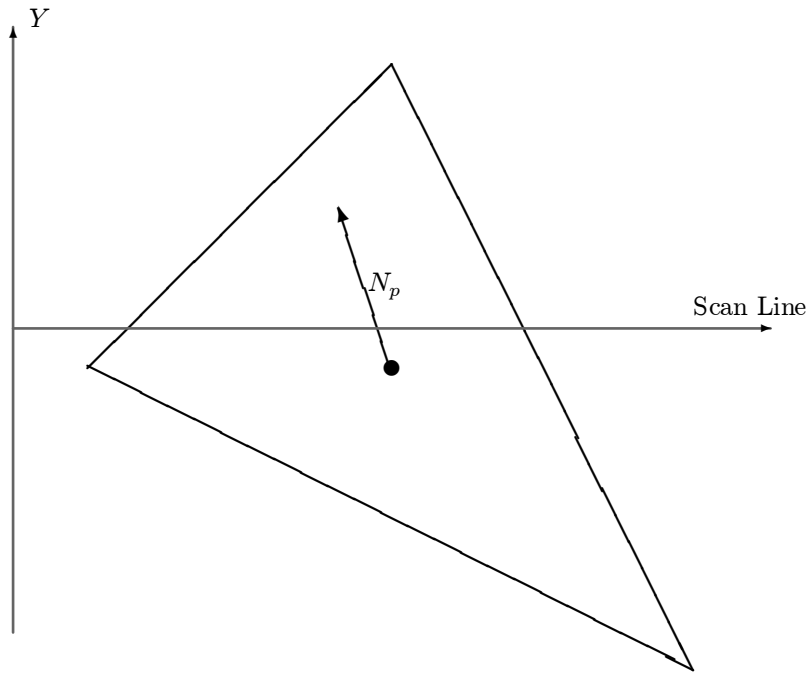


Figura 7.11 *Sombreado constante de polígonos.*

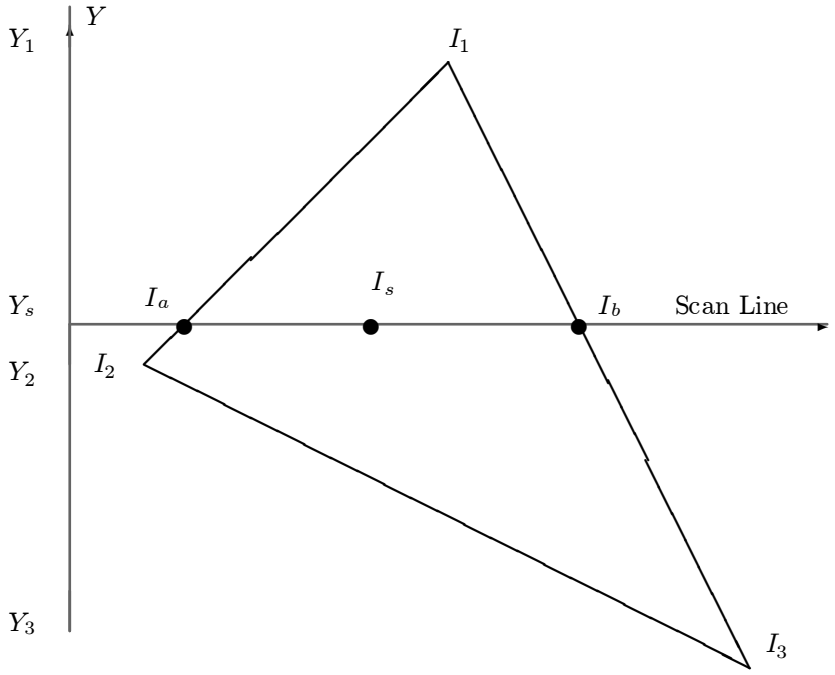


Figura 7.12 *Técnica de sombreado de Gouraud.*

en distintos puntos de la cara el modelo de iluminación arrojará colores diferentes. Puede tomarse una muestra en cada vértice, y pintar cada polígono con el promedio de los colores de los vértices. Sin embargo, se seguirá obteniendo el desagradable facetado resultante.

La primer técnica de sombreado suave, que soluciona este problema sin aumentar la cantidad de muestras, se debe a Gouraud [29]. Como en el sombreado constante, se calcula el modelo de iluminación en cada vértice (es decir, en promedio una evaluación por polígono). Durante la conversión scan del polígono, se interpola el valor de iluminación a lo largo de las aristas. Por ejemplo, dado un triángulo con vértices $1, 2, 3$, se computa el modelo de iluminación en dichos vértices, obteniéndose I_1, I_2, I_3 (ver Figura 7.12). Para una linea de scan, interpolando entre I_1 e I_2 podemos obtener I_a, y entre I_1 e I_3 podemos obtener I_b. Luego, para todo pixel en la línea de barrido, se interpola nuevamente entre I_a e I_b para obtener el valor de I_s.

$$I_a = I_1 - (I_1 - I_2)\frac{Y_1 - Y_a}{Y_1 - Y_2},$$
$$I_b = I_1 - (I_1 - I_3)\frac{Y_1 - Y_a}{Y_1 - Y_3},$$
$$I_s = I_b - (I_b - I_a)\frac{Y_b - Y_s}{Y_b - Y_a}.$$

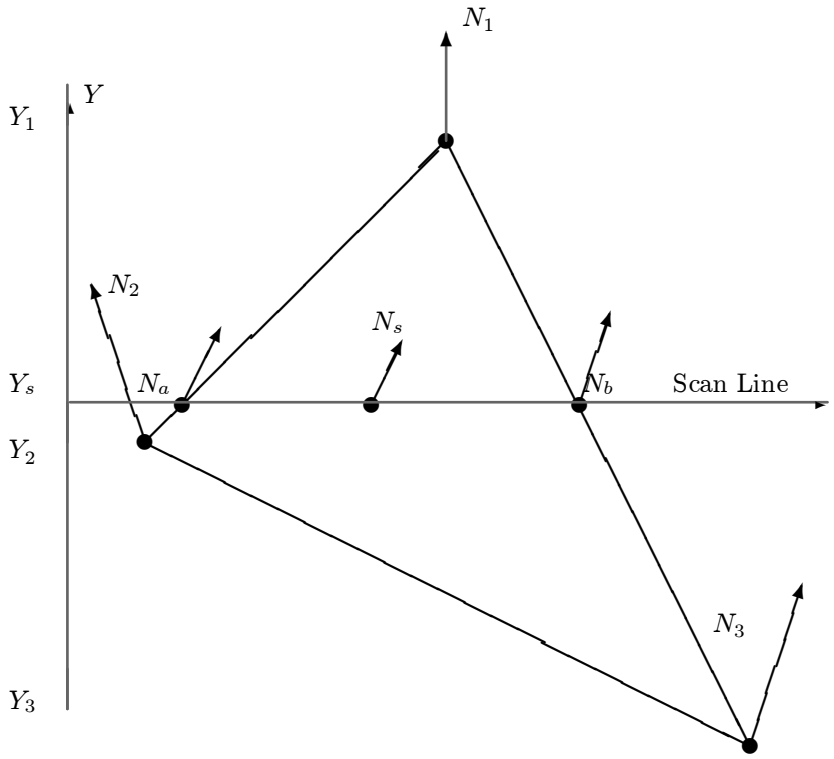


Figura 7.13 *Sombreado de Phong.*

Normalmente los normales en los vértices están mal definidos, porque dependerán con respecto a qué cara se los considere. Sin embargo, es posible promediar los normales definidos según cada cara, lo cual dará una medida consistente. Si los polígonos fueron construidos de manera disciplinada, tal cual se sugiere en la Sección 6.4, entonces encontrar los normales es bastante directo. Es muy importante destacar que todas estas operaciones que se realizan durante la conversión scan del polígono, pueden efectuarse con técnicas aritméticas que posibilitan su implementación por hardware. Tal es así que en la actualidad las PC poseen tarjetas gráficas que reciben directamente los puntos e iluminaciones de los polígonos, y efectúan la conversión scan y sombreado de Gouraud, *miles* de polígonos por segundo!

Esta técnica realiza una aproximación muy grande, y por lo tanto puede tener resultados insatisfactorios. El ejemplo característico es el sombreado de un polígono iluminado por una fuente de luz cercana, ubicada sobre su centro. La evaluación de la iluminacióna en cada vértice será aproximadamente similar, por lo que todo el polígono será pintado de un color constante! Una solución para este problema y para varios otros consiste en subdividir los polígonos. Practicamente en todos los métodos derivados del scan-line, la solución por "fuerza bruta" es aumentar la cantidad de polígonos.

Consistentemente con la técnica de Gouraud, Phong elaboró otra técnica denominada *sombreado de Phong* (que no debe confundirse con el modelo de iluminación de Phong). La técnica consiste en interpolar los normales en cada punto de la conversión scan del triángulo, a partir de los normales en cada vértice del mismo. Esto es geométricamente mucho más preciso, y restaura

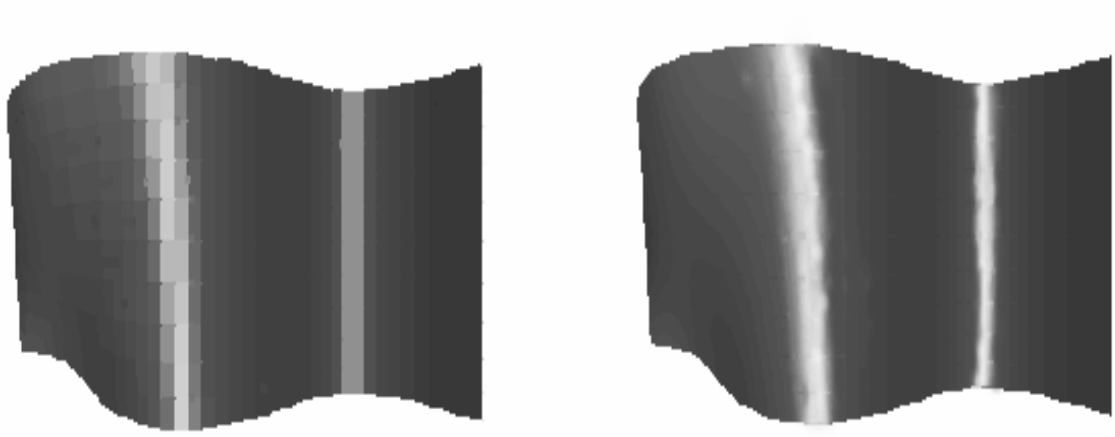


Figura 7.14 *Sombreado constante y sombreado de Phong.*

la curvatura a las caras. En el mismo, la interpolación se realiza durante la conversión scan, como en Gouraud, primero entre vértices, y luego entre lineas de barrido (ver Figura 7.13):

$$N_a = N_1 - (N_1 - N_2)\frac{Y_1 - Y_a}{Y_1 - Y_2},$$
$$N_b = N_1 - (N_1 - N_3)\frac{Y_1 - Y_a}{Y_1 - Y_3},$$
$$N_s = N_b - (N_b - N_a)\frac{Y_b - Y_s}{Y_b - Y_a}.$$

El sombreado de Phong es más preciso que el de Gouraud, reproduce mejor la reflexión especular, inclusive la interior a un polígono, y restaura la curvatura en los objetos poligonizados. Sin embargo, es más costoso computacionalmente que el de Gouraud, dado que para cada pixel debe calcularse el modelo de iluminación. Esto, además, hace que no sea posible su implementaciaón por hardware, al menos por ahora.

Todas las técnicas de sombreado que vimos tienen problemas en general. Básicamente, los objetos aparecen curvados, pero sus siluetas siguen siendo poligonales. También es necesario tener en cuenta que el recorrido de un polígono por scan-line no representa una recorrida uniforme en el espacio del objeto, debido al efecto de la perspectiva. De esa manera, un sombreado con interpolación lineal como Gouraud o Phong produce una distorsión en el efecto de la perspectiva. Esto también sucede si el objeto es rotado. Es decir, en general la interpolación no es invariante frente a transformaciones.

También pueden ocurrir problemas geométricos diversos. Por ejemplo, si los polígonos no conforman una red regular, entonces pueden existir polígonos vecinos que no compartan un vértice. De esa manera, el valor encontrado para un punto vecino a dicho vértice puede ser muy diferente al del vértice mismo. Por último, promediar los normales para encontrar los normales en los vértices puede producir una uniformidad artificial.

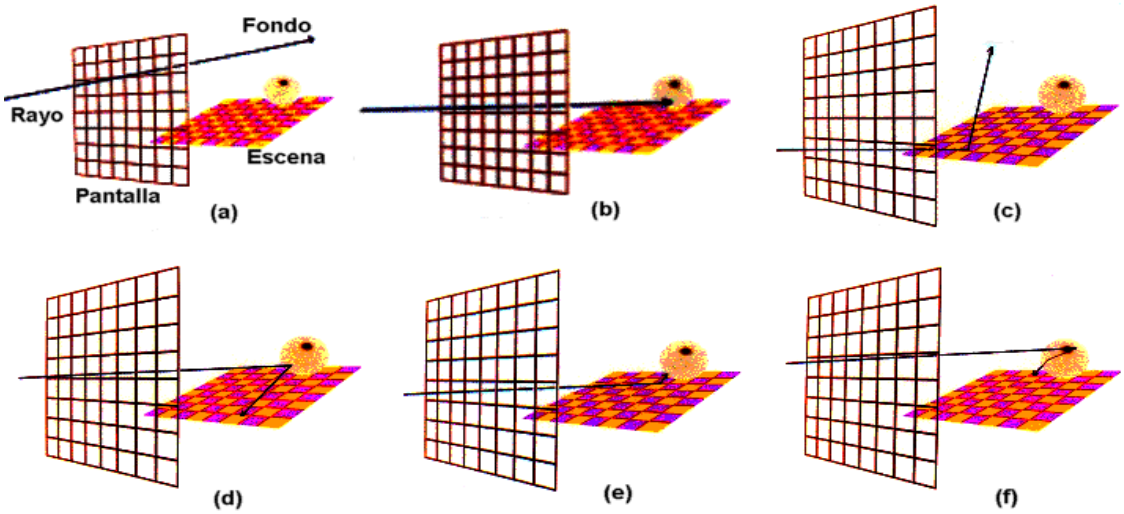


Figura 7.15 *Distintas posibilidades en el cómputo de un rayo.*

7.4 Ray Tracing

En los modelos de iluminación presentados en las Secciones anteriores se llegó paulatinamente a una ecuación que contempla en forma físicamente adecuada la reflexión difusa (Lambertiana) y la reflexión especular de iluminación proveniente de fuentes de luz puntual. Sin embargo, la interreflexión de luz entre objetos de la escena quedó representada en un término de iluminación ambiente, único para todos los objetos de la escena, lo cual es indudablemente insatisfactorio para expresar un modelo de iluminación global. Fue en 1980 cuando Whitted [85] propuso un modelo de iluminación recursivo que integra reflexión, refracción, cara oculta y sombras en un único procedimiento.

El modelo más simple consiste en trazar rayos (de ahí su nombre) a través de los caminos de reflexión y refracción entre los objetos de la escena. Cada rayo se traza desde el observador, pasando a través de un pixel de la pantalla, hasta encontrar el primer objeto de la escena. Si el rayo no intersecta ningún objeto (Figura 7.15(a)), entonces se pinta el pixel de un color de fondo. Cuando intersecta un objeto (Figura 7.15(b)), se plantea un modelo de iluminación local (por ejemplo, Phong) para determinar el color, pero se computa además un rayo hacia la fuente de luz (Figura 7.15(c)), para garantizar que otros objetos de la escena no estén "haciendo sombra" sobre el punto de la escena que estamos iluminando.

Si los objetos de la escena tienen, además, un coeficiente de reflexión especular o de refracción, entonces es necesario computar un factor proveniente de los rayos idealmente reflejados y refractados. La intensidad de cada uno de dichos rayos es computada recursivamente, para lo cual se envían nuevos rayos que experimentan el mismo procedimiento (Figura 7.15(d), (e) y (f)), lo cual configura un árbol de rayos, cada uno de los cuales ejerce una influencia en el color del pixel (ver Figura 7.16).

El costo computacional del trazado de rayos es muy alto, porque para cada pixel de la pantalla es necesario elaborar todo un árbol completo de iluminación. Es posible disminuir el costo limitando la profundidad de la recursión. Por ejemplo, en la Figura 7.18 podemos ver una escena compuesta

Figura 7.16 *Árbol de ray tracing.*

por tres esferas rodeadas por tres espejos. Trazando los árboles de ray tracing con profundidad 1 se computa solamente el modelo de iluminación local, por lo que los espejos aparecen como superficies oscuras y pulidas. Con profundidad 2, aparecen las reflexiones directas de las esferas en los espejos del costado, y con profundidad 3 aparecen las reflexiones (indirectas) de dichas reflexiones en el espejo de abajo.

Los rayos se consideran infinitesimalmente delgados, y la reflexión y refracción son ideales, sin dispersión, como si se tratara de superficies idealmente suaves. Por dicha razón, las imágenes sintetizadas con ray tracing tienen una característica especial, siempre se trata de escenas con objetos geométricamente simples y brillantes que exhiben reflexiones múltiples de una agudeza levemente sobrenatural (ver Figura 7.19).

Un prototipo de trazador de rayos recursivos sigue aproximadamente los siguientes pasos. Dado un punto `p` de donde parte el rayo, en una dirección `d`, y una profundidad de recursión `r`, se deben encontrar todos los objetos de la escena que intersectan al rayo, calcular el punto de intersección en cada caso, y elegir el más cercano de los puntos de intersección. En dicho punto se encuentra un modelo de iluminación local, y se calculan las direcciones del rayo reflejado y del rayo refractado. Luego se computa recursivamente un ray tracing sobre dichos rayos (con un nivel menos de recursión), y con el color resultante de los dos rayos más el color resultante del modelo de iluminación local se computa el color final del pixel (ver Figura 7.17).

Pese al costo computacional, es una característica muy atractiva que se combinen tantos efectos de iluminación en un único algoritmo. Por dicha razón es que el tema se ha desarrollado de una manera impresionante en los últimos 10 años. En particular, podemos mencionar que se han estudiado una gran cantidad de métodos para acelerar la ejecución del ray tracing, o para permitir una representación más versátil de objetos.

El control adaptativo (dependiente de la escena) del nivel de recursión es la optimización más directa. Hall y Greenberg [47] señalaron que el porcentaje de una escena que contiene superficies muy transparentes y reflexivas es, en general, pequeño y es por lo tanto ineficiente llegar con cada rayo hasta la profundidad máxima. Sugieren entonces usar un control de profundidad adaptativo

```
procedure trazarrayo(p:punto;d:vector;r:integer;var col:color);
...
begin
 if d < MAXRECURS then begin              {no llego al limite de recursion}
   r:=r-1;                                {un nivel menos de recursion}
   intersecta(p,d,objeto,intersec);       {primer objeto que intersecta}
   if objeto=nil then col:= COLORFONDO;   {no intersecta ningun objeto}
    else begin
     phong(objeto,intersec,collocal);        {ilum. local del objeto}
     reflejado(p,d,objeto,intersec,refl);    {direc. reflejado}
     refractado(p,d,objeto,intersec,refr);   {direc. refractado}
     trazarrayo(intersec,refl,r,colrefl);    {color rayo reflejado}
     trazarrayo(intersec,refl,r,colrefr);    {color rayo refractado}
     colorglobal(objeto,collocal,colrefl,colrefr,col);  {modelo hibrido}
    end;
 else col.r:=col.g:=col.b:=0              {supero recursion -> color negro}
 end;
end;
```

Figura 7.17 *Prototipo de un trazador de rayos.*

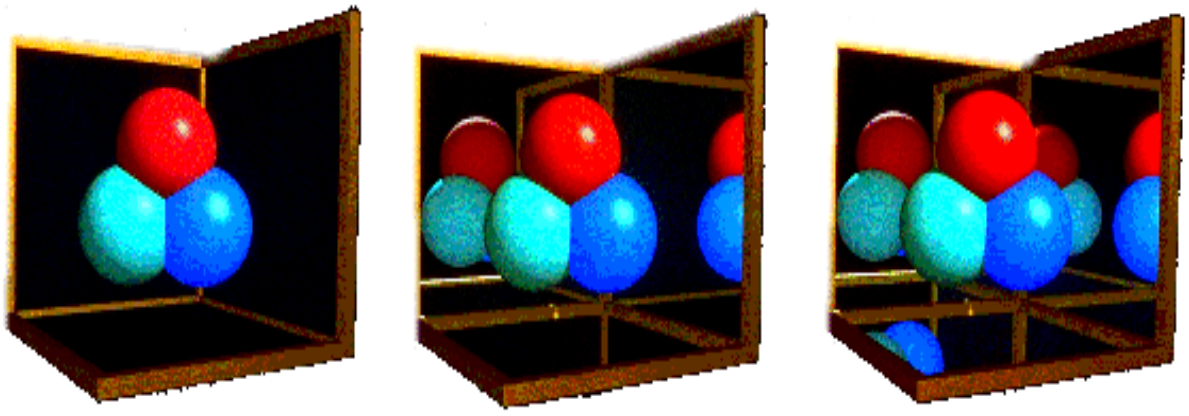


Figura 7.18 *Efecto de modificar la profundidad de la recursión.*

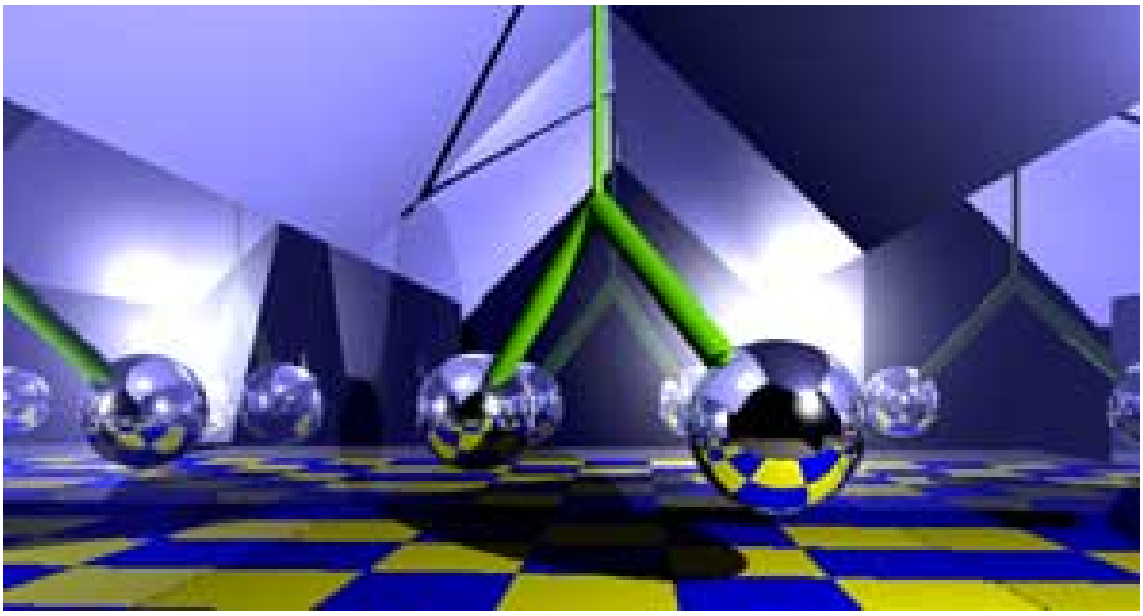


Figura 7.19 *Una escena compleja elaborada con ray tracing.*

que dependa de las propiedades de los materiales con los cuales intersectan los rayos, hasta una profundidad máxima prefijada. Cuando un rayo se refleja en una superficie se atenúa de acuerdo al coeficiente de reflexión especular de ésta. De la misma forma, cuando es refractado es atenuado por el coeficiente de transmisión global de la superficie. La atenuación final del rayo será el producto de las atenuaciones en cada intersección. Si este valor cae por debajo de un cierto nivel no tendrá sentido seguir propagando el rayo.

Otra estrategia es reducir el número de objetos con los que debe intentarse la intersección con un rayo. De esa manera, cada objeto (especialmente los muy complejos) se consideran encerrados por un volumen sencillo, por ejemplo un cubo, con el cual se chequea la intersección. Para un objeto representado por un gran número de polígonos esto evita el cálculo de intersección con cada uno de ellos. Las esferas han sido usadas casi siempre como volúmenes debido a la simplicidad de cálculo de intersecciones entre esferas y rayos. Sin embargo su conveniencia depende mucho de la naturaleza del objeto. Por ejemplo, para objetos largos, angostos y curvados resultará en una situación donde la mayor parte de los rayos que intersectan la esfera no intersectan el objeto.

Otra estrategia es el uso de coherencia espacial. El espacio ocupado por la escena se subdivide en regiones. En lugar de intersectar el rayo con todos los objetos o conjuntos de ellos, determinamos qué regiones del espacio atraviesa el rayo y consideramos solamente los objetos que ocupan esas regiones. Esta aproximación ha sido desarrollada independientemente por muchos autores. Todas las aproximaciones involucran un preprocesamiento del espacio para preparar estructuras de datos auxiliares. Entre las soluciones que se basan en la coherencia espacial encontramos:

- Dividir el espacio en una grilla fija de regiones iguales. La dificultad consiste en determinar el tamaño de las regiones y guardar la gran estructura resultante.
- Dividir el espacio en forma binaria, método conocido como BSP (Binary Space Partitioning).

- Dividir el espacio en octrees. El *octree* es una estructura de datos representada por un árbol en el que cada nodo, llamado *voxel*, tiene 8 hijos, asociados a particiones de un cubo por tres planos ortogonales que pasan por su centro. La profundidad de una rama del árbol puede variar.

También se puede considerar el trazado de rayos de un grosor dado o de un conjunto de rayos paralelos, porque en la mayoría de las escenas, muchos grupos de rayos adyacentes siguen prácticamente las mismas trayectorias. El algoritmo descripto en [50] se denomina *Beam Tracing* y está basado en este tipo de coherencia. Es indicado para escenas que solo posean superficies poligonales, ya que en ellas las reflexiones son lineales. Es esencialmente un algoritmo de superficie oculta recursivo que encuentra todos los polígonos visibles dentro de una pirámide arbitraria. Para cada pirámide de rayos, comenzando por el plano de proyección de la imagen completa, se crea una lista ordenada por profundidad de los objetos de la escena. Estos objetos se prueban uno a uno con la pirámide. En el caso que la intersecten, la pirámide se divide en la porción que hizo la intersección y la que no intersectó, que seguirá su recorrido contra los objetos restantes. Se simula reflexión y refracción lanzando recursivamente nuevas pirámides, que son deformaciones de la original por medio de transformaciones lineales.

Podemos mencionar, por último, que en la graficación de la escena es posible encontrar zonas completas de la pantalla en la cual los pixels adyacentes están fuertemente correlacionados, tienen intensidades similares e historias muy semejantes de intersección con objetos. En [1] se aprovecha esta correlación al elaborar un algoritmo que, en regiones con pocas variaciones de iluminación y con la misma historia de rayos, reduce el tiempo de cálculo lanzando solamente algunos rayos e interpolando el resto. Esta técnica se conoce como sub-muestreo. El algoritmo procede recursivamente de una manera similar al de Warnock. Se empieza calculando las intensidades en una malla de pixels separados entre sí. Luego los clasifica según la diferencia de intensidad entre rayos vecinos y sus historias de rebotes en la escena. Los pixels intermedios se calculan por interpolación.

7.5 Radiosidad

Así como el ray tracing parte del modelo de iluminación local de Phong y busca mejorar la interreflexión especular, sombreado y refracción, el método de radiosidad busca una solución global a los términos de iluminación e interreflexión difusa. La creación de imágenes con reflexión difusa requiere el cálculo de la iluminación de todas las superficies para un conjunto de posiciones y direcciones discretas del entorno que están determinadas por criterios independientes de la ubicación del observador. La discretización del entorno tridimensional (las superficies a visualizar) reduce en gran medida el número de puntos muestreados independientemente de la resolución de la imagen final. Para superficies con alto gradiente de intensidad, en los bordes de las sombras por ejemplo, no es suficiente una discretización uniforme, por lo que se usa una discretización que se ajusta adaptativamente por el gradiente de intensidad [21].

Ray tracing selecciona aspectos particulares de la interacción luz-objeto: reflexión especular y transmisión; aproxima estos y excluye otras consideraciones. Radiosidad, toma en algún sentido el punto de vista opuesto, favorece la interacción de superficies con reflexión difusa y excluye la reflexión especular. Más aún, el método tradicional de radiosidad asume comportamiento difuso ideal en todas las superficies y expresa las transferencias entre superficies por medio de un factor de forma [43]. El algoritmo de radiosidad convencional comprende las siguientes fases:

1. Computar los factores de forma: determinando los polígonos visibles (o porciones de ellos) desde cada otro polígono, usando algún artefacto auxiliar como por ejemplo un semi-cubo.

2. Resolver la ecuación de la matriz de radiosidad usando algún método numérico (por ejemplo por Gauss-Seidel). Como la matriz es diagonal dominante converge rápidamente. Se hace para cada banda de color por separado.

3. Mostrar los resultados: se determinan las superficies ocultas y se interpolan los valores de radiosidad. Para ello se puede usar z-buffer y sombreado de Gouraud respectivamente.

Las imágenes realizadas con radiosidad exhiben un tratamiento adecuado para problemas geométricos enormemente complejos, como las sombras suaves (como las que ocurren con la luz difusa), y la interreflexión difusa y el trasvasamiento de color. Estos resultados son notablemente verosímiles, no por el nivel de detalle (como en ray tracing) sino por la calidez y armonía que exhiben. Lamentablemente, radiosidad no permite un tratamiento adecuado de la reflexión especular, y hasta la fecha no se ha podido encontrar un método que combine las ventajas de ambos.

7.6 Comparación de los métodos de rendering

Podemos dedicar en este punto unos párrafos a comparar los tres grandes métodos de rendering vistos hasta ahora: scan-line, ray tracing y radiosidad.

Velocidad: Los métodos scan-line tienen siempre la ventaja con respecto a la velocidad de ejecución. El tiempo de ejecución de ray tracing puede acelerarse con alguna de las técnicas vistas en Secciones anteriores, pero siempre será un orden de magnitud mayor que scan-line. Los tiempos de radiosidad son aún más lentos, dado que el cómputo de los factores de forma y la subdivisión adaptativa involucra algoritmos de complejidad potencialmente exponencial.

Resultados: Tanto radiosidad como ray tracing producen resultados superiores a scan-line. El modelo de iluminación de radiosidad es físicamente correcto, y por lo tanto las imágenes son las más verosímiles realizadas hasta la fecha. La falta de iluminación puntual y reflexión especular es una limitación. Ray tracing, por su parte, produce resultados que impresionan pero no son siempre creíbles. El modelo de iluminación es incorrecto porque considera separadamente la componente local y la recursiva.

Algoritmos: Los algoritmos para ray tracing son sin duda los más sencillos y adaptados a una metodología de desarrollo. Scan line utiliza un modelo conceptual uniforme (la tubería de procesos), pero para determinados problemas (por ejemplo, cara oculta) este modelo se interrumpe y es necesario intercambiar información entre distintos bloques del programa, con lo cual las implementaciones se van tornando más y más desordenadas. Radiosidad necesita recurrir a algoritmos para computar el factor de forma y subdividir adaptativamente la escena, los cuales son enormemente complejos y problemáticos.

Primitivas geométricas: En scan-line es posible representar objetos de una forma arbitraria, o bien se puede trabajar con poligonizaciones. En ray tracing es indispensable contar con objetos cuya intersección con rayos sea fácilmente computable. Poligonizar objetos puede tener un costo prohibitvo porque incrementa enormemente la cantidad de tests de intersección. Radiosidad debe trabajar con poligonizaciones, por lo que en principio tampoco está limitado en la geometría de los objetos.

Aliasing: Es un problema derivado del muestreo de la escena. En scan-line, un objeto que al proyectarse se vuelve pequeño o delgado, siempre es dibujado porque su discretización encuentra por lo menos un pixel. En cambio en ray tracing puede suceder que un objeto (o parte de un objeto) pequeño se encuentre entre dos rayos trazados y por lo tanto no se dibuje.

7.7 Mapas de atributos

La síntesis de imágenes con realismo ha seguido en este Capítulo el camino de agregar a un modelo sencillo y bien conocido (aunque de base empírica) los elementos de mayor complejidad que permiten simular mejor los efectos físicos en un modelo de iluminación. De esa manera partimos de los modelos de iluminación locales, para mejorar con ray tracing la simulación de reflexiones especulares, y con Radiosidad la simulación de iluminación y reflexiones difusas. Pero otro camino para mejorar el realismo es dotar a las superficies de los objetos de propiedades y atributos que los vuelvan verosímiles, dado que hasta ahora solamente se trataban de superficies lisas, de un determinado color y rugosidad.

Las técnicas de mapeo de texturas en general constituyen una de las formas más utilizadas y de mejores resultados para realzar las cualidades de los objetos representados. De esa manera, se obtiene mayor realismo en las escenas con un costo adicional relativamente aceptable. La idea básica del mapeo de atributos consiste en mapear o proyectar un patrón bidimensional para un atributo dado sobre la superficie de un objeto. Por ejemplo, podemos tener almacenada la textura de una madera como una perturbación local de los colores (coeficientes O_d en el modelo de Phong en la ecuación 7.1). Al graficar un polígono cuya superficie debe parecer de madera, se recorre dicho patrón utilizando alguna función asociada a las coordenadas generadas durante la conversión scan del polígono, y se utiliza el coeficiente en dicha coordenada como perturbación local en el color de la superficie.

En general, los atributos que pueden mapearse a una superficie pueden ser varios:

Mapa de Texturas: Como ya describimos, consiste en alterar localmente el color de la superficie modificando los coeficientes en el modelo de reflexión difusa del objeto.

Mapa de Entorno: Consiste en almacenar en un buffer especial los objetos de la escena tal como son visibles desde la superficie del objeto que se está considerando. Dicho buffer es luego utilizado como coeficiente de iluminación especular en superficies lisas, dando la impresión de que la escena "se refleja" en ellas. Si bien el planteo es geométricamente incorrecto [7], los resultados que produce son satisfactorios, y es actualmente una técnica muy popular.

Mapa de Desplazamientos: También denominada *bump mapping*. Se perturba *la posición* local de los puntos del polígono en dirección de la normal, según una distancia especificada en el patrón bidimensional. De esa manera es posible darle a un objeto una apariencia arrugada o abollada (ver Figura 5.17).

Mapa de Normales: Dado que el modelo de iluminación de Phong es muy sensible a la orientación de los normales, en muchas situaciones no es necesario desplazar la figura para que aparezca una textura espacial, sino que basta con desplazar los normales. Por ejemplo, para modelar una pared de ladrillos, además del mapa de texturas, es necesario que el modelo de iluminación genere la sensación de que en la superficie hay hendiduras.

Mapa de Transparencias: Permite modificar localmente el coeficiente de transmisión de un objeto translúcido, de manera que permite simular efectos de transparencia como por ejemplo nubes como elipsoides de transparencia aleatoriamente distribuida.

Mapa de Refracciones: Altera localmente el índice de refracción, por lo que permite producir efectos de ondas similares a los visibles en la superficie del agua.

En la Figura 7.20 podemos ver el uso de mapas para dar texturas a una escena graficada con ray tracing. Se utiliza un mapa de textura para el piso, lo cual permite simular las baldosas del piso. Además, las esferas tienen un mapa de normales que modifica localmente la reflexión especular,

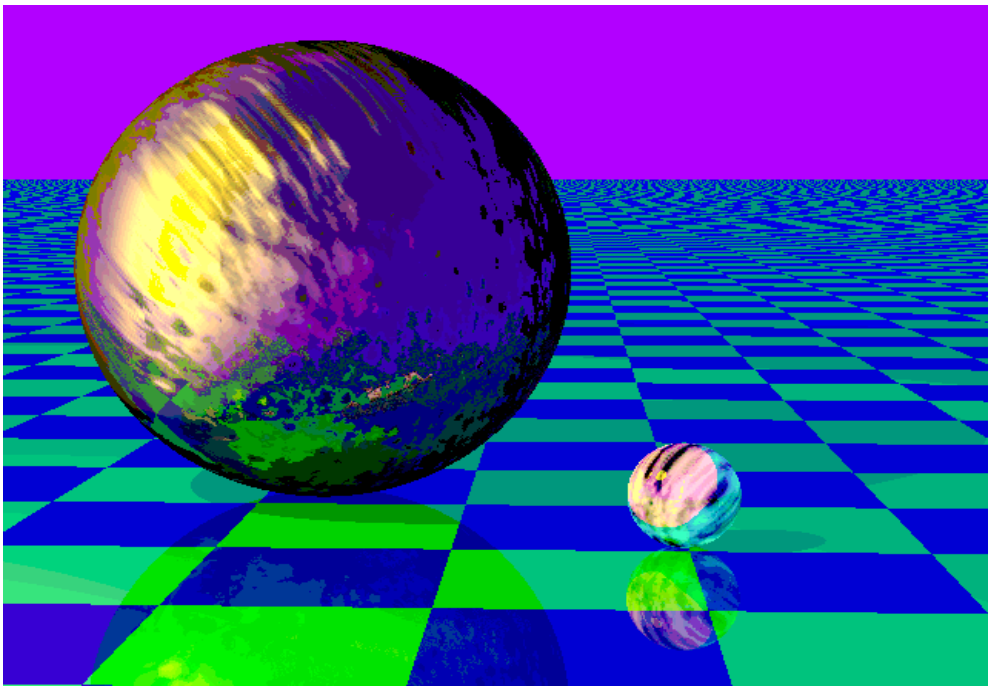


Figura 7.20 *Escena con mapas de atributos.*

por lo que aparece abollada sin estarlo (no podría aplicarse un mapa de desplazamientos porque el ray tracing no puede encontrar la intersección rayo-esfera abollada).

7.8 Ejercicios

1. Implementar el sombreado plano de polígonos y el de Gouraud. Testearlos con valores cualquiera de iluminación en los vértices.

2. Generar una escena con iluminantes, encontrar los valores de iluminación por medio del modelo de Bouknight (ambiente+Lambertiana) y sombrear con la técnica de Gouraud.

3. Repetir el ejercicio anterior con el modelo de iluminación de Phong y el sombreado de Phong. Comparar tiempos y resultados.

4. Implementar el Ray Tracing recursivo y graficar una escena sencilla en la que ocurran inter-reflexiones.

5. Agregar a los ejercicios 3 y 4 un mapa de texturas y de normales.

6. Discutir los pasos necesarios para encontrar un mapa de entorno (pauta: utilizar las caras de un cubo que rodea al objeto).

7.9 Bibliografía recomendada

La mayor parte de las referencias originales a los trabajos se fueron dando a lo largo del Capítulo. Sin embargo, como las mejoras en los modelos han sido objeto de estudio intensivo, siempre es preferible recurrir a un texto moderno que presente los resultados en forma ordenada y con una notación sistemática. Pocos son los libros en los que se aborda este tema como único asunto. Entre ellos recomendamos leer el libro de Hall [46]. Recomendamos consultar también [9, 54, 48] para mayores detalles.

Los modelos básicos de iluminación y sombreado se pueden consultar en el Capítulo 25 del libro de Newman-Sproull [66] y en el Capítulo 16 del libro de Foley et. al. [33]. Los lectores interesados en una presentación rigurosa y concisa pueden consultar el Capítulo 2 del libro de Watt y Watt [84]. Los algoritmos de Ray Tracing han generado una copiosa bibliografía. Es preferible acercarse a través de un enfoque gradual, como el presentado en el libro de Glassner [41], aunque la introducción al tema en los Capítulos 8, 9 y 10 del libro de Watt y Watt es bastante satisfactoria. Una presentación de los detalles de implementación de radiosidad puede consultarse en [21] y en el Capítulo 11 del libro de Watt y Watt. Las técnicas de mapeo de atributos están descriptas en varios artículos. Aquí recomendamos consultar [44, 51].

8 Aproximación de Superficies con Puntos de Control

8.1 Introducción

El tema de aproximación de superficies por medio de funciones biparamétricas governadas por puntos de control es uno de los más importantes en Computación Gráfica y CAD, dado que es el método por exelencia para el modelado de objetos sólidos. En rigor, se efectúa un modelo de la superficie de los objetos. Esto es suficiente, sin embargo, para realizar un rendering adecuado, dado que, como vimos, los modelos de iluminación y sombreado necesitan exclusivamente información geométrica (y óptica) de la superficie de los objetos. Los modelos de volúmenes son exclusivamente necesarios cuando algún elemento interior del sólido produce efectos que deben ser visualmente apreciables. Una presentación muy breve de este tema se puede encontrar en el próximo Capítulo.

Uno de los hechos más notables del modelado de superficies es que se pueden aplicar los modelos de interpolación y aproximación de curvas de una manera casi directa. En vez de contar con una secuencia de puntos de control, se tiene un arreglo bidimensional o matriz. Los modelos de interpolación y aproximación que vimos en el Capítulo 4 (con excepción del de Lagrange) producen una base funcional parametrizada que indica la importancia de cada punto de control en el resultado final de la curva. Por lo tanto, es matemáticamente sencillo formular la existencia de dos familias de bases, cada una con su parámetro (por fila y por columna, por ejemplo). El producto cartesiano de dichas bases produce una base biparamétrica, la cual, en función del valor de ambos parámetros, determina la importancia de cada punto de control del arreglo bidimensional en la superficie final.

En la práctica usual de la Computación Gráfica, las posiciones de los puntos de control en una superficie no determinan restricciones de exactitud, sino más bien una tendencia geométrica. Por dicha razón, los métodos de interpolación, que son geométricamente más inestables, han sido dejados de lado en el modelado de superficies, siendo los métodos de aproximación los más utilizados. En circunstancias especiales, donde se requiere definir que una superficie pase exactamente por un punto, se utilizan desarrollos inspirados en Splines pero con otro tipo de restricciones geométricas.

Como vamos a observar en la próxima Sección, las superficies de Bézier tampoco resultan satisfactorias para el modelado de objetos. Si la base de Bernstein promediaba excesivamente la geometría de los puntos de contron en las curvas de Bézier, en el caso de superficies, este efecto se potencia aún más, por lo que el resultado es practicamente inutilizable. Por dicha razón, el modelo más utilizado para generar superficies es el de B-Splines y sus variantes, especialmente los NURBS.

Por último, un tema de gran importancia tiene que ver con la configuración topológica de las superficies que se desea aproximar. Si nuestro punto de partida es un arreglo bidimensional de puntos de control (topológicamente equivalente a un plano), entonces la superficie aproximada debe poder desarrollarse a un plano. Esto excluye la posibilidad de modelar objetos con manijas, orificios, esquinas, etc. Por dicha circunstancia es que se desarrollaron modelos de aproximación de superficies con bases genuinamente biparamétricas, las cuales superan la limitación topológica de las bases bivariadas generadas por producto cartesiano.

8.2 Aproximación de Superficies I: Producto Tensorial de Curvas

Sea un arreglo rectangular de puntos de control $p_{ij} \in E^3$, con $0 \leq i \leq n$ y $0 \leq j \leq m$. Una superficie aproximante a dicho arreglo será una función biparamétrica, donde al variar los parámetros u y v se recorre la superficie según direcciones ortogonales (ver Figura 8.1).

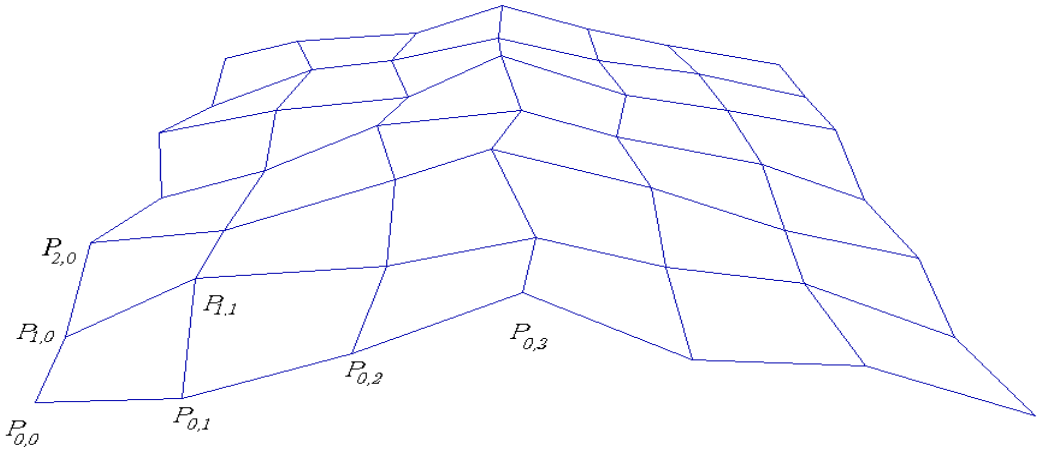


Figura 8.1 *Arreglo bidimensional de puntos de control.*

8.2.1 Superficies de Bézier

A partir del arreglo de $(n+1) \times (m+1)$ puntos de control encontramos una única superficie $\boldsymbol{S}$, de orden n según u y de orden m según v, $0 \leq u \leq 1$, y $0 \leq v \leq 1$ (ver Figura 8.2).

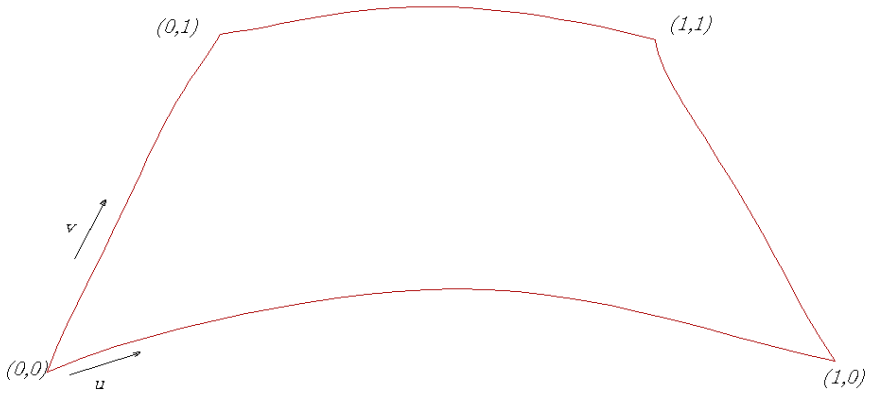


Figura 8.2 *Dominio de una superficie de Bézier.*

Recordemos que una curva de Bézier se encontraba como producto de una familia de funciones base por el arreglo de puntos de control:

$$\boldsymbol{C}(u) = [B_0^n(u), B_1^n(u), \cdots, B_n^n(u)] \begin{bmatrix} p_0 \\ p_1 \\ \cdots \\ p_n \end{bmatrix}.$$

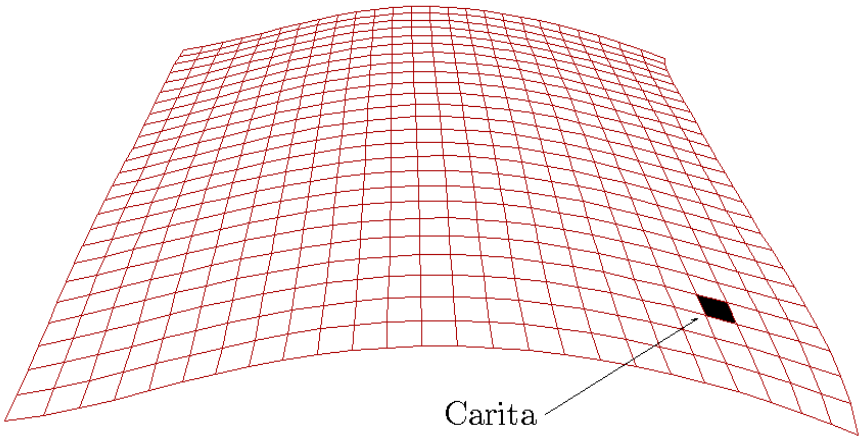


Figura 8.3 *Evaluación de una superficie de Bézier por "caritas".*

Si el arreglo de puntos de control es una matriz, entonces

$$\boldsymbol{C}(u) = [B_0^n(u), B_1^n(u), \cdots, B_n^n(u)] \begin{bmatrix} p_{00} & p_{10} & \cdots & p_{n0} \\ p_{01} & p_{11} & \cdots & p_{n0} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0m} & p_{1m} & \cdots & p_{nm} \end{bmatrix} = \begin{bmatrix} p_0(u) \\ p_1(u) \\ \cdots \\ p_n(u) \end{bmatrix},$$

es una familia de curvas parametrizadas en u.

Tomando cada $p_i(u)$ como punto de control paramétrico de una nueva curva, encontramos una expresión biparamétrica como producto tensorial de las familias de funciones base:

$$\boldsymbol{S}(u,v) = [B_0^m(v), B_1^m(v), \cdots, B_m^m(v)] \begin{bmatrix} p_0(u) \\ p_1(u) \\ \cdots \\ p_n(u) \end{bmatrix},$$

Reemplazando una expresión en la otra obtenemos

$$\boldsymbol{S}(u,v) = [B_0^m(v), B_1^m(v), \cdots, B_m^m(v)][B_0^n(u), B_1^n(u), \cdots, B_n^n(u)] \begin{bmatrix} p_{00} & p_{10} & \cdots & p_{n0} \\ p_{01} & p_{11} & \cdots & p_{n0} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0m} & p_{1m} & \cdots & p_{nm} \end{bmatrix}.$$

De la definición de producto de matrices se observa que

$$\boldsymbol{S}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{ji} B_j^n(u) B_i^m(v).$$

Normalmente esta expresión se evalúa unas 5×5 veces, y con los valores resultantes se arman "caritas" (ver Figura 8.3).

```
type grafo_contr =  array [0..pts_ctrol]  of
                    array [0..pts_ctrol]  of punto;

procedure bezier(var g:grafo_contr);
var i,pi,j,pj:integer;
    u,bu,v,bv:double;
    p:punto;
    aux: array [0..pasos] of punto;
begin
 p.w:=1;
 for pj:=0 to pasos do begin
    v:=pj/pasos;
    for pi:=0 to pasos do begin
      u:=pi/pasos;
      p.x:=0;  p.y:=0;  p.z:=0;
      for j:=0 to pts_ctrol do begin
        bv:=bernstein(v,j,pts_ctrol);
        for i:=0 to pts_ctrol do begin
          bu:=bernstein(u,i,pts_ctrol)*bv;
          p.x:=p.x + g[i][j].x*bu;
          p.y:=p.y + g[i][j].y*bu;
          p.z:=p.z + g[i][j].z*bu;
          end;
        end;
      graf_punto(aux[pi]);
      graf_linea(p);
      aux[pi]:=p;
    end;
    graf_punto(aux[0]);
    for pi:=1 to pasos do graf_linea(aux[pi]);
  end;
end;
```

Figura 8.4 *Implementación de las superficies de Bézier.*

La implementación de las superficies de Bézier puede observarse en la Figura 8.4. Como ya dijéramos más arriba, las curvas de Bézier son "perezosas" para adaptarse a cambios locales en el grafo de control. Esta misma característica se acentúa mucho más en el caso de superficies. Podemos ver en la Figura 8.5 de qué manera un gran cambio local produce escasamente una modificación el la superficie.

Una forma de conseguir que la curva se adapte a la forma deseada es "exagerar" enormemente los cambios en el grafo de control. De esa manera, para lograr un "pico" en la curva es necesario desplazar el punto de control asociado a una distancia mucho mayor que la deseada (ver Figura 8.6). Sin embargo, es posible darse cuenta que esta manera de trabajar no es lo suficientemente versátil y adecuada. Además, para curvas de grado alto, o con dos o más "picos", el resultado puede ser muy insatisfactorio.

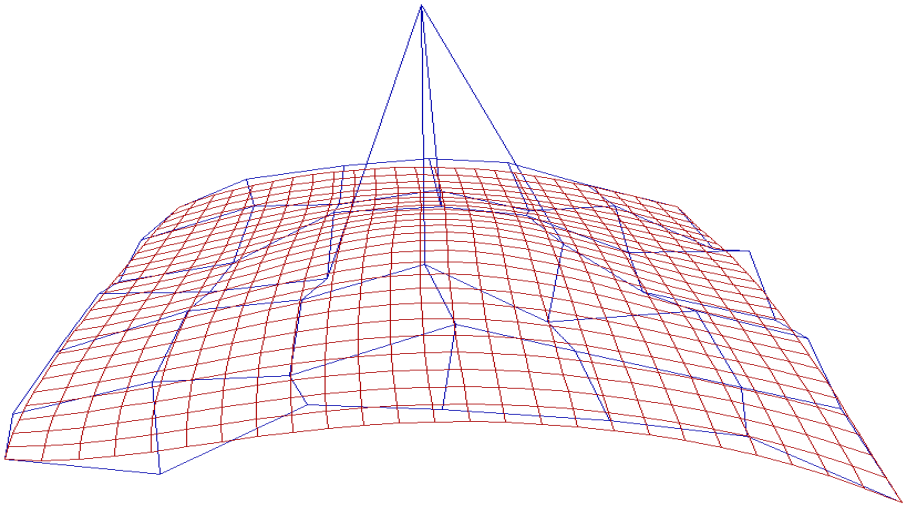


Figura 8.5 *Superficie de Bézier con un gran cambio local.*

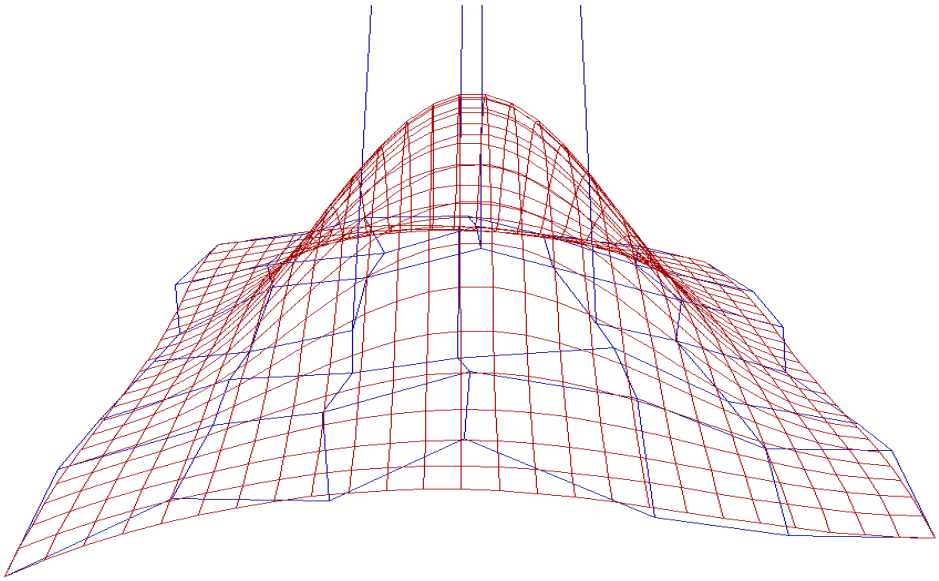


Figura 8.6 *Superficie de Bézier exagerando el cambio local.*

Para solucionar este tipo de inconvenientes es posible utilizar superficies de Bézier racionales. Éstas, al igual que las curvas de Bézier racionales, se caracterizan por asignar un *peso* a cada punto

de control (ver la bibliografía recomendada para el Capítulo 4). Es posible ver que los resultados son mucho más satisfactorios en el sentido de que la curva se adapta a la forma del grafo de control dándole peso adecuado a los puntos (ver Figura 8.7).

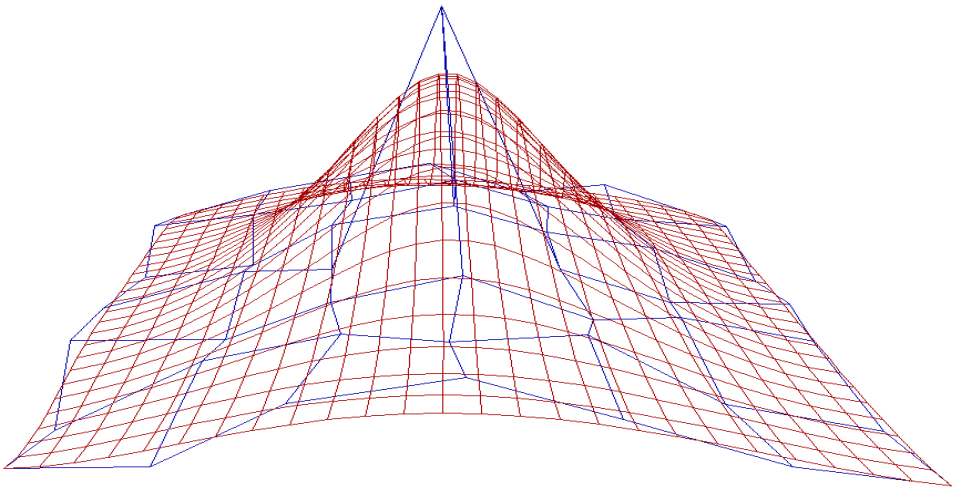


Figura 8.7 *Superficie de Bézier racional con cambio local.*

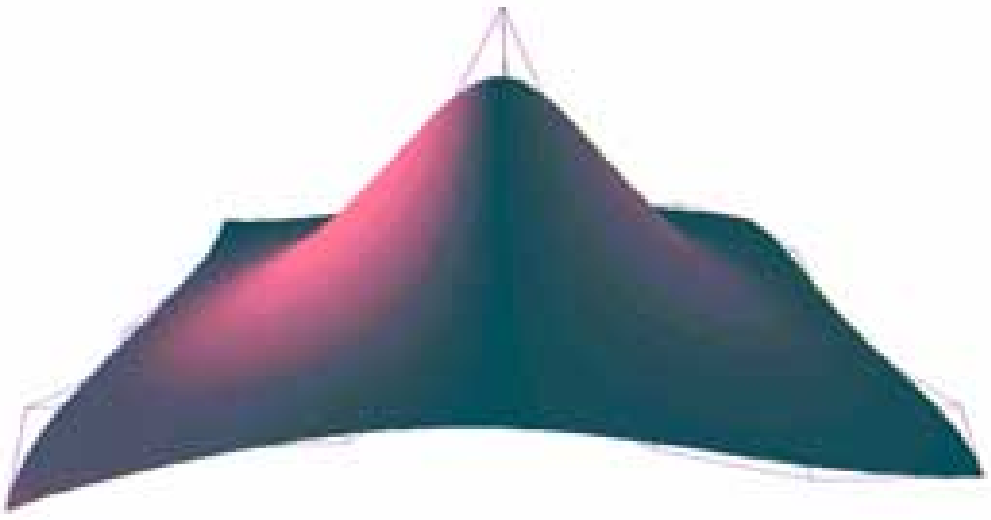


Figura 8.8 *Superficie de Bézier aplicando cara oculta y un modelo de iluminación.*

En este punto estaríamos en condiciones de poder elaborar nuestras primeras figuras graficadas con realismo. Por ejemplo, en la Figura 8.8 podemos ver la superficie de Bézier de la Figura 8.7 a la cual se le aplicaron técnicas de realismo vistas en el Capítulo anterior. Durante el procesamiento de la superficie, se agregaron instrucciones que invocan el scan line de la superficie antes de actualizar el vector de puntos auxiliares. De esa manera se construyen dos triángulos por carita: `triang(aux[pi],p,aux[pi+1])` y `triang(aux[pi],p,aux[pi-1])`. El procedimiento `triang` se encarga de graficar triángulos sombreados según su normal. El normal del triángulo se computa con el producto vectorial entre diferencias de componentes de los puntos. Con dicho normal es posible decidir si la cara es en principio visible, y además se puede computar un modelo de iluminación y aplicar una técnica de sombreado como las ya vistas. Al mismo tiempo, dado que la superficie se puede recorrer arbitrariamente, se puede aplicar el algoritmo del pintor o el de Wright para la eliminación de caras ocultas.

8.2.2 Superficies B-Spline bicúbicas

Partimos de un arreglo rectangular de $(m+1)\times(n+1)$ puntos de control, y tomando un subarreglo de 4×4 se define la superficie $\boldsymbol{S}$ como unión de $m-2\times n-2$ segmentos. El segmento o "parche" $\boldsymbol{S}_{kl}$ se define como producto tensorial de dos bases B-Spline cúbicas variadas en u y v:

$$\boldsymbol{S}_{kl}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} p_{j+k,i+l} N_j^3(u) N_i^3(v).$$

con $1 \leq k \leq m-2$ y $1 \leq l \leq n-2$. Recordando que la base es no nula solamente en cuatro intervalos, es posible simplificar la sumatoria a solamente 16 sumas ponderadas por las correspondientes subbases. De esa manera, la representación de un parche es

$$\boldsymbol{S}_{kl}(u,v) = \sum_{i=-2}^{1}\sum_{j=-2}^{1} b_j(v_k) b_i(u_l) p_{k+j,l+i},$$

con $1 \leq k \leq m-2$ y $1 \leq l \leq n-2$, y u_l y v_k son los parámetros locales del parche. Por construcción, la superficie $\boldsymbol{S}$ es C^2 contínua en las uniones entre parches.

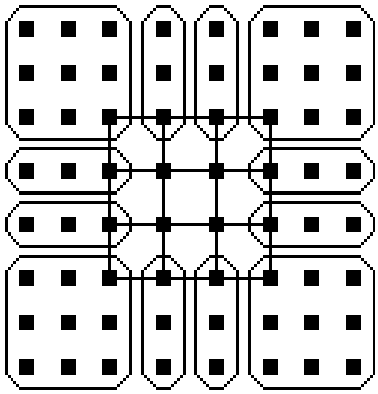
Con respecto a las condiciones de terminación de una superficie B-Spline, debemos tener en cuenta que podemos adoptar cualquiera de los dos esquemas (repetir puntos o asignar periódicamente) en cada una de las dimensiones del arreglo de puntos. De esa manera, tenemos tres opciones posibles: terminación no periódica, terminación periódica en una dimensión, y terminación periódica en las dos dimensiones. Cada una de estas opciones determina una *topología* diferente para la superficie resultante. En el primer caso obtenemos una superficie abierta, como un mapa de altitudes. Partiendo del siguiente arreglo de puntos:

$$\begin{matrix} p_{00} & p_{10} & p_{20} & p_{30} \\ p_{01} & p_{11} & p_{21} & p_{31} \\ p_{02} & p_{12} & p_{22} & p_{32} \\ p_{03} & p_{13} & p_{23} & p_{33} \end{matrix}$$

para obtener terminación con repetición de puntos de control debemos utilizar una matriz

$$\begin{matrix} p_{00} & p_{00} & p_{00} & p_{10} & p_{20} & p_{30} & p_{30} & p_{30} \\ p_{00} & p_{00} & p_{00} & p_{10} & p_{20} & p_{30} & p_{30} & p_{30} \\ p_{00} & p_{00} & p_{00} & p_{10} & p_{20} & p_{30} & p_{30} & p_{30} \\ p_{01} & p_{01} & p_{01} & p_{11} & p_{21} & p_{31} & p_{31} & p_{31} \\ p_{02} & p_{02} & p_{02} & p_{12} & p_{22} & p_{32} & p_{32} & p_{32} \\ p_{03} & p_{03} & p_{03} & p_{13} & p_{23} & p_{33} & p_{33} & p_{33} \\ p_{03} & p_{03} & p_{03} & p_{13} & p_{23} & p_{33} & p_{33} & p_{33} \\ p_{03} & p_{03} & p_{03} & p_{13} & p_{23} & p_{33} & p_{33} & p_{33} \end{matrix}$$

Este arreglo, esencialmente, repite los puntos de control de acuerdo al siguiente esquema:



En el segundo caso, la superficie resultante es cilíndrica, lo cual es de gran utilidad para modelar sólidos de revolución o figuras como jarras, copas, etc. Para obtener esta terminación 1-periódica debemos utilizar una matriz

$$\begin{array}{ccccccc}
p_{00} & p_{10} & p_{20} & p_{30} & p_{00} & p_{10} & p_{20} \\
p_{00} & p_{10} & p_{20} & p_{30} & p_{00} & p_{10} & p_{20} \\
p_{00} & p_{10} & p_{20} & p_{30} & p_{00} & p_{10} & p_{20} \\
p_{01} & p_{11} & p_{21} & p_{31} & p_{01} & p_{11} & p_{21} \\
p_{02} & p_{12} & p_{22} & p_{32} & p_{02} & p_{12} & p_{22} \\
p_{03} & p_{13} & p_{23} & p_{33} & p_{03} & p_{13} & p_{23} \\
p_{03} & p_{13} & p_{23} & p_{33} & p_{03} & p_{13} & p_{23} \\
p_{03} & p_{13} & p_{23} & p_{33} & p_{03} & p_{13} & p_{23}
\end{array}$$

En el tercer caso, la superficie es toroidal, lo cual permite modelar objetos que tengan un agujero, como por ejemplo jarras con manija. Para obtener esta terminación 1-periódica debemos utilizar una matriz

$$\begin{array}{ccccccc}
p_{00} & p_{10} & p_{20} & p_{30} & p_{00} & p_{10} & p_{20} \\
p_{01} & p_{11} & p_{21} & p_{31} & p_{01} & p_{11} & p_{21} \\
p_{02} & p_{12} & p_{22} & p_{32} & p_{02} & p_{12} & p_{22} \\
p_{03} & p_{13} & p_{23} & p_{33} & p_{03} & p_{13} & p_{23} \\
p_{00} & p_{10} & p_{20} & p_{30} & p_{00} & p_{10} & p_{20} \\
p_{01} & p_{11} & p_{21} & p_{31} & p_{01} & p_{11} & p_{21} \\
p_{02} & p_{12} & p_{22} & p_{32} & p_{02} & p_{12} & p_{22}
\end{array}$$

Podemos ver en la Figura 8.9 un ejemplo de figura con topología cilíndrica, modelado con 40 puntos de control, cuyo grafo se también se muestra.

8.3 Aproximación de Superficies II: Dominios Triangulares

Las superficies obtenidas por medio del producto tensorial de curvas son una solución adecuada para una clase bastante importante de problemas de aproximación. Sin embargo, la topología de

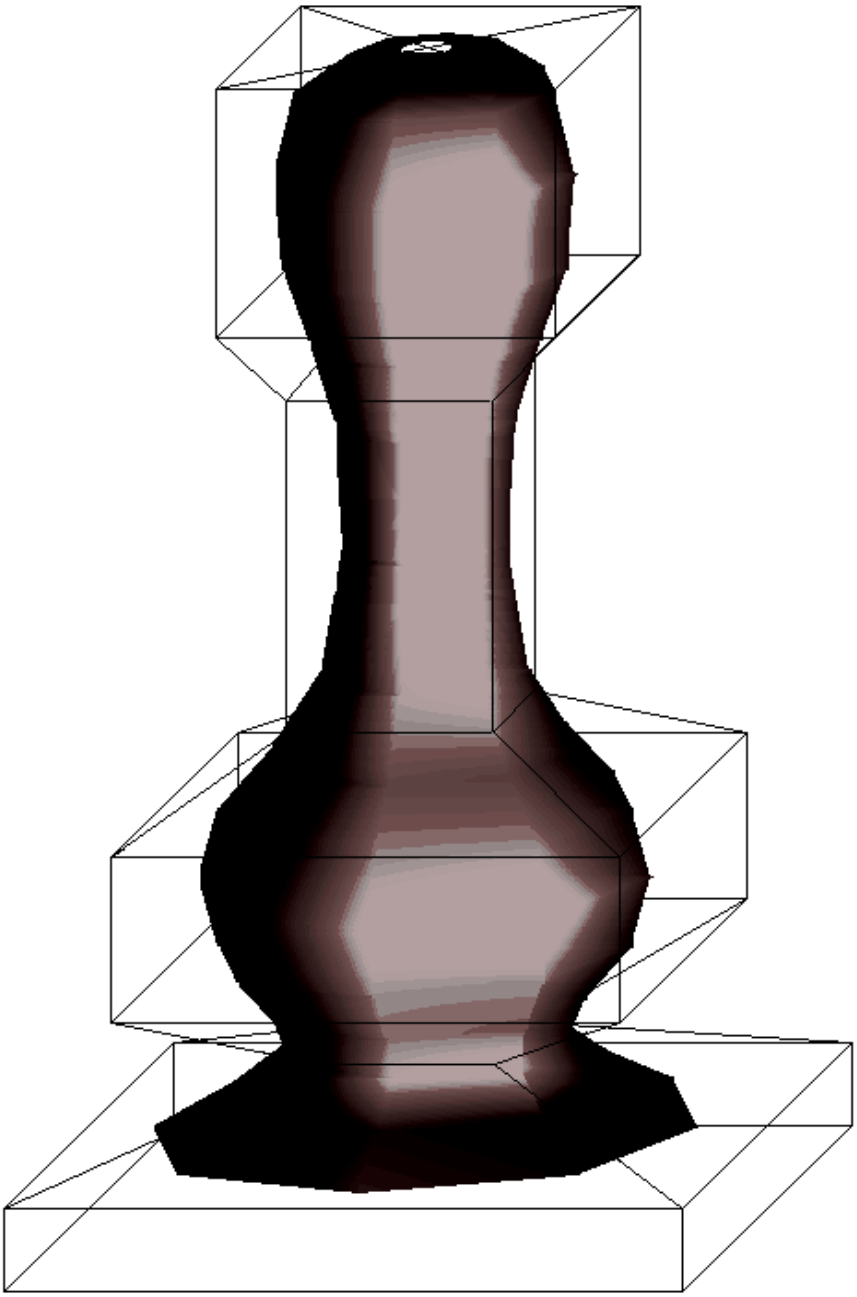


Figura 8.9 *Superficie B-Spline bicúbica con su correspondiente grafo de control.*

los modelos resultantes está limitada a tres casos posibles. Por dicha razón se estudian tambien superficies que provengan de un modelo genuinamente bivariado, es decir, donde la primitiva de aproximación sea directamente la superficie.

Una de las formas de realizar dichas aproximaciones se basa en el uso de dominios triangulares. De esa manera, la formulación de curvas de Bézier basada en polinomios de Bernstein o del algoritmo de de Casteljau se generaliza a parches triangulares.

8.3.1 Algoritmo de de Casteljau bivariado

Los puntos de control deben asumir una configuración triangular. Para poder denotarlos, necesitamos utilizar una notación especial, basada en multiíndices. Un multiíndice $\vec{\imath} = (i_1, i_2, \cdots, i_{k+1})$ es una tupla de $k+1$ enteros no negativos. La norma $|\vec{\imath}|$ de un multiíndice es $|\vec{\imath}| = \sum_{j=1}^{k+1} i_j$. También llamamos $e_1 = (1, 0, \cdots, 0)$, $e_2 = (0, 1, 0, \cdots, 0)$, etc.

Un triángulo no trivial establece un marco afín para representar un parámetro bivariado. Sea un triángulo con vértices p_1, p_2, p_3, y un cuarto punto p, en un mismo plano. La *representación baricéntrica* de p es

$$p = up_1 + vp_2 + wp_3,$$

con $u + v + w = 1$. $\boldsymbol{u} = (u, v, w)$ son las coordenadas baricéntricas de p. Si $u, v, w \geq 0$, es decir, es una suma convexa entonces p cae dentro del plano determinado por p_1, p_2, p_3.

La recurrencia de de Casteljau se establece sobre un arreglo triangular de puntos $p_{\vec{\imath}}$, donde $|\vec{\imath}|$ establece el grado de la superficie y la profundidad de la recursión. Por ejemplo, una superficie triangular cúbica de profundidad 3 queda determinada por los puntos $p_{\vec{\imath}}, |\vec{\imath}| = 4$

$$\begin{array}{ccccccccccc}
 & & & & p_{040} & & & & & & \\
 & & & p_{031} & & p_{130} & & & & & \\
 & & p_{022} & & p_{121} & & p_{220} & & & & \\
 & p_{013} & & p_{112} & & p_{211} & & p_{310} & & & \\
p_{004} & & p_{103} & & p_{202} & & p_{301} & & p_{400} & &
\end{array}$$

Un punto $p^k_{\vec{\imath}}(\boldsymbol{u})$ de la superficie que aproxima al grafo de control $p_{\vec{\imath}}$ con profundidad k es:

$$p^k_{\vec{\imath}}(\boldsymbol{u}) = up^{k-1}_{\vec{\imath}+e_1}(\boldsymbol{u}) + vp^{k-1}_{\vec{\imath}+e_2}(\boldsymbol{u}) + wp^{k-1}_{\vec{\imath}+e_3}(\boldsymbol{u}).$$

con $p^0_{\vec{\imath}}(\boldsymbol{u}) = p_{\vec{\imath}}$.

8.3.2 Polinomios de Bernstein multivariados

Como en el caso de las curvas de Bézier, en las superficies triangulares existe la posibilidad de encontrar una expresión polinomial equivalente al algoritmo de de Casteljau, de menor costo computacional. Sea un vector de $k+1$ variables $\boldsymbol{u} = (u_1, u_2, \cdots, u_{k+1})$, tales que $u_1 + u_2 + \cdots + u_{k+1} = 1$. El polinomio de Bernstein k-variado es

$$B^d_{\vec{\imath}}(\boldsymbol{u}) = \frac{d!}{i_1!i_2!\cdots i_{k+1}!} u_j^{i_j},$$

con $|\vec{\imath}| = d$ y $j \in \{1, 2, \cdots, k+1\}$.

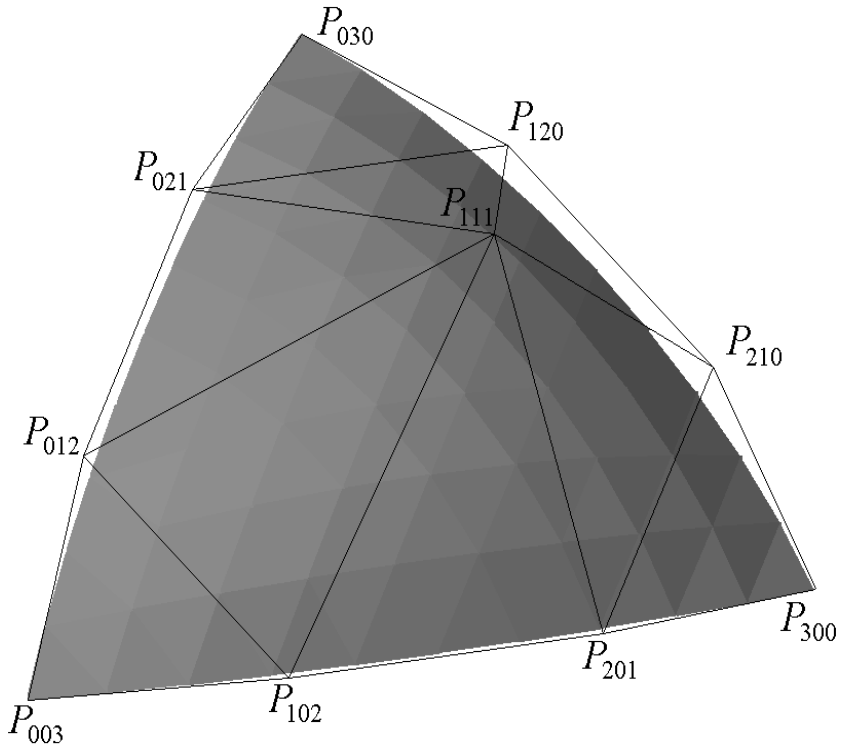


Figura 8.10 *Arreglo bidimensional de puntos de control.*

De esa manera, la forma de Bézier de la superficie bivariada proviene de la sumatoria de cada punto de control en el arreglo triangular por su función asociada en la familia de polinomios de Bernstein bivariados:

$$p^k_{\vec{\imath}}(\boldsymbol{u}) = B^d_{\vec{\imath}}(\boldsymbol{u}) p_{\vec{\imath}}.$$

En forma explicita

$$\boldsymbol{S}(u,v) = \sum_{i_0=0}^{k} \sum_{i_1=0}^{k-i_0} B^K_{(i_0,i_1,k-i_0-i_1)}(u,v,(1-u-v)) p_{(i_0,i_1,k-i_0-i_1)}.$$

En la Figura 8.10 podemos ver un arreglo triangular de puntos de orden 4 y la superficie triangular de Bézier resultante.

Estas superficies se utilizan en general en conjunción con las superficies rectangulares vistas en la Sección anterior, mayormente para solucionar los problemas topológicos que pueden llegar a ocurrir. Las configuraciones anómalas para los parches que se necesitan para el modelado de esquinas, agujeros, manijas, etc., pueden lograrse con parches con n lados, los cuales, a su vez, pueden modelarse con el orden de continuidad adecuado por medio de n parches triangulares (ver Figura 8.11). De esa manera, la conjunción de parches triangulares junto con parches rectangulares (por producto tensorial) permite cubrir adecuadamente una clase mucho más amplia de problemas (ver Figura 8.12).

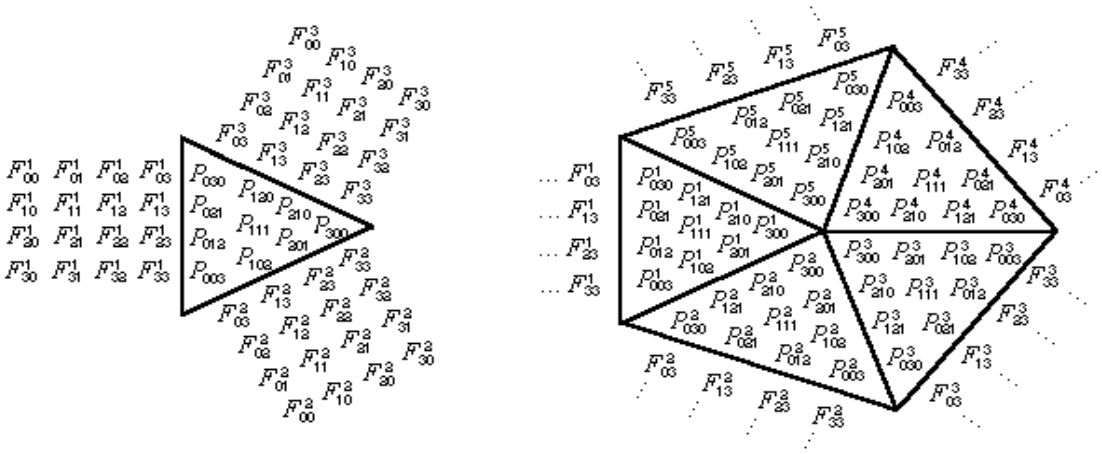


Figura 8.11 *Parches de n lados como n parches triangulares.*

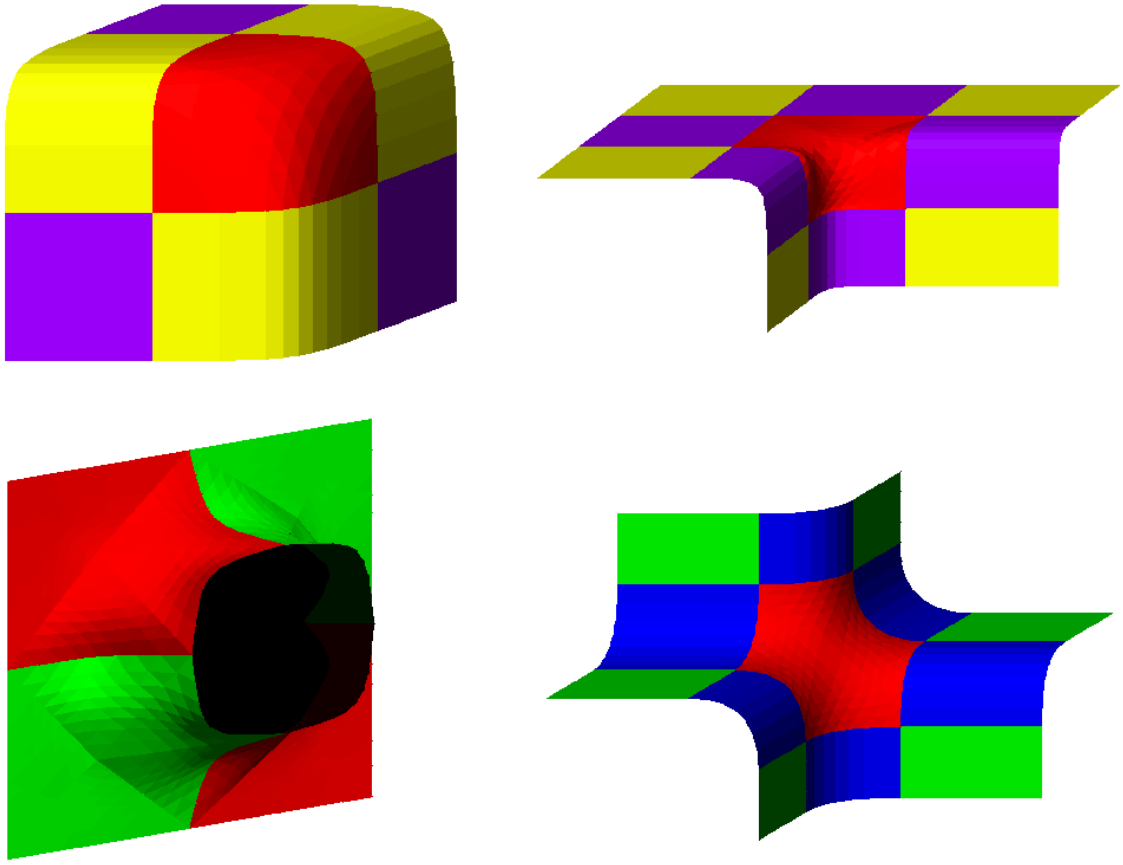


Figura 8.12 *Solución a problemas topológicos con parches de n lados.*

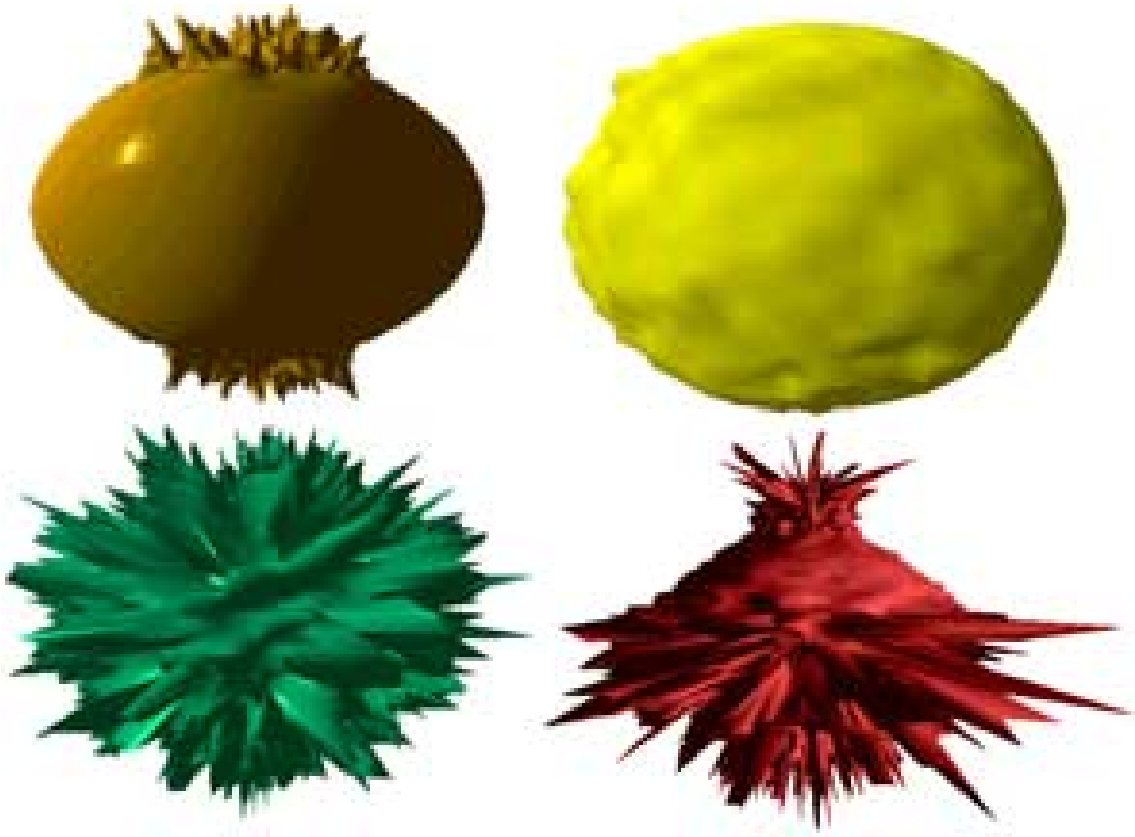
8.4 Ejercicios

1. Implementar los algoritmos de aproximación de superficies de Bézier y B-Spline cúbicos uniformes, como producto tensorial de curvas.
2. Aplicar los métodos a un grupo de grafos de control de prueba (por ejemplo, un tubo de teléfono, una cabeza, etc.).
3. Proponer soluciones para el armado de grafos de control de objetos no desarrollables, sin utilizar parches triangulares.
4. Implementar el método sencillo de construcción y sombreado de triángulos mostrado en el texto.
5. Implementar los parches de Bézier triangulares.

8.5 Bibliografía recomendada

Gran parte de las referencias recomendadas para el Capítulo 4 son adecuadas también para este Capítulo, en especial el libro de Farin [30] y el de Bartels et. al. [6]. Las discusiones relacionadas con la limitación topológica de la aproximación con superficies obtenidas con productos tensoriales, así como el desarrollo de los parches de Bézier triangulares, y B-Spline triangulares (no vistos aquí), puede consultarse en [58, 59, 45, 23].

9 Temas Avanzados



9.1 Modelos de refracción

Los modelos de refracción constituyen el desafío más significativo en la representación de realismo gráfico, por la riqueza de los efectos que se producen, y por la enorme complejidad involucrada en el cálculo. La técnica de rendering más satisfactoria al respecto parece ser en la actualidad el ray-tracing, porque esta forma de computar los modelos de refracción es geométricamente exacta por definición. Una objeción importante es que si bien el modelo es geométricamente exacto, la representación de objetos en los cuales el modelo es efectivamente (o prácticamente) computable es muy limitada.

Estas consideraciones llevan a investigar modelos de iluminación no locales dentro del marco de los sistemas de rendering tradicionales, es decir, los sistemas scan-line. El estado del arte en el rendering scan-line de refracciones, sin embargo, está lejos de ser satisfactorio. Los modelos más usuales ignoran la geometría de la refracción, y realizan únicamente un modelo de la transparencia del objeto traslúcido. Esta transparencia se puede computar interpolando el color del pixel p en función del color del objeto translúcido P_1 y del color del objeto (opaco) P_2 que se encuentra detrás: $I_f(p) = (1-k)I_{P_1}(p) + kI_{P_2}(p)$, donde $0 \geq k \geq 1$ es el coeficiente de transmitancia del objeto traslúcido, normalmente dependiente de la longitud de onda. Este esquema es denominado *transparencia interpolada* en [33]. Otro esquema posible es utilizar *transparencia filtrada*, en la cual se considera que P_1 "filtra" el color de P_2 con un coeficiente O de transparencia: $I_f(p) = I_{P_1}(p) + O.k.I_{P_2}(p)$.

Claramente se observa que estos mecanismos son *ad hoc*, dado que —al margen de ignorar la geometría de la refracción— plantean una fórmula de interpolación que no proviene de un modelo de iluminación. Una ulterior mejora fue propuesta por Kay y Greenberg [56], donde el coeficiente de transmitancia k es una función no lineal de z_n, la componente z del normal. Kay y Greenberg sugirieron una función $k = k_{min} + [k_{max} - k_{min}].[1-(1-z_n)^m]$, donde k_{min} and k_{max} son las transparencias mínima y máxima respectivamente, y m es un coeficiente arbitrario, generalmente entre 2 y 3, donde un m mayor modela un objeto traslúcido de menor grosor (ver Figura 9.1(a)).

En [67] se propuso modelo de rendering scan-line de objetos traslúcidos con características superiores a otros modelos presentados, dado que no solo se computa un modelo de iluminación adecuado, sino que ademas se considera *la geometría* de la refracción. Básicamente, el modelo aproxima la forma de los objetos traslúcidos interpolando superficies esféricas de la expresión poligonal de los mismos (ver Figura 9.1(b)). Dicha aproximación permite considerar a cada objeto traslúcido como una *lente*, para la cual es sencillo encontrar una expresión analítica de la geometría de la refracción, expresión que puede simplificarse considerando la aproximación paraxial de los rayos de luz [52].

La determinación de los elementos refractados se efectúa utilizando mapas de entorno procesados por medio de dicha expresión analítica. El modelo de iluminación considerado en este caso extiende el modelo local de Phong para incorporar la componente refractada, por lo cual es más adecuado que el modelo de ray tracing dado que le da un tratamiento uniforme a las distintas fuentes de iluminación, sean locales o globales. Una ulterior simplificación, considerar la geometría de la refracción en una lente delgada, permite encontrar una expresión más simple. En particular, se demuestra que en este caso la geometría de la refracción es sencillamente obtenible a partir de la geometría de la reflexión, y que por lo tanto un preprocesamiento de un mapa de reflexiones estándar permite computar la refracción. Por último, se considera la representación de objetos complejos como series de lentes, lo cual constituye una primera aproximación muy económica (ver Figura 9.2).

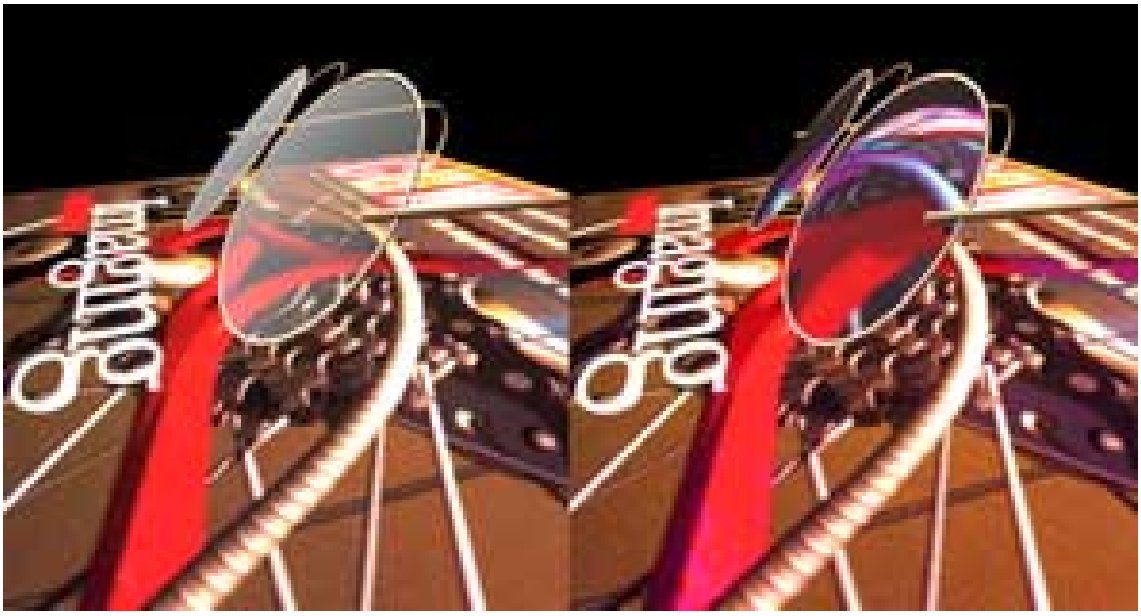


Figura 9.1 *Modelo de Kay-Greenberg* vs. *modelo de Patow.*

Figura 9.2 *Aproximación de objetos complejos como series de lentes.*

9.2 Animación

Por animación en Computación Gráfica entendemos bastante más que modelar el movimiento de los objetos y la cámara dentro de la escena. Una animación por computadora normalmente involucra, además, cambios en la forma y color de los objetos, en la iluminación, en las texturas o en las estructuras. Actualmente es posible reconocer las grandes posibilidades de la animación por computadora en las publicidades y en los video juegos y aplicaciones interactivas. Sin embargo, las aplicaciones de la animación van más allá, dado que es posible encontrarla en sistemas educacionales, simuladores de vuelo, usos industriales, visualización científica, y hasta en los cascos de realidad virtual.

Las técnicas básicas de animación por computadora se basan en la metodología convencional de los dibujos animados. Esencialmente se parte de lo que se denominan *key frames* o cuadros clave, donde cada cuadro es exactamente una de las tantas imágenes por segundo que se generan. Todo lo que sucede entre dos cuadros clave sucesivos puede interpolarse. Por lo tanto, dados dos cuadros clave sucesivos, y el tiempo que debe transcurrir entre ellos, es posible computar los cuadros intermedios por medio de técnicas de interpolación. A esta operación se la denomina *in betweening* [61], (ver Figura 9.3).

La técnicas que computan los estados intermedios de la escena reciben de los cuadros claves una descripción de cada elemento de la escena (objetos, iluminantes, posición de la cámara, etc.) por medio de una lista de parámetros, y su tarea consiste en calcular los valores de dicha lista para los cuadros intermedios, y luego graficar la escena con dichos valores. Esta interpolación no es necesariamente lineal, dado que los objetos pueden tener *leyes* de movimiento para conseguir determinados efectos. Por ejemplo, un objeto en caída libre describe una parábola en el tiempo, por lo que su ley de movimiento del primer cuadro clave al siguiente debe ser una parábola. Una aplicación importante de las curvas de Bézier es, entonces, la descripción de las leyes de movimiento en animación.

En los sistemas comerciales existe una estandarización de los parámetros que pueden animarse, y de las leyes que se utilizan. Nacen, de esa manera, los lenguajes para descripción de animaciones. Recomendamos al lector interesado que consulte el libro de N. Magnenat-Thalmann y D. Thalmann [61] o el Capítulo 21 del libro de Foley et. al. [34].

Las técnicas avanzadas de animación por computadora, sin embargo, no pueden quedarse en un nivel tan superficial de descripción. Por ejemplo, si en la animación de un objeto modelado con NURBS, la forma del mismo debe modificarse, lo lógico es modificar adecuadamente la posición de sus puntos de control cuadro por cuadro. Sin embargo, resulta imposible pensar que nuestro lenguaje de animación esté preparado para contemplar la interpolación de este tipo de parámetros.

Otros problema propios de la animación por computadora son la animación de objetos articulados y la animación de objetos no rígidos. Una técnica que ha ganado gran popularidad es la animación por medio de sistemas de partículas. Todos estos temas son de un nivel de complejidad considerable. Recomendamos al lector interesado que consulte los Capítulos 15, 16, 17, y 18 del libro de Watt y Watt [84].

9.3 Visualización Científica

Como ya dijéramos en la Introducción, la Visualización Científica significó todo un cambio de paradigma dentro de la Computación Gráfica, al buscar la representación gráfica de entidades abstractas, elaboradas a partir de conjuntos de datos. La Visualización Científica constituye ac-

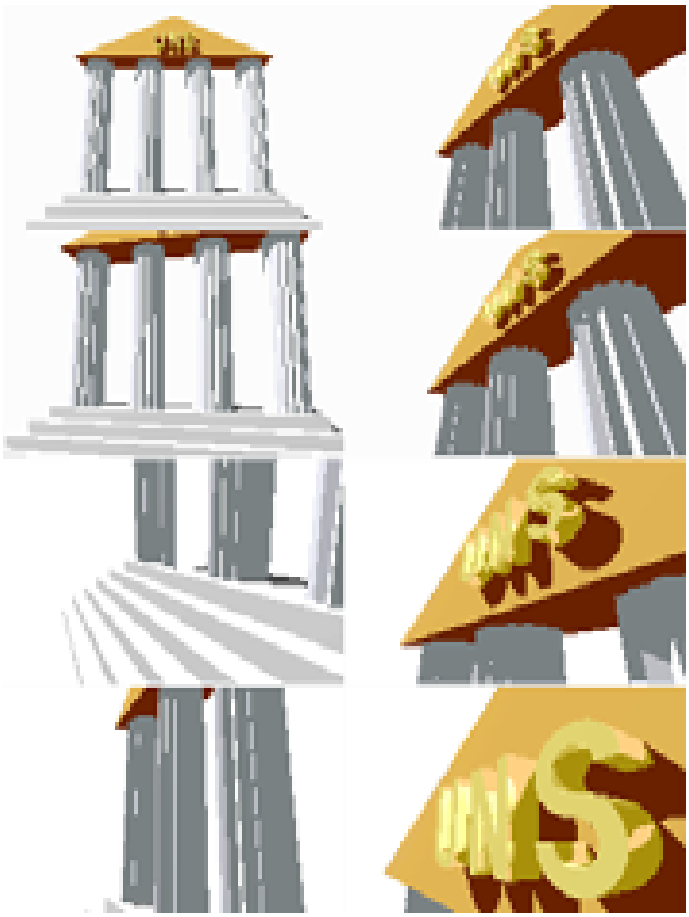


Figura 9.3 *Algunos cuadros de una secuencia animada.*

tualmente una verdadera revolución en la metodología de investigación científica, tanto básica como aplicada, comparable al invento del microscopio o el telescopio. Con el tiempo, la comunidad que estudia estos temas se fue apartando de la Computación Gráfica tradicional, y actualmente constituye de hecho una disciplina aparte, con sus propios congresos y publicaciones. Sin embargo, dada la enorme relevancia del tema, no queremos dejar de considerarlo aquí.

En la esencia de la Visualización está la representación de cantidades enormes de datos (probablemente multidimensionales) por medio de alguna metáfora visual que transmita adecuadamente la información. Se utilizan los sistemas computacionales no para simular sino para *representar* enormes conjuntos de datos, apelando a la vasta capacidad de interpretación visual del cerebro. En este sentido, la Visualización Científica representa la culminación de las actuales posibilidades de los sistemas de computación gráfica, dado que exigen una capacidad enorme de procesamiento. Por ejemplo, para manipular una base de datos tridimensional de $1000 \times 1000 \times 1000$ se requieren 1000M de datos. Estos datos pueden provenir de sensores, como en el caso de tomógrafos o de satélites, o bien pueden provenir de tareas computacionales anteriores, como por ejemplo de simulaciones o de análisis por elemento finito. El resultado gráfico que se espera de la visualización de estos datos no es meramente *cuantitativo* -no se busca necesariamente la representación fiel de valores- sino *cualitativo* -se busca un entendimiento global de determinadas propiedades de los datos [24, 74].

La Visualización involucra el empleo de técnicas derivadas de la Computación Gráfica utilizadas para la representación de datos científicos de los tipos más diversos. Dentro de la investigación en Visualización Científica, la representación de datos volumétricos se destaca por las dificultades computacionales que plantea, pero al mismo tiempo concentra la mayor atención en la investigación actual, dado que es actualmente de gran utlildad en la investigación científica en temas tan diversos como en matemática, medicina, ciencias naturales e ingeniería [37, 53], y es utilizada para representar datos que pueden provenir de diversas fuentes.

Los volúmenes de datos pueden pensarse abstractamente como matrices tridimensionales, en los que cada celda contiene valores uni o multivaluados. Estos datos fueron eventualmente preprocesados para extraer y enfatizar adecuadamente las características intuitivamente adecuadas en una determinada aplicación y para un determinado propósito en su visualización (por ejemplo, destacar determinadas áreas en la representación visual de una tomografía). Una vez que los datos volumétricos están adecuadamente preparados para el rendering, el mismo procede según algoritmos de mayor o menor sofisticación. Las técnicas usuales de rendering de volúmenes están normalmente asociadas a una representación de los mismos en estructuras de celdas volumétricas o en *voxels*. De esa manera, un voxel en particular representa el factor de ocupación que el sólido posee en una determinada fracción del espacio tridimensional.

Una de las primeras técnicas de rendering de volúmenes [32] consiste en graficar por capas el volumen de datos. Normalmente el volumen de datos se hace coincidir con los ejes del sistema de coordenadas del mundo, de modo que el eje z (hacia donde mira el observador) coincida con uno de los ejes del volumen de datos. Planos perpendiculares a dicho eje son entonces procesados de adelante hacia atrás. El procesamiento es sencillo, consistiendo en una proyección paralela de los datos al buffer de pantalla, utilizando alguna técnica de *pseudocoloring* para asociar los valores a representar con colores de una paleta predeterminada (por ejemplo, asociar un determinado color a un determinado tejido). Cada voxel, en función de su valor, tiene a su vez una determinada transparencia, es decir que no es necesariamente opaco, permitiendo que se visualize parcialmente las partes del volumen que se encuentran detrás. La transparencia en cada dirección visual se computa acumulándola en un α-buffer de pantalla [15]. Para emular una proyección tridimensional, los datos de las capas se van desplazando una determinada distancia en x e y a medida que éstas son más distantes en el eje z. Esta técnica es bastante primaria, pero por esa misma razón es implementable directamente con hardware específico. Su mayor limitación consiste en que, al

no existir un sólido propiamente dicho en ningún momento del procesamiento, no es posible una representación con realismo [28, 84], por ejemplo, la interacción con iluminantes o con otros objetos.

Otros métodos más sofisticados buscan extraer la *representación* de un objeto tridimensional a partir del volumen de datos. Una de las primeras técnicas [38] consiste en procesar capa por capa al volumen de datos, en función de un determinado valor umbral. De esa manera, es posible identificar en una capa dada aquellos voxels en los cuales ocurre una transición cercana al valor umbral. Dichos voxels conforman un contorno. Entre dos capas adyacentes, entonces, es posible vincular los contornos para determinar un esqueleto de polígonos. El conjunto de polígonos encontrado entre todas las capas procesadas de esta manera constituye una representación del sólido con una estructura "intermedia", en este caso, una superficie. Esta estructura de polígonos permite la visualización de los datos originarios, y tiene la ventaja de ser una estructura "tradicional" en el sentido de la computación gráfica, es decir, es un conjunto de polígonos, el cual puede graficarse con los algoritmos usuales, utilizando cara oculta, sombreado, iluminación, etc. Sin embargo, esta técnica encuentra problemas cuando no es directo encontrar el esqueleto de polígonos entre dos capas sucesivas (por ejemplo si ocurren discontinuidades topológicas).

Otra solución, más estable con respecto a este tipo de problemas, es la denominada "marching cubes" [60], en la cual se clasifican los voxels que pertenecen a una superficie umbral. Un voxel pertenece a la superficie umbral si por lo menos uno de sus vértices está por debajo del valor umbral y por lo menos otro está por encima. En este caso, cada uno de los ocho vértices de un voxel puede asumir un valor por debajo o por encima del umbral. El total de todos los casos posibles es $2^8 = 256$, pero por consideraciones de simetría se reducen a solo 14. Para cada uno de dichos casos es posible aproximar la superficie umbral con polígonos sencillos (normalmente triángulos) que cortan al voxel, y al mismo tiempo ubicar los voxels vecinos en los cuales dicha superficie debe continuar.

Una solución más completa (y tambien más compleja) para el rendering de volúmenes consiste en arrojar rayos desde el observador hacia el sólido, de una manera similar al ray tracing pero computando el comportamiento de la luz a través del volumen. Esta técnica, denominada ray casting, comienza por considerar que cada voxel es el modelo de un objeto físico, en el cual ocurre un fenómeno de interacción con la luz y con los rayos visuales provenientes de los demás voxels. Por lo tanto, es necesario establecer un modelo de iluminación que, a diferencia de los modelos tradicionales en Computación Gráfica como el de Phong, considere la interacción de la luz con una densidad volumétrica. Estos modelos fueron estudiados por Blinn [9] y por Kajiya [54, 55], llegando ambos a una formulación matemática similar.

Dada una densidad volumétrica $D(x, y, z)$ y un rayo visual $\vec{v}$ que la atraviesa entre dos puntos $t1$ a $t2$, consideraremos por un lado la iluminación acumulada en su interacción con una distribución de energía luminosa I que representa una determinada condición de iluminación, y por otro lado la densidad acumulada por el rayo desde que ingresa al sólido en $t1$. Sea entonces un punto t entre $t1$ y $t2$. La iluminación que recibe dicho punto es la sumatoria de las intensidades de las fuentes luminosas puntuales. El modelo considera que la densidad volumétrica puede pensarse como una distribución gaussiana de partículas idealmente especulares. Los algoritmos basados en esta técnica, normalmente simplifican esta ecuación según ciertas consideraciones. Por ejemplo, si los rayos visuales son paraxiales, es decir, con pequeña desviación angular respecto del eje z, entonces la distancia de $t1$ a $t2$ es constante en todos los voxels, y por lo tanto la integral puede aproximarse con una productoria.

Otra forma de graficar un volumen de datos es el procesamiento *cell by cell* (por celdas). Esta técnica consiste en ir recorriendo la base de datos en una forma ordenada de adelante hacia atrás (según la posición del observador) y se va proyectando dato por dato, ésto requiere un buffer con la información de los datos ya proyectados (color y transparencia acumulados). En arquitecturas compututacionales complejas, como por ejemplo en las máquinas Sillicon Graphics, el buffer de

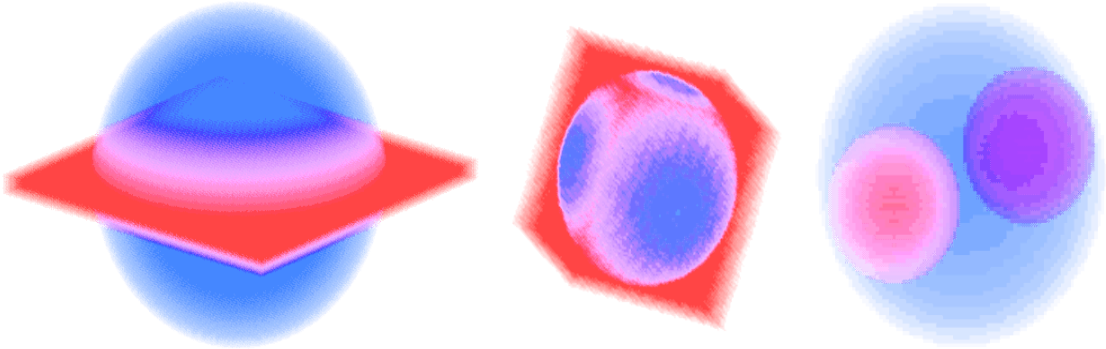


Figura 9.4 *Matrices volumétricas de datos procesadas por celdas.*

pantalla está pensado como para brindar soporte a este tipo de operaciones, es decir, se opta por la solución más natural del hardware dedicado. En máquinas PC, sin embargo, el buffer de pantalla no tiene capacidad para soportar estos cómputos intermedios, por lo que hay que recurrir a la memória de propósito general (RAM), con la consiguiente complicación en la programación, y los tiempos de cómputo mayores.

El procesamiento por celdas puede pensarse en forma diferente si se considera que los datos están formando voxels o formando celdas. El voxel es la mínima cantidad de información en 3D y vendría a ser lo que es el pixel en 2D. En cambio, la celda está formada por 8 datos que serían los vértices, entonces se puede realizar un promedio o bien una interpolación (trilineal) para obtener la información dentro de la celda y de esta forma evitar que en la imagen se perciban los pequeños cubos que la componen, cuando la matriz de datos es pequeña. La resolución de los gráficos está dada por el tamaño de la matriz de datos. Las etapas más importantes de este proceso son:

- Preprocesamiento de datos, etapa que comprende la generación y acondicionamiento de la matriz de datos o buffer volumétrico (*V-Buffer*).
- Adecuación a las transformaciones de proyección. De acuerdo con el punto de observación se elige el orden en que se van a ir tomando los datos para su proyección y se toma el dato correspondiente.
- *Voxelización*, etapa que consiste en armar un voxel con una cierta ubicación en el espacio, de acuerdo a la posición que tiene el dato en la matriz.
- Mapeo de color y transparencia, es decir, cada voxel va a tener un color y una transparencia asignada, de acuerdo al valor numérico del dato de la matriz que ha sido tomado.
- Proyección de las caras de cada voxel que son visibles sobre la pantalla computando la contribución de color y transparencia que aporta el voxel sobre la imagen.

En la Figura 9.4 podemos ver el resultado de procesar matrices volumétricas de datos con este procedimiento.

Deseamos dedicar en este último Capítulo por lo menos un par de hojas a cada uno de los temas que consideramos de gran importancia. De esa manera, el lector interesado puede consultar la bibliografía recomendada. Un tratamiento más profundo de estos temas puede ser motivo para la segunda parte de este libro.

9.4 Modelos no Determinísticos y Fractales

Una de las razones más importantes de la gran difusión de las técnicas de modelado con objetos fractales reside en su capacidad para producir una representación económica en tiempo y espacio de modelos o fenómenos naturales. El inconveniente clave de la geometría tradicional que se utiliza en la Computación Gráfica para representar objetos naturales es su falta de *simetría a escala*. Una costa podría describirse mediante segmentos, y si se tomara un número grande de ellos, se vería bastante real. Sin embargo, cuando nos acerquemos lo suficiente, esta costa revelaría su aspecto geométrico subyacente. Es por este motivo que los *objetos fractales* [62] están teniendo un uso creciente en la computación gráfica, especialmente aquellos basados en el modelo del ruido Browniano fraccional (fBm) [63].

Existen principalmente dos tipos de fractales: los determinísticos y los no determinísticos. Los primeros resultan generalmente de iterar sistemas de ecuaciones, y no incluyen al azar en su cálculo. Es decir, cada vez que se grafican, su aspecto es el mismo. Podemos citar muchos ejemplos notables, aunque el más popular es sin duda el conjunto de Mandelbrot (el mapa de complejos c cuyo conjunto de Juliá al iterar $z \leftarrow z^2 + c$ es conexo).

Los fractales no determinísticos, en cambio, utilizan explícitamente números aleatorios en determinadas etapas del cálculo. Por lo tanto, el resultado final es imprevisible. En la representacion de fenómenos naturales, como por ejemplo relieves montañosos, costas, nubes, etc., suelen usarse los fractales no determinísticos. El cálculo de los mismos depende solamente de un parámetro denominado *dimensión fractal* D, que intenta reflejar el aspecto global del resultado. Los demás aspectos quedan librados a lo que el generador de números aleatorios entregue en cada paso.

Una de sus características interesantes es que satisfacen la propiedad de *autosimilitud*, o sea, se ven estadísticamente similares (esto es "parecidos a primera vista") a sí mismos a diferentes escalas. Un *zoom* a un objeto fractal no causará una pérdida de naturalidad de su aspecto, sino que, por el contrario, incorporará nuevos detalles no visibles a menor escala. La autosimilitud puede concebirse como una simetría a escala. Si el objeto es un fractal no determinístico, esta similitud se da en términos estadísticos, es decir, las propiedades geométricas del objeto tienen distribuciones semejantes a diferentes escalas. En algunos casos particulares pueden existir modelos matemáticos para determinados fenómenos naturales cuya fractalidad es un fenómeno emergente. El interés de la Computación Gráfica en los fractales, sin embargo, no es realizar una simulación exhaustiva de dichos fenómenos, sino en producir herramientas computacionales que los imiten adecuadamente en un tiempo razonable.

Un modelo matemático de gran utilidad para un gran conjunto de series de tiempo es el *movimiento Browniano fraccional* (fBm) [63]. La computación del fBm puede hacerse por medio de la transformada rápida de Fourier (FFT), pero en la práctica resultan más adecuados los métodos en el dominio tiempo. Las técnicas más comunes para efectuar dichos cálculos son

Cortes Independientes [68]: Es el primer método históricamente implementado para reproducir terrenos. Consiste en generar números aleatorios de importancia decreciente, los cuales son sumados a un arreglo en una cantidad de direcciones que sigue la proporción $\Delta V = \alpha(\Delta t)^{2-D}$. La *fase* de dichos cortes (la dirección de comienzo) es determinada aleatoriamente.

Cortes Secuenciales: Es una variación del anterior que tiene la ventaja de poder ser calculado en secuencia, sin requerir calcular todo el arreglo. Por ello es aplicable a la generación de números aleatorios que sigan una distribución fraccional. Consiste en tener un acumulador al que en cada tirada se le agrega una cantidad aleatoria. Esta cantidad tiene componentes proporcionales a períodos irracionales, cuyas amplitudes son inversamente proporcionales a los períodos. Una metáfora para entenderlo mejor es suponer que tenemos varios dados,

Figura 9.5 *Terrenos fractales computados con el algoritmo de desplazamiento del punto medio.*

algunos con números pequeños, otros medianos y otros grandes. En cada tirada se arrojan ciertos dados. La probabilidad de que un dado sea arrojado es inversamente proporcional a los números que figuran en él.

Desplazamiento Aleatorio del Punto Medio [35]: Este método es computacionalmente más eficiente que los anteriores, y algorítmicamente es más sencillo de entender. Se toma como comienzo y como final dos puntos dados $V(0)$ y $V(1)$. Se calcula el V promedio en el punto medio, y a ese valor se le agrega un ΔV proporcional al Δt según la ecuación ya vista. Luego, cada mitad es tratada recursivamente de forma similar.

La extensión del algoritmo del punto medio a dos dimensiones puede aplicarse para generar una distribución bidimensional fractal de alturas. En este caso el punto medio tiene en cuenta las alturas de sus cuatro vecinos. Es decir, cada punto medio está en el centro de una cara, cuyos vértices son tenidos en cuenta para calcular su altura esperada. Esta altura es desplazada aleatoriamente en función de la escala. Con los puntos medios en el centro de cada cara es posible ahora calcular la altura en los puntos medios de cada arista. Estos dos pasos permiten pasar de una retícula de 1×1 y cuatro puntos a una retícula de 2×2 caras y nueve puntos. A cada cara de la retícula generada se le aplica recursivamente el mismo algoritmo.

La Figura 9.5 muestra dos mapas de alturas fractales desarrollados con el método de desplazamiento del punto medio. Estos resultados, además de permitir la simulación de terrenos, pueden utilizarse como mapas de texturas para otro tipo de objetos. Por ejemplo, en la Figura 5.17 se utilizaron fractales como mapa de desplazamientos en un algoritmo scan-line, y en la Figura 7.19 se utilizaron como mapa de normales en un algoritmo de ray tracing.

A Notación

p, q	Puntos en un espacio afín dado.
x, y, z	Coordenadas en un espacio vectorial.
h	Coordenada homogénea.
`x,y`	Coordenadas en el espacio de pantalla.
s, t	Parámetros escalares.
T, T_1, T_2	Transformaciones en un espacio afín dado.
`r,g,b`	Atributos de un color representado en RGB.
C^k	Orden de continuidad de una curva.
$\boldsymbol{C}(u)$	Curva paramétrica representada con funciones polinomiales.
$\dot{\boldsymbol{C}}(u)$	Derivada paramétrica de la curva, dado un valor del parámetro.
$\dot{p}$	Derivada de una curva interpolante al pasar por el punto (nudo) p.
$\boldsymbol{S}(u, v)$	Superficie paramétrica representada con funciones polinomiales.
u	Parámetro global (o único) de una curva paramétrica.
u, v	Parámetros de una superficie paramétrica.
$\boldsymbol{C}_i(u)$	i-ésimo segmento de una curva paramétrica.
u_i	Parámetro local del i-ésimo segmento de una curva paramétrica.
p_i	Puntos de control para una curva o superficie.
$\overline{p, q}$	Segmento de recta que une los puntos p y q.
p_i^k	Puntos de interpolación intermedia.
B_i^n	Base de Bernstein para polinomios de grado n.
N_i^n	Base de Splines para curvas de grado n.
$b_{-2}(u)..b_1(u)$	Sub-bases de Splines cúbicos uniformes.
I, I_a, I_p	Intensidad luminosa (ambiente, puntual).
k_a, k_p, k_d	Coeficientes de reflexión ambiente, especular y difuso.
N, L, R	Vectores unitarios (Normal, incidencia Luminosa, Reflexión).

B Elementos Matemáticos de la Computación Gráfica

B.1 Álgebra de Matrices

Una matriz A es un arreglo de elementos escalares a_{ij}:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Si $m = 1$ entonces estamos en el caso particular de un *vector fila*:

$$\boldsymbol{f} = [v_1, v_2, \cdots, v_n]$$

Si $n = 1$ entonces es un vector columna:

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

La traspuesta A^t de A es una matriz tal que

$$a^t_{ij} = a_{ji}$$

La suma de dos matrices A y B es una matriz C tal que

$$c_{ij} = a_{ij} + b_{ij}$$

El producto de un escalar s por una matriz A es una matriz B tal que

$$b_{ij} = sa_{ij}$$

El producto de dos matrices conformes A [m×k] y B [k×n] es una matriz C[m×n] tal que

$$c_{ij} = \sum_{l=1}^{k} a_{il} b_{lj}$$

La submatriz A_{ij} de la matriz A es la matriz resultante de eliminar de A la i-ésima fila y la j-ésima columna.

El determinante $|A|$ de una matriz A es un escalar definido recursivamente como

$$|A| = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} |A_{ij}|.$$

Si $|A| = 0$, entonces A es una matriz *singular*.

La matriz adjunta de A es una matriz A^a tal que

$$a^a_{ij} = (-1)^{i+j} |a_{ij}|.$$

La inversa A^{-1} de A es una matriz tal que

$$a^{-1} = \frac{(a^a)^t}{|A|}.$$

Si A es singular entonces su inversa no está definida.

B.2 Álgebra de vectores

Un espacio lineal (o vectorial) es un conjunto de valores (denominados vectores) cerrado bajo suma y multiplicación escalar. Debe existir un elemento 0 (identidad de la suma) denominado *origen*. La descripción usual de los elementos de un espacio vectorial es a partir de *coordenadas* respecto de una *base*.

Ejemplo: En $\boldsymbol{R}^3$ la base usual es $\boldsymbol{E}^3 = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$, con

$$\boldsymbol{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \boldsymbol{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

El vector $\boldsymbol{v} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$ se representa directamente como $\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$

dado que $\boldsymbol{v} = 2\boldsymbol{e}_1 + 3\boldsymbol{e}_2 + 4\boldsymbol{e}_3$.

El **módulo** o longitud de un vector es su norma-2:

$$\| \boldsymbol{v} \| = \sqrt{v_1^2 + v_2^2 + \cdots + v_l^2}.$$

El **producto escalar** o interno de dos vectores es

$$\boldsymbol{v} \cdot \boldsymbol{w} = v_1 w_1 + v_2 w_2 + \cdots + v_l w_l.$$

($\boldsymbol{v}$ es de $1 \times l$ y $\boldsymbol{w}$ es de $l \times 1$. Entonces esta definición coincide con el producto de matrices).

Es también útil la identidad

$$\boldsymbol{v} \cdot \boldsymbol{w} = \| \boldsymbol{v} \| \| \boldsymbol{w} \| \, cos(\theta),$$

donde θ es el ángulo entre ambos vectores.

El **producto vectorial** de dos vectores es

$$\boldsymbol{v} \times \boldsymbol{w} = \begin{bmatrix} (v_2 w_3 - v_3 w_2) \\ (v_3 w_1 - v_1 w_3) \\ (v_1 w_2 - v_2 w_1) \end{bmatrix}.$$

Es también útil la identidad

$$\boldsymbol{v} \times \boldsymbol{w} = \| \boldsymbol{v} \| \| \boldsymbol{w} \| \, sen(\theta),$$

donde θ es el ángulo entre ambos vectores.

B.3 Transformaciones Lineales y Afines

Dado un espacio vectorial $\boldsymbol{R}^n$ y dos elementos (vectores) u y v de dicho espacio. Entonces una transformación $T : \boldsymbol{R}^n \to \boldsymbol{R}^n$ se denomina *lineal* si se cumple que para escalares α, β arbitrarios

$$T(\alpha u + \beta v) = \alpha T(u) + \beta T(v).$$

Es importante observar que las traslaciones no son transformaciones lineales. En particular, es fácil ver que una transformación arbitraria queda definida por los coeficientes por los que transforma a la base del espacio vectorial. Por dicha razón, una transformación lineal puede representarse con una matriz de $n \times n$, y la transformación de un vector es el producto del mismo por dicha matriz.

Las transformaciones lineales son bien estudiadas en los cursos de matemática, no así las afines, las cuales son indispensables en Computación Gráfica. Dado un espacio vectorial $\boldsymbol{R}^n$ y dos elementos (vectores) u y v de dicho espacio. Entonces una transformación $T : \boldsymbol{R}^n \to \boldsymbol{R}^n$ se denomina *afín* si se cumple que para un escalare α arbitrario

$$T((1 - \alpha u) + \alpha v) = (1 - \alpha) T(u) + \alpha T(v)$$

. Es importante observar que las transformaciones lineales son afines pero no a la inversa. En particular, una transformación afín puede representarse como una transformación lineal compuesta con una traslación (ver [42]).

La representación de una transformación afín puede hacerse de dos maneras. La primera, menos usual, es utilizar un marco afín de referencia y definir un sistema de coordenadas baricéntrico dentro de dicho marco. Por ejemplo, dos puntos P, Q distintos definen un marco afín unidimensional (un subespacio), y cualquier punto $R = (1 - \alpha)P + \alpha Q$ pertenece a la recta que los une. En este caso

el par $(1-\alpha), \alpha$ es la coordenada baricéntrica de R respecto del marco afíi n P, Q. El concepto de marco afín puede extenderse a cualquier espacio Euclídeo. Una transformación lineal en un marco afín representa, entonces, una transformación afín en un espacio lineal.

La seguna manera de representar transformaciones afines, la usual en Computación Gráfica, consiste en recorrer el camino inverso, es decir, encontrar un superespacio lineal S, del cual nuestro espacio usual E es un espacio afín. Entonces, las transformaciones lineales en S serán transformaciones afines en E. S debe ser, entonces, un espacio de una dimensión mayor que E, obtenido por *homogenización*. El proceso está detalladamente descripto en la Subsección 3.2.2 del texto.

Referencias

[1] T. Akimoto, K. Mase, and Y. Suenaga. Pixel-Selected Ray Tracing. *IEEE Computer Graphics and Applications*, 11(4):14–22, 1991.

[2] John Amanatides. Realism in Computer Graphics: A Survey. *IEEE Computer Graphics and Applications*, 7(1):44–56, January 1987.

[3] Brian Barsky. A Description and Evaluation of Various 3D Models. *IEEE Computer Graphics and Applications*, 4(1):38–52, January 1984.

[4] Brian Barsky and John Beatty. Local Control of Bias and Tension in Beta Splines. *ACM Computer Graphics*, 17(3):193–218, July 1983.

[5] Brian Barsky and Tony DeRose. Geometric Continuity of Parametric Curves: Three Equivalent Characterizations. *IEEE Computer Graphics and Applications*, 9(6):60–68, November 1989.

[6] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*. Springer-Verlag, New York, 1987.

[7] J. F. Blinn and M. E. Newell. Texture and Reflection in Computer Generated Images. *Communications of the ACM*, 19(10):542–547, October 1976.

[8] James F. Blinn. Models of Light Reflection for Computer Synthesized Pictures. *ACM Computer Graphics*, 11(2):192–198, 1977.

[9] James F. Blinn. Light Reflection Function for Simulation of Clouds and Dusty Surfaces. *ACM Computer Graphics*, 16(3):21–29, 1982.

[10] James F. Blinn. Where am I? Where am I Looking at? *IEEE Computer Graphics and Applications*, 8(4):76–81, July 1988.

[11] James F. Blinn. A Trip Down the Graphics Pipeline: Line Clipping. *IEEE Computer Graphics and Applications*, 11(1):98–105, January 1991.

[12] James F. Blinn. A Trip Down the Graphics Pipeline: Pixel Coordinates. *IEEE Computer Graphics and Applications*, 11(4):81–85, July 1991.

[13] James F. Blinn. A Trip Down the Graphics Pipeline: Grandpa, What Does "Viewport" Mean? *IEEE Computer Graphics and Applications*, 12(1):83–87, January 1992.

[14] James F. Blinn. A Trip Down the Graphics Pipeline: The Homogeneous Perspective Transform. *IEEE Computer Graphics and Applications*, 13(3):75–80, May 1993.

[15] James F. Blinn. Compositing I — Theory. *IEEE Computer Graphics and Applications*, 14(5):83–87, 1994.

[16] W. K. Bouknight and K. C. Kelley. An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources. In *Proceedings of the AFIPS, vol. 36*, 1970.

[17] P. J. Bouma. *Physical Aspects of Colour*. Philips Research Lab., 1947.

[18] Jack Bressenham. Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal*, 4(1), 1965. Reimpreso en [36], páginas 106–111.

[19] I. Carlbom and J. Paciorek. Planar Geometric Projections and Viewing Transformations. *ACM Computing Reviews*, 10(4):465–502, 1978.

[20] Silvia Castro, Claudio Delrieux, and Andrea Silvetti. Una Interfase Amigable Orientada a Desarrollos en Computación Gráfica. In *INFOCOM '95*, Buenos Aires, Argentina, 1995. Congreso Internacional de Informática y Telecomunicaciones.

[21] Michael F. Cohen, Donald P. Greenberg, David S. Immel, Philip J. Brock, S. Casey, and N. Reingold. An Efficient Radiosity Approach for Realistic Image Synthesis. *IEEE Computer Graphics and Applications*, 6(2):26–35, 1986.

[22] William Cowan. An Inexpensive scheme for Calibration of a Color Monitor in terms of CIE Standard Coordinates. *Computer Graphics*, 17(3):51–72, 1983.

[23] T. de Rose and R. Goldman. Functional Composition Algorithms via Bloosoming. *ACM Transactions on Graphics*, 12(2):115–135, 1993.

[24] T. A. Defanti, M. D. Brown, and B. H. McCormick. Visualization: Expanding Scientific and Engineering Research Opportunities. In G. M. Nielson and B. D. Shriver, editors, *Visualization in Scientific Computing*, pages 32–47. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[25] C. Delrieux, S. Castro, A. Silvetti, and S. Anchuvidart. Paletas Estáticas para Gráficos de Alta Calidad en PC. In *XXII CLEI - Latin American Conference on Informatics*, Gramado, Brasil, 1995. SBC.

[26] Claudio Delrieux. Curvas y Superficies para Computación Gráfica y CAD. Technical Report Escuela de Ciencias Informáticas, UBA-FCEN, Departamento de Computación, 1998.

[27] Claudio Delrieux, Daniel Formica, Fernando Caba, and Esteban Pedroncini. Una Solución Eficiente al Problema de la Cara Oculta. *Revista Telegráfica Electrónica*, 934, 1991.

[28] R. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. *ACM Computer Graphics*, 22(4):65–74, 1988.

[29] Gouraud F. Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers*, 20(6):623–629, 1971.

[30] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, New York, 1988.

[31] Hamid Faroosh and Gunther Schrack. CNS-HLS Mapping Using Fuzzy Sets. *IEEE Computer Graphics and Applications*, 6(6):28–35, June 1986.

[32] J. Farrell. Colour Display and Interactive Interpretation of Three-Dimensional Data. *IBM Journal of Research and Development*, 27(4):356–366, 1983.

[33] J. Foley, A. Van Dam, S. Feiner, and J. Hughes. *Computer Graphics. Principles and Practice*. Addison-Wesley, Reading, Massachussetts, second edition, 1990.

[34] T. Foley, D. Lane, G. Nielson, and R. Ramaraj. Visualizing Functions over a Sphere. *IEEE Computer Graphics and Applications*, 10(1):32–40, 1990.

[35] A. Fournier, D. Fussell, and L. Carpenter. Computer Rendering of Stochastic Models. *Communications of the ACM*, 25(6):371–384, 1982.

[36] Herbert Freeman. *Tutorials and Selected Readings in Interactive Computer Graphics*. IEEE Press, 1980.

[37] H. Fuchs, M. Levoy, and J. K. Lam. Interactive Visualization of 3D Medical Data. In G. M. Nielson and B. D. Shriver, editors, *Visualization in Scientific Computing*, pages 140–146. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[38] S. Ganapathy and T. Dennehy. A New General Triangulation Method for Planar Contours. *ACM Computer Graphics*, 16(3):69–75, 1983.

[39] Michael Gervautz and Werner Purgathofer. A Simple Method for Color Quantization: Octree Quantization. In Andrew S. Glassner, editor, *Graphics Gems*. Academic Press, 1990.

[40] W. K. Giloi. *Interactive Computer Graphics - Data Structures, Algorithms, Languages*. Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[41] A. Glassner. *An Introduction to Ray Tracing*. Academic Press, Cambridge, Massachussets, 1991.

[42] Jonas Gomes, Lucia Darsa, Bruno Costa, and Luiz Velho. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, San Francisco, 1999.

[43] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the Interaction of Light Between Diffuse Surfaces. *ACM Computer Graphics*, 18(3):213–222, 1984.

[44] Ned Greene. Environment Mapping and Other Applications of Word Projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, 1986.

[45] G. Greiner and H. Seidel. Modelling with Triangular B-Splines. *IEEE Computer Graphics and Applications*, 14(2):56–60, 1994.

[46] Roy Hall. *Illumination and Color in Computer Generated Imaginery*. Springer-Verlag, New York, 1988.

[47] Roy A. Hall and Donald P. Greenberg. A Testbed for Realistic Image Synthesis. *IEEE Computer Graphics and Applications*, 3(8):10–20, 1983.

[48] X. D. He and K. E. Torrance. A Comprehensive Physical Model for Light Reflection. *ACM Computer Graphics*, 25(4):175–186, 1991.

[49] Paul Heckbert. Color Image Quantization for Frame Buffer Displays. *ACM Computer Graphics*, 16(3):119–127, 1982.

[50] Paul Heckbert and Pat Hanrahan. Beam Tracing Poligonal Objects. *ACM Computer Graphics*, 18(3):119–127, 1984.

[51] Paul S. Heckbert. Survey on Texture Mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, 1986.

[52] Max Herzberger. *Modern Geometrical Optics*. Interscience Publishing, 1958.

[53] W. Hibbard and D. Santek. Visualizating Large Meteorological Data. In G. M. Nielson and B. D. Shriver, editors, *Visualization in Scientific Computing*, pages 147–152. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[54] James T. Kajiya. The Rendering Equation. *ACM Computer Graphics*, 20(4):143–150, 1986.

[55] James T. Kajiya and B. Von Herzen. Ray Tracing Volume Densities. *ACM Computer Graphics*, 18(4):91–102, 1984.

[56] T. L. Kay and D. Greenberg. Transparency for Computer Sinthetized Images. *ACM Computer Graphics (Proc. SIGGRAPH '79)*, 13(3):158–164, 1979.

[57] Donald Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, Massachussetts, 1973.

[58] C. T. Loop and T. de Rose. A Multisided Generalization of Bézier Patches. *ACM Transactions on Graphics*, 8(3):204–234, 1989.

[59] C. T. Loop and T. de Rose. Generalized B-Spline Surfaces of Arbitrary Topologies. *ACM Computer Graphics*, 24(4):347–356, 1990.

[60] W. Lorensen and H. Cline. A High Resolution 3D Surface Construction Algorithm. *ACM Computer Graphics*, 21(4):163–169, 1987.

[61] N. Magnenat-Thalmann and D. Thalmann. *Computer Animation: Theory and Practice*. Springer-Verlag, Tokyo, segunda edition, 1988.

[62] B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman, New York, 1983.

[63] B. Mandelbrot and J. van Ness. Fractional Brownian Motion, fractional noises and applications. *SIAM Review*, 10(4):422–437, 1968.

[64] G. Murch. Physiological Principles for the Effective Use of Color. *IEEE Computer Graphics and Applications*, 4(11):49–54, 1984.

[65] H. D. Murray. *Colour in Theory and Practice*. Chapman-Hall, New York, 1952.

[66] W. Newman and R. Sproull. *Principles of Interactive Computer Graphics*. McGraw-Hill, New York, second edition, 1979.

[67] Gustavo A. Patow and Claudio A. Delrieux. On Low Cost Refraction Models. *Computer Networks and ISDN Systems*, forthcoming:Elsevier Science, 1997.

[68] H.-O. Peitgen and D. Saupe. *The Science of Fractal Images*. Springer-Verlag, New York, 1986.

[69] Bui-Tong Phong. Illumination for Computer-Generated Pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[70] Les Piegl. Interactive Data Interpolation by Rational Bézier Curves. *IEEE Computer Graphics and Applications*, 7(4):45–58, 1987.

[71] Les Piegl. On NURBS: A Survey. *IEEE Computer Graphics and Applications*, 11(1):55–71, 1991.

[72] P. K. Robertson. Visualizing Color Gamuts: a User Interface for the Effective Use of Perceptual Color Spaces in Data Displays. *IEEE Computer Graphics and Applications*, 8(5):50–64, 1988.

[73] P. K. Robertson and J. O'Callaghan. The Generation of Color Sequences for Univariate and Bivariate Mapping. *IEEE Computer Graphics and Applications*, 6(2), 1986.

[74] L. Rosenblum. Scientific Visualization at Research Laboratories. *IEEE Computer*, 22(8):68–100, 1989.

[75] Michael W. Schwarz, William B. Cowan, and John C. Beatty. An Experimental Comparison of RGB, YIQ, LAB, HSV and Opponent Color Models. *ACM Transactions on Graphics*, 6(2):123–158, 1987.

[76] M. Stone. Color Printing for Computer Graphics. Technical Report EDL-88-5, XEROX Palo Alto Research Center, 1988.

[77] M. Stone, W. Cowan, and J. Beatty. Color Gamut Mapping and the Printing of Digital Color Images. Technical Report EDL-88-1, XEROX Palo Alto Research Center, 1988.

[78] I. E. Sutherland, R. F. Sproull, and R. A. Schumaker. A Characterization of Ten Hidden Surface Algorithms. *ACM Computer Surveys*, 6(1):387–441, 1974.

[79] I. E. Sutherland and Hodgman G. W. Reentrant Polygon Clipping. *Communications of the ACM*, 17(1):34–42, 1974.

[80] Ivan Sutherland. Sketchpad: A Man-Machine Grphical Communication System. In *Spring Joint Computer Conference Proceedings*. AFIPS, 1963. Reimpreso en [36], páginas 2–19.

[81] Alain Trouvé. *La Mesure de la Couleur*. AFNOR-CETIM, Paris, 1991.

[82] Aleksej G. Voloboj. The Method of Dynamic Palette Construction in Realistic Visualization Systems. *Computer Graphics Forum*, 12(5):289–296, 1993.

[83] J. E. Warnock. *A hidden-surface algorithm for computer generated half-tone pictures*. PhD thesis, Univ. Utah Comput. Sci Dept., TR 4-15, 1969, NTIS AD-753 671, 1969.

[84] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques*. Addison-Wesley, London, 1992.

[85] T. Whitted. An Improved Illumination Model for Shaded Displays. *Communications of the ACM*, 23(6):343–349, 1980.

[86] Thomas Wright. A Two-Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables. *IEEE Transactions on Computers*, 4(1), 1973. Reimpreso en [36], páginas 284–289.

[87] C. Wylie, G. W. Romney, D. C. Evans, and A. C. Erdahl. Halftone Perspective Drawings by Computer. In *Fall Joint Computer Conference 1967*, pages 44–58, Washington, DC, 1967. AFIPS.