40

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SÍNTESIS DE IMÁGENES EN LA SIMULACIÓN DE SISTEMAS FÍSICOS

T E S I S P R O F E S I O N A L
PARA OBTENER EL TÍTULO DE INGENIERO MECÁNICO ELECTRICISTA EN EL ÁREA ELÉCTRICA-ELECTRÓNICA, MÓDULO DE ELECTRÓNICA DIGITAL.

PRESENTA
MARIO ALFREDO IBARRA CARRILLO

DIRECTOR: DR. VÍCTOR GARCÍA GARDUÑO

MÉXICO, D.F. 2002

PAGINACIÓN DISCONTINUA

Agradecimientos

No es posible vivir aislado y hacer todas las cosas por uno mismo; somos humanos y nos necesitamos.

A mis padres **Mario** y **Teresa** por su paciencia, su apoyo y porque siempre están ahí.

A mi asesor **Víctor García Garduño** por esa paciencia y apoyo inimaginablemente grandes.

A **Ángel Labastida** quien fuera encargado del Laboratorio de Síntesis de Señales por las facilidades prestadas.

A mis padrinos **Javier** y **Magdalena** por esos momentos de apoyo y convivencia familiar.

Al Ing. **Solórzano Palomares** por su ánimo contagioso.

Finalmente, agradezco a todo el **Departamento de Telecomunicaciones** por todas las facilidades prestadas

Índice

Capítulo 6: Subsistema de vídeo: las tarjetas gráficas

Parte I.b. Estado del arte del software

Capítulo 7: Introducción a las primitivas de salida

Capítulo 8: Introducción a las curvas spline

Capítulo 9: Deformación digital de imágenes y texturización

Capítulo 10: Modelos de color

Capítulo 11: Modelos de iluminación

Capítulo 12: Métodos de presentación de superficies

Capítulo 13: superficies de spline

Capítulo 14: Introducción al modelo de iluminación de radiosidad

Parte II. Modelos de movimiento, de superficie y de sombreado

Capítulo 15: El modelo de movimiento

Capítulo 16: El modelo de superficie y texturizado

Parte III. Pruebas, resultados conclusiones

Capítulo 17: Pruebas y resultados

Capítulo 18: Conclusiones

Bibliografía

Introducción

I Antecedentes del tema (contexto)

¿Qué es la síntesis de imágenes?

Antes de proporcionar una definición tal cual receta de cocina, debemos repasar algunos conceptos que pueden ayudarnos a entender lo que significa sintetizar una imagen. Para tal actividad nos apoyaremos en el diccionario, así que:

Imagen: Una imagen es la representación de alguna cosa en pintura, escultura, dibujo, fotografía, etc (Ramón García Pelayo y Gross (1995)).

Síntesis: Método que procede de lo simple a lo compuesto, de los elementos al todo, de la causa a los efectos, del principio a las consecuencias. La síntesis es la operación inversa al análisis. Suma o compendio (Ramón García Pelayo y Gross (1995)).

Sintetizar: Reunir por medio de la síntesis (Ramón García Pelayo y Gross (1995)).

Partiendo de los conceptos anteriores, la síntesis de una imagen es tomar los materiales y herramientas que tenemos a nuestra disposición y entonces construir esa imagen. De acuerdo con esto, la síntesis de imágenes es una actividad tan antigua como el hombre. Por ejemplo, el hombre de Cro-Magnon plasmaba en las paredes de las cavernas escenas de sus actividades importantes. La figura 1.a nos brinda un ejemplo. El ser humano es una criatura que evoluciona, así Miguel Ángel también sintetizó muchas imágenes en pintura y escultura. Un ejemplo que es característico del trabajo de este hombre es la escultura de *el David*. La figura 1. b nos muestra una representación de este trabajo.

Los tiempos cambian y la síntesis de imágenes es ahora asistida por computadora, ya sea que el producto final implique un cartel espectacular, una pieza mecánica, una estatua o una fotografía de una escena salida de la imaginación de su autor.

(a)

(b)

Figura 1
En (a) se muestra una pintura de un caballo en las paredes de la caverna Lacroux en Francia. La representación fue realizada por el hombre de Cro-Magnon. Reproducido de Tom Prideaux (1981). En (b) se muestra la escultura del David la cual fue realizada por Miguel Angel durante el Renacimiento. Reproducido de John R. Hale (1984).

¿Qué es una computadora?

¿Qué es una computadora hoy en día? veamos, una computadora es un sistema diseñado para procesar información, es decir, a la entrada de este sistema colocamos información de algún tipo y mediante algún programa la información se evalúa, se ordena según su importancia (transformación de la información) y se obtiene un resultado que la computadora ha colocado en su salida. Importante es considerar que la computadora realiza de forma rápida y eficiente aquellas tareas que antes se realizaban de forma manual.

Ya tenemos el conocimiento de lo que es una computadora en función del trabajo que realiza pero, ¿qué ocurre con las partes que la forman? Un cierto punto de vista nos dice que los elementos principales de una computadora son el hardware y el software. El

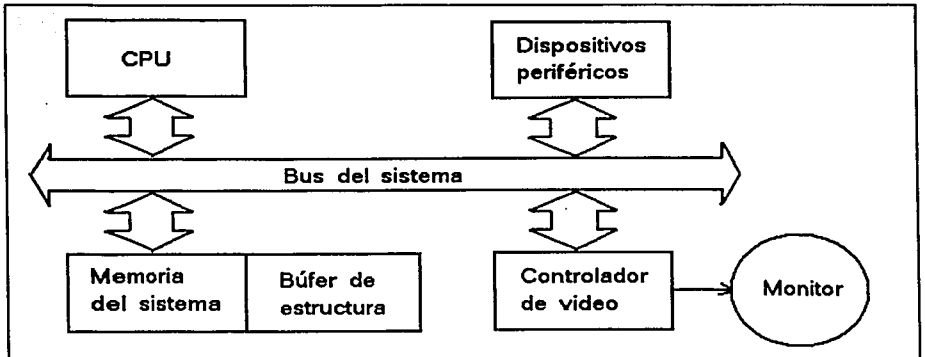


Figura 2
Ejemplo de arquitectura de una computadora casera. Foley, Van Dam, Feiner Hughes, "*Computer Graphics, Principles and Practice*", 2° ed., 1990.

hardware implica a todos los elementos físicos o tangibles de la computadora. Otro cierto punto de vista nos proporciona cinco elementos principales:

- Bus del sistema , el cual es remplazado por dos circuitos que son el controlador de memoria y el controlador de entrada-salida.
- El procesador.
- Memoria.
- Periféricos.
- Subsistema de video.

El software, al contrario que el hardware, son todos aquellos elementos intangibles. Otro punto de vista muy acertado nos dice que son todos los programas que la computadora requiere para su labor de procesar información. Adicionalmente, la modernidad exige que algunos de estos programas, en conjunción con el hardware, sirvan de intermediarios (la interfaz) entre la computadora y la persona interesada en el procesamiento de la información (el usuario).

Arquitectura de una computadora

Según Tanenbaum (1997), para reducir la complejidad del diseño de cualquier sistema, éste se organiza como una serie de bloques interrelacionados. El número de bloques, el nombre, el contenido y la función de cada uno difiere de fabricante a fabricante. Sin embargo, en todo diseño, el propósito de cada bloque es ofrecer ciertos servicios a los otros de modo que no tengan que ocuparse de los detalles de la implementación real de los servicios. En un diseño de bloques, éstos se comunican entre sí, las reglas y convenciones que se siguen para esta conversación se conocen como el **protocolo**. Básicamente un protocolo es un acuerdo entre las partes que se comunican sobre como va a proceder la comunicación. Si se viola el protocolo, la comunicación será más difícil si no imposible.

La figura 2 muestra un diagrama a bloques de una estación de trabajo, observe los bloques y en especial las flechas que indican dos situaciones:

- Los bloques que se comunican.
- El flujo de información.

Entre cada par de bloques que se comunican hay una interfaz. La interfaz define cuales operaciones y servicios ofrece un bloque al otro. Cuando los diseñadores de computadoras deciden que bloques incluir en una computadora y lo que cada uno debe hacer, una de las consideraciones más importantes es definir las interfaces claras entre ellos. Esto implica que cada bloque incluirá una colección de funciones bien conocidas.

Además de minimizar la cantidad de información que se debe pasar entre bloques, las interfaces bien definidas también simplifican el reemplazo de un bloque por otro completamente diferente, pues todo lo que se requiere de la nueva implementación es que ofrezca a su vecino exactamente el mismo conjunto de servicios que ofrecía la implementación vieja.

Un ejemplo del reemplazo de un bloque lo presenciamos cuando cambiamos un disco duro por otro de mayor capacidad. Otro ejemplo es cuando reemplazamos ese dispositivo que conocemos como la tarjeta de video.

Al conjunto de bloques y protocolos lo llamamos la **arquitectura de la computadora**. La especificación debe contener información suficiente para que un implementador pueda construir el hardware para cada bloque. Es necesario que cada bloque obedezca, en forma correcta, el protocolo apropiado. Los detalles de la implementación y de las interfaces no forman parte de la arquitectura, éstas normalmente se proporcionan en un documento aparte y están en función del fabricante.

A veces los fabricantes se reúnen para establecer un estándar con los detalles de la implementación y de las interfaces. Esto beneficia a los fabricantes ya que en corto y en mediano plazo se evitan los monopolios. Al consumidor también le beneficia ya que la competencia tiende a bajar los precios.

Software gráfico

Como ya se mencionó, el software es la otra mitad de la computadora y en sí, es el conjunto de programas que el hardware requiere para procesar la información. Hoy día el software puede clasificarse en:

- Programas de aplicación.
- Lenguajes de programación.
- Controladores de dispositivos.
- API o herramientas generales de programación.
- Sistemas operativos.

De particular interés para la presente investigación son los programas de aplicaciones gráficas y las API orientadas también a la graficación. Una API gráfica ofrece un amplio conjunto de funciones gráficas que se pueden emplear en un lenguaje de programación de alto nivel, como Pascal, C o FORTRAN. El ejemplo más clásico de la API gráfica es OpenGL[1], el cual incluye funciones para crear los componentes más básicos de una imagen: líneas rectas, polígonos, circunferencias y otras figuras. También se incluyen componentes que se construyen a base de superficies: planos, esferas, cilindros, toroides, teteras, etc. OpenGL también incluye funciones para controlar color e intensidad, seleccionar vistas y aplicar transformaciones.

Al contrario que una API, los paquetes de aplicaciones gráficas están diseñados para personas que no son programadores, de modo que los usuarios pueden generar despliegues sin preocuparse por el desarrollo de las operaciones gráficas. Los programas como *Paint* de Microsoft Windows y *AutoCad* de la empresa AutoDesk, son ejemplos de dichos paquetes de aplicaciones.

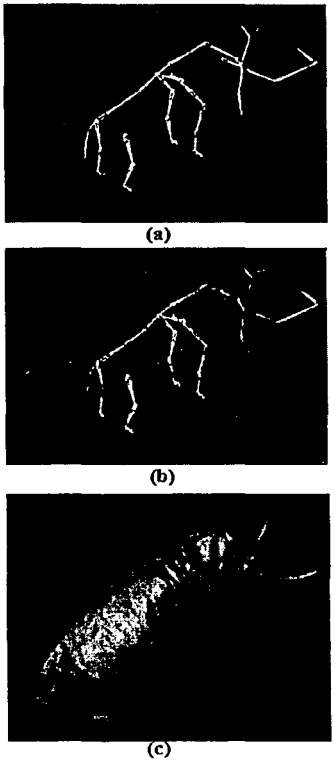


Figura 3
Reconstrucción de un ser vivo de hace 10,000 años.

Además de OpenGL existen otras API que también están orientadas a la síntesis de imágenes:

- IrisGL
- Mesa
- Glide
- Direct3D
- Heidi
- Y otras...

Y aunque no lo crea, todas las API gráficas mencionadas hasta ahora, están diseñadas para trabajar de forma semejante a IrisGL.

Síntesis de imágenes por computadora

Como en la mayoría de las actividades de hoy día, la computadora es la herramienta que siempre debe estar y en el caso del tema que nos abarca es fundamental. Si fuéramos programadores, los conceptos que debiéramos tener en cuenta son:

- Física
- Matemáticas

Si en cambio, fuéramos productores o directores, lo que nos interesaría pudiera ser:

- ¿Cuánto realismo deseamos en nuestro producto?
- ¿Cuánto tiempo debemos esperar?
- ¿Cuánto invertiremos en ello?

No está demás decir que el tiempo y el dinero que invertiremos en una producción está en función del realismo que deseemos tener.

Tal vez, la causa principal de que las computadoras sean la herramienta principal en la síntesis de imágenes, se encuentra en la posibilidad de traer a la realidad una idea.

---

[1] OpenGL es ahora controlado por un consorcio industrial conocido como *OpenGL Architectural Review Board* (ARB).

Finalmente, algunas de las actividades en las que la síntesis de imágenes se ha implicado son:

- Cine
- Medicina
- Ciencia
- Ingeniería
- Aviación
- Geología
- Animación
- Educación
- etc.

Síntesis de imágenes y la industria cinematográfica

El cine no es sólo una industria de millones de dólares y en donde las personas fingen ser algo que no son. El cine tiene un carácter informativo y puede verse en las escenas filmadas y en la tecnología que se usó para crear y filmar esas escenas.

La industria cinematográfica recurre a avances tecnológicos en diversas ciencias para generar o recrear escenas y personajes que sólo existen en nuestra memoria. Es este entonces el carácter informativo del cine, nos comunica las últimas tecnologías y nos muestra personas y escenarios de otros tiempos.

En las figuras 3.a a 3.c vemos una de las técnicas empleadas por científicos paleontólogos para reconstruir de una criatura ya extinta. En 3.a tenemos el esqueleto de un mastodonte y en 3.b se muestra la reconstrucción de su superficie mediante facetas triangulares. Las flechas verdes y azules que se ven en las figuras son indicativos de sistemas de referencia locales a cada hueso del esqueleto. Esto último nos indica que se manejan conceptos de robótica para calcular el movimiento de cada hueso. En 3.c se muestra el resultado final. Note que la superficie se ha dejado algo floja para simular la piel.

Seguimos ahora con la figura 4, podemos observar una serie de técnicas que dan realismo a personajes animados de la película *Final Fantasy*[2]. Algunas de estas técnicas fueron tomadas de la ciencia de la Paleontología que se dedica a la reconstrucción realista de seres vivos. Las figuras 4.a y 4.b nos muestran una técnica empleada que nos permite apreciar los cambios en la forma del cuerpo humano, específicamente un brazo. Observe entonces los músculos se tensan o se aflojan. Las figuras 4.c y 4.d nos muestran una técnica muy empleada en la medicina forense para reconstrucción de rostros de víctimas y que ha sido llevada a la industria del celuloide: la reproducción de una textura que semeje a la piel humana. Normalmente se comienza escaneando una cabeza humana, generalmente en yeso, y de la cual se extrae sólo la superficie que implica al rostro; esto para obtener la forma de lo que será una máscara. Luego se fotografía la piel para generar una textura. La figura 3.c nos muestra la proyección la máscara en un plano en donde un artista agregará o quitará detalles. Finalmente, en la figura 3.d se ve el resultado.

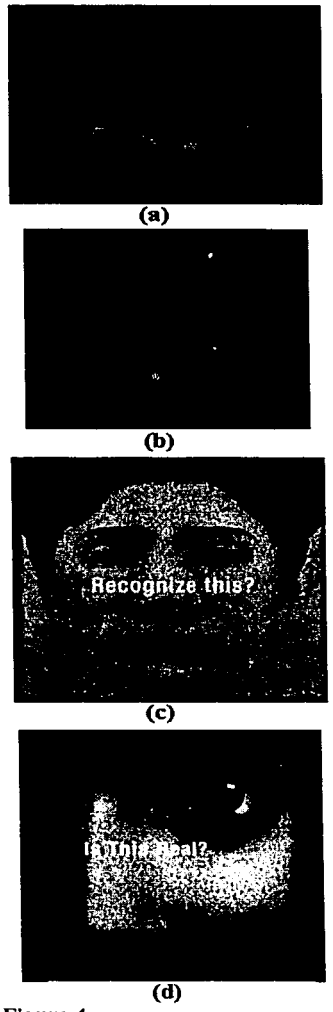


Figura 4
(a) y (b) representan Simulación de músculos. (c) y (d) implican la síntesis piel humana.

[2] *Final Fantasy* es una producción de Square Pictures.

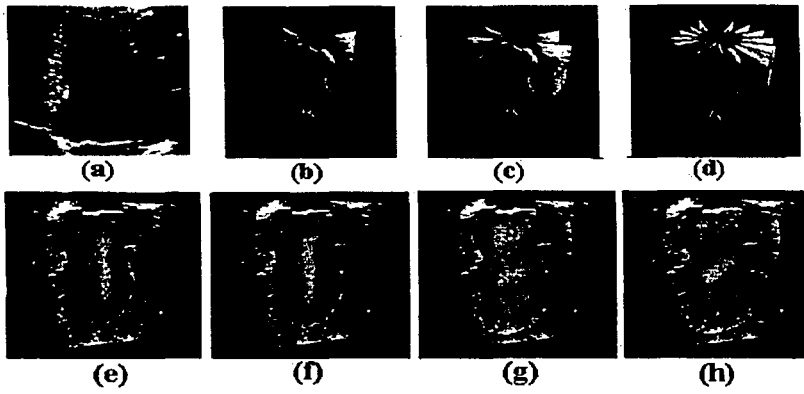


Figura 6
Secuencias de la reconstrucción en imágenes por computadora del ventrículo izquierdo de un corazón humano. Figura obtenidas del grupo EPIDAURE, mismo que se encuentra dentro del tema 3 del proyecto INRIA:

Figura 5
Esta figura muestra la integración de la robótica con la síntesis de imágenes al enseñarnos el control de un instrumento quirúrgico virtual a través de un laparosocopio. Figura obtenida de las páginas de INRIA: http://www.inria.fr/recherche/equipes/projets_theme3.en.html.

Síntesis de imágenes y la medicina

Son las cinco de la tarde, y un estudiante de medicina de octavo semestres se prepara para una laparoscopia hepática, pero... ¿qué es eso? Es una técnica que permite extirpar un fragmento enfermo del riñón de alguien. La técnica implica que el estudiante deberá realizar la operación empleando un laparoscopio, o bien, en términos simples, una guía hueca que lleva un conjunto de fibra ópticas, una línea para una lámpara y un instrumento que puede ser un bisturí, unas tijeras o cualquiera otra cosa. Notará que la técnica no implica abrir al paciente y hacer a un lado los órganos que estorban. También notará que ¿cómo es posible que un estudiante de octavo semestre pueda realizar una operación tan compleja? Pues bien, nuestro paciente voluntario está almacenado en el disco duro de una computadora y el estudiante está a punto de realizar una simulación de esta operación. Tal vez estará de acuerdo en que una computadora que puede simular una operación en un cuerpo humano que implica las siguientes consideraciones:

- Los órganos son cuerpos blandos.
- La cámara del laparoscopio muestra una visión interior del cuerpo.
- Las escenas que vemos no muestran colores básicos sino verdaderas texturas indicando un órgano humano.
- Degradación del estado del paciente y de los órganos.
- Interacción en tiempo real entre el alumno de medicina y el simulador.
- etc.

aún no existe, pero no estamos lejos. La figura 5 nos muestra un ejemplo de integración de la robótica con la medicina.

Veamos ahora la figura 6.a, se trata de un sonograma (mejor conocido como ecografía). Como puede ver, sólo un especialista puede reconocer la forma y movimientos del corazón. Por supuesto, sólo se ve un perfil y es posible que muchos detalles escapen a su

vista. Las figuras 6.b, 6.c y 6.d nos muestran los sonogramas generados por la rotación del instrumento sensor y la pregunta es ¿de qué sirve esto? Las diversas posiciones de los sonogramas nos permiten realizar la reconstrucción 3D del órgano de interés. En consecuencia, es posible tener en la pantalla de un computador una imagen de cualquier órgano rotando para exhibir su superficie completa y así visualizar fácilmente si existen complicaciones. Las figuras 6.e a 6.h nos muestran la técnica básica para realizar la reconstrucción del ventrículo izquierdo del corazón: un globo semejando ser de hule se inflama hasta amoldarse completamente a la forma del ventrículo.

Ahora debemos acordar en que, lo comentado hasta ahora implica algunos problemas que representan un obstáculo a vencer. Uno de los problemas implica que, tanto software como hardware aún no están suficientemente desarrollados como para ofrecer el realismo adecuado en una simulación. En cuanto a la parte del software, INRIA[3] se dedica a resolver este problema implementando algoritmos que se ejecutan en un hardware que consta de varios microprocesadores. En sí, ellos implementan algoritmos que permiten a varios procesadores ejecutar, cada uno, una parte de la simulación. Como consecuencia, los conocimientos que este grupo requiere implican:

- Conocimientos del hardware.
- Conocimientos avanzados de programación.
- Matemáticas.
- Física.

Es una buena idea considerar que, aunque cada procesador realiza su trabajo, el esfuerzo de uno no es independiente del esfuerzo de los otros y es esta una situación implica comunicación entre los distintos procesadores.

Animación

Según Roy Disney, "*la historia de la animación es la historia de una relación dinámica entre el ser humano y la máquina*[4]". La necesidad del ser humano de más y mejores herramientas para lograr animaciones más sofisticadas es la fuerza que impulsa el desarrollo de las herramientas con que cuenta. Por supuesto, esta relación es recíproca ya que al desarrollar las herramientas también se requiere de personas calificadas para manejarlas.

¿Qué es animar? Bueno, animar es dar vida y en un caso particular, es dar movimiento. Podemos tener escenas animadas, lo que implica exhibir una secuencia de n escenas por unidad de tiempo y en donde cada una es ligeramente diferente de la anterior. Otro tipo de animación implica a un modelo de algún cuerpo que cambia de posición o forma en el tiempo debido a la acción de otro: fuerzas.

Del párrafo anterior notamos una palabra importante: *tiempo*. Decimos que existe movimiento dinámico si algún objeto cambia de posición en el tiempo o bien, en un sentido más estricto, si algún cuerpo cambia de posición en el tiempo y debido a la acción de fuerzas.

Ahora ¿en dónde empleamos la animación? Pues bien:

[3] INRIA se divide en grupos de investigación, cada grupo cuenta con un presupuesto que los integrantes de los mismos administran. El grupo en particular que realiza la integración de la medicina con síntesis de imágenes es *Epidaure*.

[4] Palabras de Roy Disney tomadas de su discurso de presentación para el largometraje *Fantasia 2000*.

- Entretenimiento, películas y dibujos animados.
- Publicidad.
- Estudios científicos y de ingeniería.
- Capacitación y educación.
- Arquitectura.
- etc.

Elementos de una escena animada

En una escena podemos distinguir los siguientes elementos:

- Fuentes de luz.
- Cámaras.
- Personajes y objetos del entorno.
- Guión.
- Especificaciones de cuadros clave.
- Generación de cuadros intermedios.

Empíricamente, una fuente de luz la podemos especificar con su posición, con el color de su luz, la intensidad, si la luz que emite tiene forma de haz, de cono o bien, si es omnidireccional como la luz que emite un foco de 60 watts. Una cámara la especificamos con su posición, su orientación y las características de la lente. Los personajes de la escena y los objetos que forman el entorno son más difíciles de especificar. Para estos últimos debemos considerar su forma, su posición, ¿como es la superficie que representan? ¿el objeto es una piedra, un metal o una manzana?

El guión es una descripción de la acción. Define la secuencia de movimiento como un conjunto de eventos básicos que deben ocurrir. De acuerdo con el tipo de animación que se debe producir, el guión podría consistir en un conjunto de borradores o ser una lista de ideas básicas para el movimiento.

Un cuadro clave es un diseño detallado de la escena en un momento determinado de la secuencia de animación. En cada cuadro clave, se sitúa cada objeto y cada fuerza de acuerdo con el tiempo para ese cuadro. Una secuencia animada compleja requiere de al menos dos cuadros clave por cada segundo de animación.

Los cuadros intermedios son los cuadros o escenas que van entre los cuadros clave. El número de cuadros intermedios que se necesitan se determina por los medios que se van a utilizar para desplegar la animación. Una película para una sala de cine requiere de 24 cuadros por segundo, en tanto que su exhibición para la televisión requiere de 30 cuadros por segundo.

La especificación del movimiento

El guión es nuestra guía para determinar las acciones de los personajes y objetos de la escena. Ahora bien, para dar movimiento a estos necesitamos recurrir a diversos conceptos tanto físicos como matemáticos. Así, la especificación del movimiento de un ente implica el como se moverá en escena se moverán. Básicamente tenemos la siguiente clasificación.

- Especificación directa del movimiento.
- Cinemática y Cinemática inversa.
- Dinámica y Dinámica inversa.

Especificación directa de movimiento

Este técnica implica dos formas en las cuales se puede especificar el movimiento de un cuerpo:

- Ecuación matemática
- Matrices de transformación

Mediante una ecuación de movimiento podemos especificar ciertas clases de movimientos, por ejemplo, la figura 7 muestra la trayectoria de una pelota que rebota. El camino que sigue la pelota se aproxima bastante bien con una curva seno moderada y rectificada como la siguiente

$$y(x) = A|sen(\omega x + \theta_0)|e^{-kx}$$

Empleando matrices de transformación, requerimos especificar ángulos de rotación y vectores de traslación para que de esta manera, los vértices que definen a un cuerpo sean afectados.

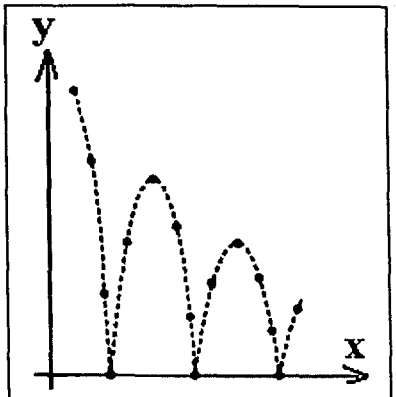


Figura 7
Aproximación del movimiento de una pelota con una función seno moderada. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall , 2° ed., 1995. pg 627

Cinemática y cinemática inversa

Podemos generar una animación al especificar los siguientes parámetros de movimiento: posición, velocidad y aceleración. Note que no se hace referencia a las fuerzas que causan el movimiento. Los casos mas simples corresponden a movimientos rectilíneos y circulares, no obstante, cuando se requiere que un cuerpo recorra una trayectoria cualquiera, se emplean curvas de spline.

Una alternativa es emplear los conceptos de la cinemática inversa. Aquí especificamos las posiciones inicial y final de los objetos en sus correspondientes tiempos. El sistema entonces calcula los parámetros del movimiento.

Dinámica y Dinámica inversa

Por otra parte, las descripciones dinámicas requieren de la especificación de fuerzas que producen las velocidades y aceleraciones. Por lo general, las descripciones del comportamiento del objeto bajo la influencia de las fuerzas se conoce como *modelado con base en las características físicas*. Algunos ejemplos de fuerzas que afectan el movimiento de los objetos incluyen a las fuerzas electromagnéticas, a las de gravitacionales, las fuerzas de fricción y otras fuerzas mecánicas.

Los movimientos de los objetos se obtienen a partir de las ecuaciones de fuerza que describen las leyes físicas, tales como las leyes de movimiento de Newton para los procesos de fricción y gravitación, las ecuaciones de Euler y Navier-Stokes que describen el flujo de líquidos y las ecuaciones de Maxwell para las fuerzas electromagnéticas.

También podemos emplear la dinámica inversa para obtener las fuerzas que actúan sobre un cuerpo, al especificar las posiciones inicial y final de los objetos y el tipo de movimiento.

Las aplicaciones del modelado con base en las características físicas incluyen los sistemas de cuerpos rígidos (como edificios y sillas) y sistemas no rígidos como las telas y materiales plásticos.

Por lo regular, los métodos numéricos se emplean para obtener los parámetros de movimiento en forma incremental a partir de las ecuaciones dinámicas utilizando condiciones iniciales y de frontera.

Benchmark[5]

Un *benchmark* es un punto de referencia a través del cual, algunas cosas son medidas. En una medición, un *bench mark* (dos palabras) es un poste o marca permanente establecida en una elevación conocida que es usada como base para medir la elevación de otros puntos topográficos. En el entorno de las computadoras y de Internet, el término puede tener cualquiera de los siguientes significados:

- Un conjunto de condiciones contra las cuales un producto o sistema es medido. Los laboratorios PC Magazine constantemente examinan y comparan nuevas y diversas computadoras o dispositivos de computadora contra el mismo conjunto de programas de aplicación, interacciones del usuario y otras situaciones. El contexto total contra el cual todos los productos son medidos y comparados es referido como el *benchmark*.
- Un programa que es especialmente diseñado para proveer mediciones acerca de un sistema operativo en particular o bien, de algunas aplicaciones.
- Un producto con el cual los usuarios estén familiarizados y que sirva de punto de comparación contra otros nuevos productos.
- Es un conjunto de criterios o parámetros de desempeño que se espera encontrar en un producto.

Los laboratorios que generan los benchmark en ocasiones no reflejan el uso de un producto en el mundo real, por esta razón, entre los consumidores de productos de computadora, un benchmark es "*una medida imprecisa del desempeño de una computadora*". Los hackers en cambio nos dicen "*en la industria de las computadoras hay tres tipos de mentiras: las mentiras, la mentirotas y los benchmark*".

iComp[6]

Comparar el desempeño de un procesador con otro puede ser complejo ¿cuál sistema operativo empleas? ¿con cuales programas obtienes resultados? Ejecutar una variedad de pruebas de benchmark sobre todos los diferentes procesadores de Intel y otros sistemas implicaría una tarea enorme. Para simplificar este proceso, Intel ha creado el índice iComp.

---

[5] Obtenido de http://www.whatis.com

[6] Puede encontrar más información en http://www.intel.com

El índice iComp (Intel comparative Microporcessor Performance) provee una medición relativamente simple del desempeño de los procesadores Intel y es independiente del sistema operativo y cualquier otro software. Así, el iComp, tiene la intención de ayudar a los usuarios finales a decidir cual es el microprocesador más adecuado a sus necesidades. Por supuesto, el mejor microprocesador para la síntesis de imágenes bien pudiera ser el último que ha salido al mercado.

El índice iComp está basado en cuatro principales aspectos, ya sea que se empleen instrucciones de 16 o 32 bits:

- Desempeño en operaciones con enteros.
- Desempeño en operaciones en punto flotante.
- Graficación.
- Vídeo.

Para concluir

El tema de la síntesis de imágenes es tan amplio que requeriría de la edición de varios volúmenes, es por ello que proporciono únicamente la información que nos sirve de punto de partida para el desarrollo de nuestra investigación.

La síntesis de imágenes o como se le conoce de forma común, *gráficas por computadora*, es una industria muy variada y de muchos millones de dólares. Así que aún no hay una empresa que haya logrado tomar las delantera en todos los aspectos de esta actividad, sin embargo, si hay empresas que se han especializado en cierta áreas. Un ejemplo es la empresa Pixar que se especializa en la síntesis de personajes animados. Su desarrollo ha llegado a tal grado que sus personajes tiene expresiones faciales.

Otras actividades específicas de la síntesis de imágenes son:

- Sobreimposición de imágenes.
- Transformación de imágenes (morphing).
- Restauración de películas.
- etc.

II Propósito del estudio (objetivo)

Piense ahora que estoy dispuesto a iniciar una empresa de juegos de video para computadora (software). Se trata de un negocio que puede ser rentable y para el cual necesito conocer aspectos como:

- La plataforma (hardware) sobre la que voy a realizar mi producto.
- La plataforma que he elegido ¿tiene las características que requiere mi producto para un buen funcionamiento?
- Quiero que mi producto funcione en varias plataformas ¿Qué programas me permiten evitar realizar cambios radicales al cambiar de plataforma?

Estas preguntas me dan la pauta para plantear dos objetivos:

- **Conocer el estado del arte del hardware, es decir, a que nivel de desarrollo ha llegado el diseño de computadoras que son de carácter personal o PC y estaciones de trabajo.**
- **Conocer el estado del arte del software, es decir, verificaremos las técnicas básicas empleadas en la síntesis de imágenes: primitivas, modelos de color, modelos de sombreado, modelos de presentación de superficies y aplicación de texturas.**

Ahora bien, durante nuestra revisión de las técnicas mencionadas en el segundo objetivo, daré a conocer el concepto de OpenGL como una capa entre programador y máquina. Ente las capacidades de esta capa están el dibujar primitivas, aplicar texturas, controlar la iluminación ,etc. En resumen, revisaremos algunos de sus comandos, situación que queda justificada cuando conozca el siguiente objetivo:

- **Introducir al lector en el concepto de que OpenGL puede funcionar como herramienta para representar gráficamente un sistema mecánico.**

Actualmente a una computadora, tal como una PC o una estación de trabajo, se le asocia con el procesamiento de textos y de bases de datos. También se le conoce como juguete cuando ejecutamos un juego de video. Así que nuestro siguiente objetivo es:

- **Demostrar que una computadora, sea PC o estación de trabajo, puede ser una herramienta de carácter científico que nos permita simular el mundo real.**

Tal vez el objetivo anterior pueda parecer abstracto, no obstante trataremos de cumplir con éste simulando una bandera que ondea en el viento. Es de esperar que piense que la computadora debe simular el material de que está hecha la tela, debe simular al viento y debe simular la interacción entre ambos, bandera y viento. A consecuencia, éste será nuestro quinto y principal objetivo:

- **Simular una bandera que ondea en el viento.**

Es probable que, paseando durante algún día de asueto, se ha parado frente al aparador de una tienda de computadoras. Adivinando lo que ha visto, se ha encontrado con que ofrecen al consumidor distintos tipos de computadoras, cada una con capacidades diferentes. Si nosotros deseáramos sintetizar imágenes y aún más, hacer una animación ¿cuál de estas computadoras nos sirve? Pues bien, este párrafo nos da la pauta para el siguiente objetivo:

- **Evaluar el desempeño de la plataforma PC para la síntesis de imágenes**

Se preguntará ¿por qué no se han incluido las estaciones de trabajo? La respuesta es un tanto reveladora. El mundo del PC es el más conocido, está lleno de manuales y cualquiera puede armar una computadora sin garantía de que funcione adecuadamente. El software abunda y se caracteriza por su capa de interfaz amigable. En cambio el mundo de la estación de trabajo queda relegado a los verdaderamente fanáticos. Estos seres místicos no conocen el mundo de las ventanas ya que todo lo manejan a través de una línea de comandos. He advertido que aún estas personas encuentran problemas para manejar sus entornos operativos y que por tanto no son garantía de una ayuda adecuada e incondicional.

A pesar de las limitantes, una estación de trabajo, cuyo nombre es causa de respeto, Silicon Graphics, será revisada en el presente trabajo escrito.

III Límites del estudio (marcos teórico y práctico)

Cuando una computadora tiene la tarea de sintetizar un cuerpo dinámico, es decir, un cuerpo que es movido por la acción de fuerzas, debe resolver al menos los siguientes modelos matemáticos.

- Modelo de movimiento.
- Modelo de superficie.
- Modelo de sombreado.
- Modelo de presentación de superficies.
- Modelo de texturizado.

Estos conceptos corresponden básicamente al software y ya que los programas son la mitad de una computadora también se hace necesario conocer un poco de las arquitecturas en las cuales se programan estos modelos. Como consecuencia, en la primera parte hablaré sobre hardware y software y tendrá un carácter transcriptivo. En un segmento posterior, dentro de un mismo documento, mostraré como se llevará a cabo la investigación.

El modelo de movimiento corresponde al ámbito científico (Matemáticas y Física). El cuerpo al que daremos animación será una bandera. Tal cuerpo será representado por una matriz de puntos ubicados en un mundo tridimensional. El modelo de movimiento que animará de forma realista la bandera, implica un sistema de ecuaciones que deben ser resueltas por la computadora.

La solución de este modelo implica la posición de algunos puntos muestra de la bandera. Fíjese bien, algunos puntos. Esto significa que aún nos falta por determinar las posiciones de todos los demás puntos intermedios. El modelo de superficie nos dirá como calcular esos puntos intermedios. Un método simple implica el uso de parches planos y cuadrilaterales, tan pequeños que no se note que la bandera tiene discontinuidades en su superficie. El otro método implica a las superficies de spline. Este último es un modelo que exige al computador la máxima capacidad de todos sus sistemas. El concepto de splines para representar una bandera no será empleado debido a la complejidad y cantidad de cálculos que implica.

El modelo de superficie determina las posiciones de todos los puntos de la bandera que serán representados en la pantalla, sin embargo aún no se determina que color tendrán. Los modelos de sombreado, presentación de superficies y aplicación de texturas harán visible a la misma.

El propósito de esta investigación no es sólo diseñar la representación de un objeto dinámico en la pantalla de un computador. El otro objetivo que se persigue es evaluar el desempeño de una computadora (hardware y software) cuando se le encomienda la tarea de sintetizar una imagen y un movimiento dinámico. Así que en un segmento de la investigación se considerará una serie de pruebas del tipo benchmark, muy sencillas por cierto. Estas pruebas verificarán la capacidad del equipo para resolver cada uno de los modelos matemáticos.

Índice de materias abarcadas en el presente trabajo escrito

Con el fin de cumplir los requisitos exigidos, presento una lista de las materias abarcadas por la presente investigación:

- Álgebra lineal
- Cálculo diferencial e integral
- Cálculo vectorial
- Computadoras y programación
- Microprocesadores
- Estática, Cinemática, Dinámica
- Óptica
- Dinámica de sistemas físicos
- Electrónica digital

Los cursos de Álgebra lineal, Cálculo diferencial y Cálculo vectorial son las matemáticas básicas requeridas para la presentación de un proyecto que trata de la simulación de un sistema mecánico. Adicionalmente, el curso de Álgebra lineal me apoya con las herramientas que me permiten entender el concepto de transformación geométrica. Este último concepto se requiere para explicar el funcionamiento de una tubería gráfica y entender el proceso de aplicación de mapas de texturas a superficies.

El curso de Computadoras y programación me apoya con la habilidad de programar y adicionalmente me proporciona un cierto marco histórico sobre el desarrollo del hardware de las computadoras.

El curso de Microprocesadores me aporta los conceptos básicos sobre el modo de trabajo de los dispositivos que hacen el mundo hoy día: los microprocesadores. El curso de Electrónica digital permite un conocimiento más profundo de estos circuitos al explicarnos cómo se forman a partir de transistores. Este último curso también es útil para explicar la constitución de los circuitos de memoria.

Los cursos de Estática, Cinemática, Dinámica y Dinámica de sistemas físicos fueron fundamentales para el desarrollo del modelo de movimiento de la bandera ondeante en el viento.

Óptica es un curso que facilita el entendimiento de los modelos de sombreado de superficies. A manera de ejemplo, uno de los conceptos que serán requeridos es el de la reflexión de un **rayo de luz en una superficie**. Tal concepto está en función de la complejidad que se desee implementar. En esta investigación no pretendo abordar un tratamiento electromagnético completo, tan sólo incluiré lo necesario para explicar los modelos de reflexión más básicos.

Temas adicionales

Falta algo ¿con cual software realizaremos la representación de nuestra bandera en la pantalla del ordenador? Aún cuando el desarrollo del software para sintetizar imágenes 3D es muy amplio (IRISGL, OpenGL, MesaGL, Direct3D, Heidi, Glide y demás) no pretendo abarcar cada una de estas posibilidades, en vez de ello me limitaré a una sola herramienta que es seguro puede encontrarse como soporte de cualquier plataforma de computadora. Me refiero a OpenGL.

Lo que no será tratado en definitiva es lo siguiente

- Procesamiento de imágenes
- Realidad virtual
- Desarrollo de aplicaciones herramienta
- Una guía de programación

IV supuestos, expectativas y justificación del tema

El objetivo es terminar un ciclo de mi vida que ya ha durado demasiado y es la carrera de Ingeniero Mecánico Electricista.

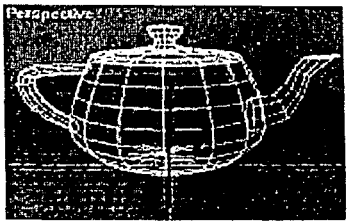
En gran parte de mi vida ha existido la fascinación por los efectos visuales en el cine y anuncios publicitarios. Es ahora que pretendo dar satisfacción y punto final a tal necesidad. El mundo que pretendo estudiar es tan amplio que no me sería posible abarcarlo todo, y aún cuando veo el índice de temas propuesto quedo perplejo. Es en esta vida que tengo la oportunidad de comparar los últimos desarrollos tecnológicos en cuanto a síntesis de imágenes.

Muchos países realizan investigaciones recurriendo a los sistemas de síntesis de imágenes desarrollados en sus propias Universidades. Nuestra propia Facultad, en su división de postgrado también realiza investigaciones en cuanto al análisis y escasamente en la síntesis de las mismas. Esto nos lleva a un cierto rezago, tanto en el desarrollo de software como en el de hardware y así a una dependencia comercial y tecnológica. Por tanto, considero que el presente trabajo es relevante ya que se ha elegido un tema que tiene importancia tanto en Ingeniería Electrónica como en Ingeniería Civil.

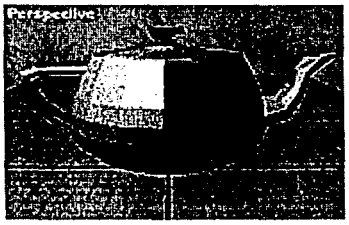
V Aportación a la disciplina y a la escuela

Con la presente investigación pretendo:

- Crear una guía de información para que cualquier interesado en el área de la síntesis de imágenes, pueda elegir el equipo y la configuración de éste que más le convenga.
- Proporcionar conocimientos básicos de la herramienta OpenGL y su poder en la síntesis de imágenes y animaciones.
- Interesar a nuevas generaciones de estudiantes en el uso de este tipo de equipo, para que de esta manera, desarrollen aplicaciones cada vez más sofisticadas.
- Mostrar una técnica sencilla para simular distintos tipos de materiales, en particular tela.
- Restituir a la computadora su carácter de herramienta científica.
- Mostraré que una PC, con la configuración de hardware adecuada, puede servir para la simulación de sistemas físicos y visualización científica, las cuales son actividades más exigentes que la de procesar textos.
- Daré a conocer, de manera teórica, el alto desempeño del equipo Silicon Graphics así como también atribuirle el título de modelo a seguir en cuanto a la construcción de una computadora orientada a la síntesis de imágenes: "de manera teórica" quiere decir en el papel.



(a)



(b)

(c)

Figura 9

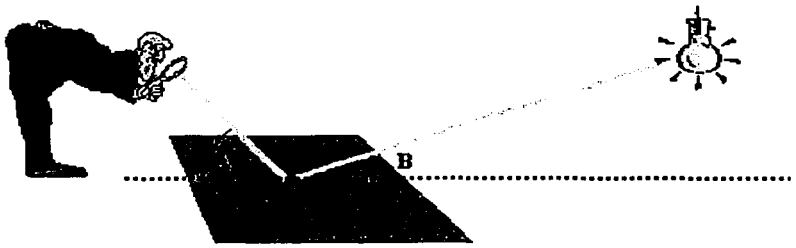


Figura 8
La intensidad y el color del rayo de luz reflejado está en función del color e intensidad del rayo de luz incidente, de las propiedades ópticas de la superficie y de las orientaciones espaciales del observador y la superficie. Figura de http://serdis.dis.ulpgc.es/~atrujill/iga/.

VI diseño de la investigación (metodología)

La presente investigación se dividirá en tres partes:

1. Estado del arte tanto del hardware como del software.
2. Diseño de los modelos de movimiento y de superficie.
3. Pruebas, resultados y conclusiones.

La primera parte

La primera parte es meramente transcriptiva. Por supuesto no se trata sólo de localizar libros y copiar información, hay que adaptarla a la necesidad de la presente. Esta parte se divide a su vez en dos:

- Estado del arte del hardware
- Estado del arte del software.

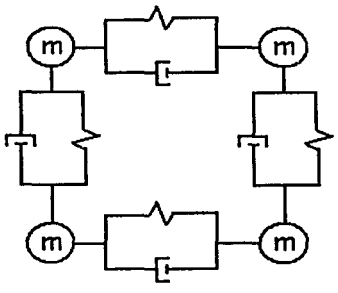


Figura 10

En cuanto al hardware, realizaré un análisis no muy profundo, de algunas partes de la computadora:

- Memoria
- Procesador
- Subsistema gráfico

El análisis también abarca revisar algunas arquitecturas genéricas, como las que podemos encontrar en cualquier libro de organización de computadoras. Finalmente revisaremos algunas implementaciones particulares como la PC, la O_2 de Silicon Graphics y la arquitectura UPA de Sun Microsystems.

En cuanto al software, revisaremos como están diseñadas las rutinas fundamentales que sirven para generar imágenes en la pantalla del monitor:

- Cómo se despliegan líneas, círculos.
- Cómo se despliegan superficies de polígono.
- Curvas y superficies de spline.
- Cómo se simulan efectos de iluminación en una superficie: modelo empíricos y principios de modelos físicos.
- Cómo se aplica una textura a un polígono.

Ahora, a manera de ejemplo, voy a hablar acerca de los modelos de sombreado. Cuando hablamos de un modelo de sombreado (o dicho de otra forma, modelo de iluminación) nos referimos a un método que nos permite calcular la intensidad y el color de la luz reflejada por un punto de una superficie en la dirección del observador, tal como muestra la figura 8. Según esta última figura, la intensidad de la luz reflejada estará en función de los ángulos A, B y en función de las propiedades ópticas del material.

Ahora bien, debemos considerar que en síntesis de imágenes, una superficie curva, por lo regular, se aproxima con polígonos enlazados, lo cual a veces, recordando nuestras clases de geometría, llamamos poliedro. La figura 9.a nos muestra a que me refiero.

Los algoritmos que se encargan de crear superficies de polígono en la pantalla, por lo general aplican un modelo de iluminación para presentar cada punto de la superficie de algún polígono con una sola intensidad o bien, se puede obtener la intensidad en cada punto al emplear un esquema de interpolación. Los cuadros (b) y (c) de la figura 9 nos ilustran estas técnicas.

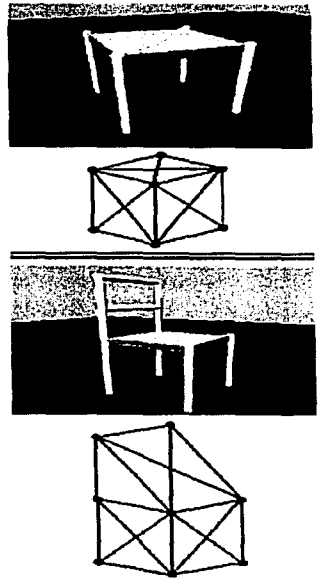


Figura 11

La segunda parte

Este segmento está orientado el diseño de un modelo de movimiento basado en principios físicos que adquirí en la materia de Dinámica de Sistemas Físicos. También se diseñará un tipo de superficie que presente un compromiso aceptable entre realismo y rendimiento del sistema. En resumen, la segunda parte se avoca a desarrollar:

- El sistema físico
- Un modelo de superficie

El sistema físico

Partiremos del concepto de que toda la materia e incluso estructuras como edificios, sillas, gelatinas, pedazos de tela y demás, tienen un comportamiento que puede modelarse con partículas de masa *m* que se enlazan, unas con otras, con resortes y amortiguadores, de tal forma que éstos formen una red. La figura 10 ilustra el concepto al que me refiero.

La figura 11 nos muestra una mesa y una silla cuyas deformaciones pueden modelarse mediante el concepto de partículas. En el caso de la investigación que nos atañe, un trozo de tela, como el de la bandera que pretendo simular puede modelarse también con partículas enlazadas.

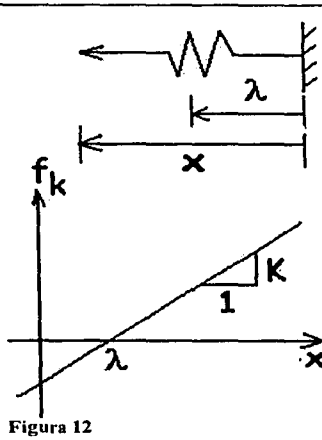


Figura 12

Fk : Fuerza producida por el resorte, se opone al movimiento de éste y es igual a la fuerza externa aplicada (f) con base en la tercera ley de Newton (N).

x : Es el desplazamiento del resorte (m).

k : Es el valor de la constante que tiene el resorte (N/m).

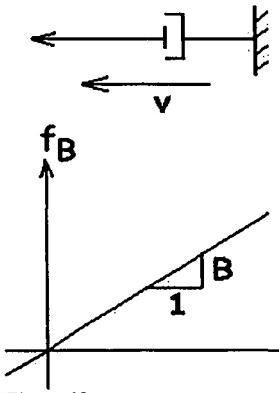


Figura 12

f_B : Fuerza producida por el amortiguador, se opone al movimiento de éste y es igual a la fuerza externa aplicada (f) con base en la tercera ley de Newton (N).

dx/dt: Es la velocidad a la cual se mueve el amortiguador (m/s).

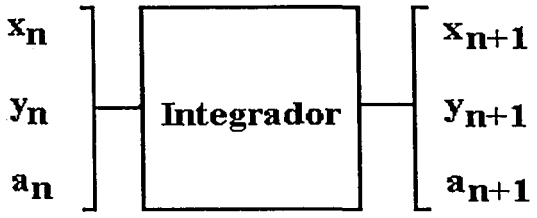
B : Es la constante del amortiguador o coeficiente de fricción viscosa (Ns/m).

Las leyes físicas que gobiernan a un resorte y a un amortiguador las podemos encontrar en cualquier bibliografía de sistemas físicos[7]. Veamos por lo mientras el caso de un resorte. Sabemos que su comportamiento está dado por la **Ley de Hooke**. La figura 12 nos ilustra la relación entre la fuerza y la longitud del resorte desde su longitud de reposo.

En el caso del amortiguador, figura 13, su comportamiento queda claro cuando recordamos la fricción viscosa. Ahora bien, ¿qué es fricción viscosa? Cuando un cuerpo pasa por un líquido o un gas, sufre los efectos de la fricción, o sea, una fuerza que se opone a su movimiento. Debemos aclarar que la fuerza de oposición es proporcional a la velocidad del cuerpo.

El siguiente concepto que debe desarrollarse es el de **partícula de fluido**. Veamos, una forma de modelar un fluido es considerar que se trata de un conjunto de partículas, en donde cada una se caracteriza por su masa y por su **coeficiente de fricción viscosa**. ¡Otra vez la fricción viscosa! Pues sí, echemos un vistazo a la figura 13. La letra *B* es la constante de proporcionalidad entre la fuerza de fricción y la velocidad de un cuerpo en un medio. A esta letra la conocemos también como el **coeficiente de fricción viscosa** y es una forma de indicar cuan viscoso es un fluido.

Toda partícula, tanto de la bandera como del viento, están caracterizadas por tres variables cinemáticas que son la aceleración, la velocidad y la posición. Estas tres variables son lo que en conjunto llamamos el estado del sistema y sus valores serán calculados mediante un método recursivo basado en la técnica de Runge-Kutta para resolver una ecuación diferencial de primer orden.



De acuerdo a la figura de arriba, el estado de un sistema de partículas en tiempo t_n (sólo se muestra el estado de una sola partícula para simplificar la notación) es alimentado al integrador. Este último nos entrega el estado del sistema en tiempo t_{n+1}. Ahora bien, el valor de la aceleración se calcula, en realidad, a partir de la fuerza neta que actúa sobre ella, esto es:

$$a_n = \frac{\sum f_{externas\ en\ n} + \sum f_{internas\ en\ n}}{n}$$

Las fuerzas externas son la fuerza de gravedad y la fuerza que ejerce el viento sobre la bandera. La suma de fuerzas internas está en función de las posiciones y velocidades relativas de las partículas de la bandera.

[7] Francisco Rodríguez Ramírez, "*Dinámica de Sistemas Físicos*", Trillas, 1989.

El modelo de superficie

De las secciones anteriores tenemos una bandera que está representada por partículas. Éstas, sin embargo, representan la estructura física que nos permite modelar el comportamiento de algunos puntos críticos que forman la bandera. En consecuencia, a tales puntos los llamaremos puntos muestra de la bandera o simplemente **puntos muestra**.

Así que ahora contamos con una matriz rectangular de puntos muestra, los cuales están separados una cierta distancia según indica la ecuación de movimiento. Seguramente puede visualizar que aún nos falta por calcular el resto de los puntos que conforman nuestra bandera. El método que nos permite hacer este cálculo es lo que llamamos el **modelo de superficie**.

El modelo de superficie nos permite calcular las posiciones en pantalla de todos los puntos intermedios entre los puntos muestra. La implementación de éste queda a cargo de OpenGL, nosotros sólo debemos elegir entre dos posibles:

- Superficies de spline.
- Superficies poligonales, en particular, facetas triangulares.

Las superficies de spline suelen ser el caso ideal puesto que se requieren relativamente pocos puntos muestra, el resto es interpolado de manera correcta aún cuando la bandera tenga dobleces. Sin embargo, existe un defeco, los cálculos que se requieren pueden hacer que su representación, en una animación, sea lenta.

Las facetas triangulares nos proporcionan una forma rápida de representar cualquier superficie. Tal vez piense que la bandera tendrá una apariencia extraña empleando este tipo de parches, sin embargo, es posible dar a cualquier superficie una apariencia homogénea empleando las técnicas de sombreado y de presentación de superficies.

Modelo de presentación de superficies

Sabemos que un modelo de iluminación calcula la iluminación reflejada en un punto de una superficie. Luego, el modelo de presentación nos indica como calcular la iluminación reflejada en todos los puntos de la misma superficie.

Ahora bien, ¿cómo se aplica el modelo de presentación de superficies? Veamos, he elegido que nuestra superficie se construya con facetas triangulares, así que aplicamos un modelo de iluminación a cada vértice y luego interpolamos la iluminación reflejada en todos los puntos dentro del perímetro del triángulo. El proceso descrito en este párrafo se repite entonces para cada triángulo que forma la superficie de la bandera.

El modelo de presentación de superficies que mostrará la bandera de la Facultad se explicará en los capítulos 9 y 16.

La tercera parte

Esta parte abarca pruebas, resultados y conclusiones. Las pruebas que se plantearán avalúan la capacidad del computador para realizar el trabajo de simulación de la bandera. Estas pruebas consisten en:

- Evaluar el modelo de movimiento.
- Evaluar el modelo de movimiento y el despliegue del perímetro de las primitivas que forman la superficie de la bandera.
- Evaluar el modelo de movimiento y el despliegue de una superficie verde mate.
- Evaluar el modelo de movimiento y el despliegue de una superficie texturizada con la imagen de una bandera.

Son sólo cuatro pruebas, su implementación es muy simple y sin embargo no deben despreciarse: muchos programadores, entre ellos su servidor, pasamos horas y horas frente a nuestras pantallas para desarrollar las herramientas que me permitieron dibujar una bandera ondeando en un viento de partículas.

El entorno

¿Cómo es el entorno en el que se programa la bandera? Éste se puede describir tanto en hardware como en software. El Hardware implica a cualquier PC, alto, así es, el tipo de computadora más despreciada y también la más accesible del mercado. Las pruebas se realizan en equipos mostrados en la tabla siguiente.

Procesador		Subsistema gráfico			Datos generales del sistema	
Tipo	Reloj [MHz]	Modelo	Memoria [Mbytes]	AGP/PCI	Sistema Operativo	Memoria del sistema [Mbytes]
Pentium MMX	200	SiS6326	4	PCI	Windows 98	64
Pentium II	330	S2VIRGE DX/GX	4	PCI	Windows 98	64
Pentium III	650	SiS 6326	4	PCI	Windows 2000	512
Pentium III	C50	SiS 6326	8	AGP 2X	Windows 2000	512
Pentium III	650	nVidia TNT2 M64	32	AGP 2X	Windows 2000	512

Tabla 1 Datos de las computadoras que serán evaluadas con nuestro modelo de bandera ondeando en el viento.

La selección del hardware mostrado en la tabla 1 puede parecer extraña, sin embargo, como explicaré a continuación, tiene bastante lógica. Los tres primeros equipos representan algunos peldaños del desarrollo histórico de los procesadores Intel: Pentium MMX, Pentium II y Pentium III[8]. Estos tres circuitos servirán para verificar si los cambios tecnológicos son un avance o un retroceso en cuanto a la tarea de sinterizar imágenes. Así que la pregunta a plantear es:

Los cambios tecnológicos en el procesador principal y placa base de una computadora ¿Representan un avance o un retroceso para la síntesis de imágenes?

[8] Pentium MMX, Pentium II y Pentium III son marcas registradas de Intel Corp.

Regresemos otra vez a la tabla 1 y verifiquemos que los últimos tres equipos son en realidad el mismo. Lo que cambia en cada caso es el subsistema gráfico. Así que ahora tenemos una segunda pregunta por responder:

Los cambios tecnológicos en el subsistema gráfico (procesador de video y circuitos de memoria) ¿Representan un avance o un retroceso para la síntesis de imágenes?

En cuanto al software, la misma tabla 1 nos permite ver que he recurrido al conocido Windows 98[9], el sistema operativo más despreciado y también el más necesitado en su tiempo. Puede notar que también recurrí a un sistema Windows 2000[10], no tan despreciado. Ambos sistemas son sumamente amigables, a diferencia de UNIX. Además, hay una amplia cantidad de información tanto en libros como en la Internet sobre Windows y cualquier herramienta desarrollada en él.

No sólo se requiere de un sistema operativo, también se requiere de un lenguaje de programación y el elegido fue *Delphi*[11]. La razón de esta elección es la simpleza de su lenguaje nativo: Pascal, que en comparación con el lenguaje C es mucho más amigable. El entorno de Delphi también hace que el usuario se olvide de la administración de las ventanas permitiéndole así dedicarse a la solución de algún problema.

El último elemento de Software que falta considerar es OpenGL[12]. OpenGL, como ya sabemos es una interfaz gráfica muy poderosa para equipo gráfico que permite a los programadores producir imágenes en color de alta calidad a partir de objetos 2D y 3D. Los componentes básicos de OpenGL son:

GL	Que es un conjunto de 120 comandos básicos para construir cuerpos complejos a partir de primitivas.
GLU	Extiende los comandos para agregar la definición de cámaras, vistas en perspectiva y objetos geométricos comunes como cilindros, esferas, etc.
GLUT	A diferencia de los anteriores, proporciona comandos para generar una interfaz con el usuario. Esta interfaz implica ventanas y control de eventos desde el teclado o el ratón de manera simple y así, se libera al programador de la compleja tarea de generar y administrar ventanas.

Tal vez no parezca claro, pero tenemos otra pregunta que hacer. Veamos, tenemos los siguientes modelos que juntos nos dan una bandera ondeante:

- Modelo de movimiento.
- Modelo de superficie.
- Dos modelos de sombreado, uno es verde y otro representa los colores de la bandera.

El modelo de movimiento tiene que ver con el cálculo de las posiciones de algunos untos muestra de la bandera. El modelo de superficie calcula la posición de puntos muestra adicionales. A este respecto, por el modelo de superficie elegido ya no hay necesidad de realizar estos cálculos. Luego, los puntos muestra corresponden a los vértices de polígonos triangulares, mismos que deben ser coloreados por algún modelo de presentación de superficies.

---

[9] Windows 98 es una marca registrada de Microsoft Corp.

[10] Windows 2000 es una marca registrada de Microsoft Corp.

[11] Delphi es una marca registrada de Borland International Inc.

[12] OpenGL es un producto controlado por el Consorcio *OpenGL Architectural Review Board* (ARB).

Estos tres modelos se combinan en cuatro formas distintas, en un programa de aplicación, para evaluar el desempeño de la computadora.

- Modelo de movimiento.
- Modelo superficie de alambre, que implica al modelo de movimiento, a un modelo de superficie y a un modelo de presentación de superficies que dibuja el perímetro de los polígonos que definen la bandera.
- Modelo de superficie verde mate, que también implica al modelo de movimiento, a un modelo de superficie y a un modelo de presentación de superficies que dibuja triángulos verdes bajo los conceptos de luz y sombrea.
- Modelo de superficie de bandera, que implica al modelo de movimiento, al modelo de superficie y a un modelo de presentación de superficies que nos da la textura de la bandera de nuestra Facultad bajo los conceptos de luz y sombra.

Observe que tenemos cuatro aplicaciones que dibujan una bandera en la pantalla del ordenador, excepto por la primera aplicación. Entonces la pregunta obligada es:

¿Cómo se ve afectado el rendimiento del sistema por cada una de las cuatro aplicaciones arriba mencionada?

Sobre los resultados

Para responder a esta preguntas, debemos realizar diversas pruebas a distintas configuraciones de equipo, la información recopilada se organiza en las siguientes tablas:

Procesador		Subsistema gráfico	Resultados [pfs]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS6326	
Pentium II	330	S2VIRGE DX/GX	
Pentium III	650	SiS 6326	
Pentium III	650	SiS 6326	
Pentium III	650	nVidia TNT2 M64	

Tabla 2. Resultados de las pruebas al modelo de movimiento

Procesador		Subsistema gráfico	Resultados [pfs]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS6326	
Pentium II	330	S2VIRGE DX/GX	
Pentium III	650	SiS 6326	
Pentium III	650	SiS 6326	
Pentium III	650	nVidia TNT2 M64	

Tabla 3. Resultados de las pruebas al modelo de superficie de alambre

Procesador		Subsistema gráfico	Resultados [pfs]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS6326	
Pentium II	330	S2VIRGE DX/GX	
Pentium III	650	SiS 6326	
Pentium III	650	SiS 6326	
Pentium III	650	nVidia TNT2 M64	

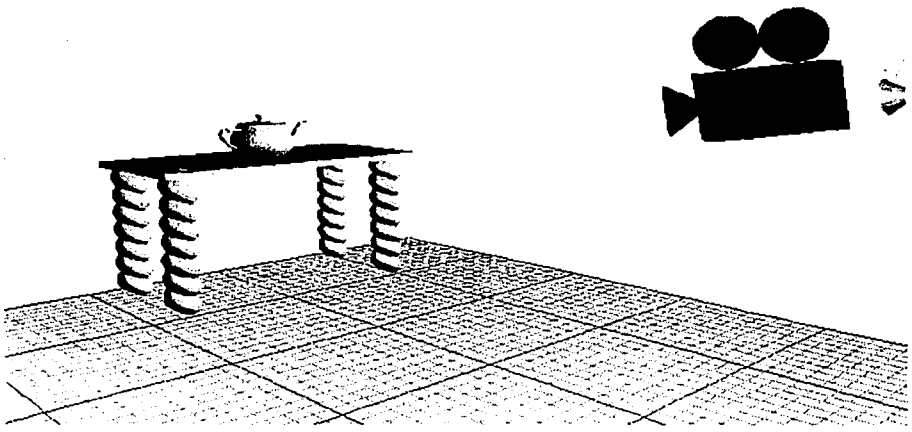
Tabla 4. Resultados de las pruebas al modelo de superficie verde mate.

Procesador		Subsistema gráfico	Resultados [pfs]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS6326	
Pentium II	330	S2VIRGE DX/GX	
Pentium III	650	SiS 6326	
Pentium III	650	SiS 6326	
Pentium III	650	nVidia TNT2 M64	

Tabla 5. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.

Como es de esperar, el caso de la PC es predecible, siempre las últimas y más novedosas tecnologías implican un mayor rendimiento. Así, se espera que:

- Un CPU y una placa base de última generación sean más eficientes.
- Un subsistema gráfico basado en bus AGP proporcione mejores prestaciones que uno basado en bus PCI.
- Una visualización científica más detallada sobre los resultados de una simulación implica un mayor esfuerzo por parte de la computadora.
- Respecto del punto anterior, un subsistema gráfico más avanzado y un software de simulación más eficiente ayudan a que el rendimiento del sistema aumente.



Capítulo 1 Tubería gráfica

Cuando diseñamos un objeto, tal como una casa con muebles, primero diseñamos cada elemento mediante un conjunto de primitivas. Las primitivas son un conjunto (una biblioteca) de puntos, líneas y polígonos que unimos para formar las superficies de objetos complejos. Una vez definidos los cuerpos geométricos y sus posiciones en la escena, suministramos toda esta información a la computadora para que forme la imagen.

La tubería gráfica es un modelo lógico que define un conjunto de etapas por las que pasan las primitivas para convertirse en una imagen. A tal modelo lo llamamos también **tubería de representación**. Es necesario diferenciar entre el modelo lógico y la implementación en hardware del mismo, por lo cual incluyo información adicional sobre la arquitectura básica de una computadora orientada a la síntesis de imágenes y que llamamos comúnmente el **sistema de despliegue**.

Una tubería es un modelo muy complejo, por lo que se hace necesario un análisis del mismo; me refiero a un estudio por partes. Comenzamos tal empresa en las transformaciones por las que pasa un punto coordenado dado por el programador, hasta convertirse en un punto coordenado en un sistema de despliegue tal como un monitor, una impresora, etc.

Sabemos que los seres humanos tenemos un campo de visión limitado por nuestras propias características y defectos. Con facilidad podemos distinguir objetos desde una distancia de 15 centímetros hasta varios kilómetros y además en función del tamaño del objeto. Este espacio que captamos tiene su equivalente en síntesis de imágenes y se llama **volumen de vista**. Así entonces, es el programador quien define el volumen de vista y a partir de éste, el computador determina que primitivas no serán procesadas y cuales continuarán por las etapas de la tubería.

Finalmente queda por mostrar la integración de los conceptos anteriores en una tubería gráfica completa. Una tubería gráfica completa siempre se divide en dos subsistemas, el frontal y el posterior. El subsistema frontal se subdivide a su vez en dos segmentos. Uno de esos segmentos es el subsistema geométrico y representa las transformaciones de un punto coordenado dado por el programador.

1-1 La tubería gráfica

Cuando deseamos sintetizar una imagen a través de un lenguaje de programación como el *C*, creamos una lista de **primitivas** (puntos, líneas, polígonos, etc.) y dejamos que la computadora se encargue de exhibir nuestra escena la pantalla. Por supuesto, nuestro programa pasa por los procesos de compilación, ligado, carga y ejecución; sin embargo la lista de primitivas que conforman nuestra escena debe pasar por otro tipo de procesos antes de ser exhibidas en la pantalla del ordenador. El conjunto de procesos al que son sometidas las primitivas se les conoce como la **tubería gráfica o tubería de representación**. En general, se trata de un modelo un lógico que representa todos los cálculos necesarios para representar una primitiva, o varias de ellas, en la pantalla de la computadora. La tubería gráfica puede implementarse ya sea por software o por hardware o bien, ser un híbrido entre ambos, software y hardware.

La tubería gráfica, para su estudio, se divide en las siguientes partes: subsistema frontal que a su vez se divide en modelo de despliegue y subsistema geométrico. La segunda parte es el subsistema posterior.

1-2 Sistema gráfico de despliegue por rastreo

Cuando hablamos del sistema gráfico de despliegue por rastreo, entendemos que se trata de una computadora completa que lleva a cabo todas las etapas de la tubería gráfica. En general, un sistema de despliegue consta básicamente de seis partes:

- CPU
- El bus del sistema
- La memoria principal
- El búfer de estructura
- El controlador de vídeo
- El monitor CRT

La figura 1-1 nos muestra una arquitectura en la que se abarcan los elementos ya mencionados. Por supuesto es un ejemplo fundamental para ilustrar y no representa el estado actual de la tecnología. El CPU se encarga de realizar todos los cálculos correspondientes para el modelado de figuras, por ejemplo transformaciones geométricas como rotaciones, escalas, deformaciones, etc. También se encarga de dar color a las superficies de una figura. El resultado de todo se almacena en una memoria llamada **búfer de estructura**, que es un segmento de la memoria del sistema reservado para este fin. El controlador de vídeo lee los valores de píxel del búfer de estructura en el mismo orden en que se dibujan las líneas de rastreo en el CRT. La etapa final es convertir estos datos de píxel en señales analógicas que pueda entender el monitor.

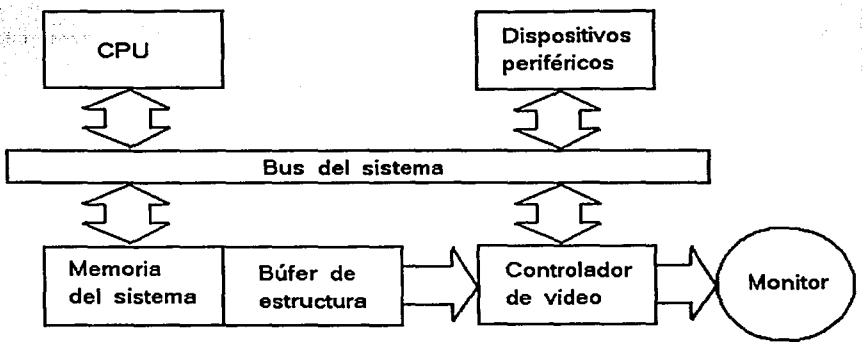


Figura 1-1
Arquitectura fundamental de una computadora. Los elementos fundamentales de la misma son el CPU, el bus del sistema, la memoria principal, el búfer de estructura, el controlador de video y el monitor. Foley, Van Dam, Feinderm Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, 1995, pg. 179.

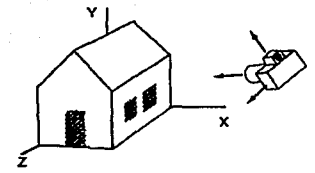


Figura 1-2
La fotografía de una escena implica la selección de la posición y la orientación de la cámara. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2º ed., 1995, pg. 454.

1-3 Subsistema geométrico

En cierto modo, los pasos para la generación por computadora de una vista de alguna escena tridimensional son análogos a los pasos necesarios para tomar una fotografía; véase entonces la figura 1-2. Para tomar una fotografía instantánea, primero debemos colocar la cámara en un punto particular en el espacio. Así, es preciso decidir la orientación de la cámara ¿en cuál dirección apuntamos y como debemos girarla alrededor de la línea de visión a fin de establecer la dirección hacia arriba para la imagen? Por último, activamos el disparador, se corta la escena al tamaño de la apertura del diafragma y la luz de las superficies visibles se proyecta sobre la película plana de la cámara. En la síntesis de imágenes, todo este proceso tiene una representación matemática. Cada punto de la escena debe pasar por un proceso matemático a fin de lograr su representación en la pantalla del ordenador o cualquier otro dispositivo de despliegue. La figura 1-3 ilustra al subsistema geométrico con algunos de los pasos por los cuales se afecta un punto coordenado. En general, estos pasos se emplean para proyectar todos los objetos en una escena en la pantalla.

Coordenadas de modelado

Se puede construir la forma de objetos individuales, como árboles o muebles, en un sistema de coordenadas que son únicas para el objeto. Este sistema de coordenadas puede ser cartesiano, cilíndrico, esférico, hiperbólico, etc. Generalmente es necesario convertir los puntos coordenados del objeto a un sistema de coordenadas cartesianas antes de poder capturarlos en un programa para el computador. Al sistema de coordenadas de modelado también se le conoce como **sistema de coordenadas locales o sistema de coordenadas maestras**. A manera de ejemplo, la figura 1-4 nos muestra tres figuras modeladas: una tetera, una caja que simula una mesa y una pata de la mesa, cada una, en un sistema de coordenadas cartesianas y que servirán para construir nuestra escena de una tetera sobre una mesa.

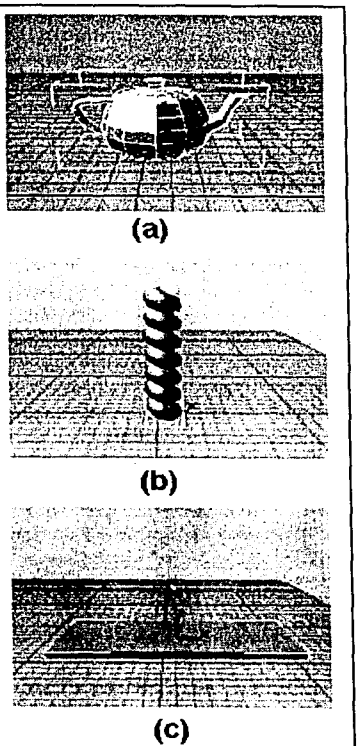


Figura 1-4
Tres objetos diseñados, cada uno en su propio sistema de coordenadas: una tetera, una pata y una mesa. Figuras cortesía de Autodesk.

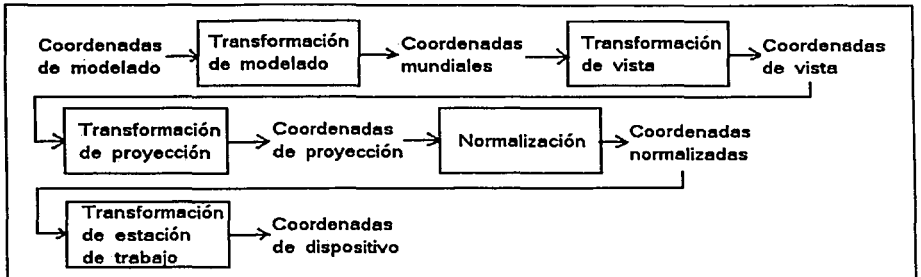


Figura 1-3
Una tubería geométrica representa los cálculos por los que pasa un punto coordenado en un sistema de coordenadas mundiales (mundo tridimensional el que se ubican los objetos en una escena) a un sistema de coordenadas de dispositivo, esto es, un mundo bidimensional correspondiente a un área en un papel o en la pantalla del monitor, etc. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2ª ed., 1995, pg. 455.

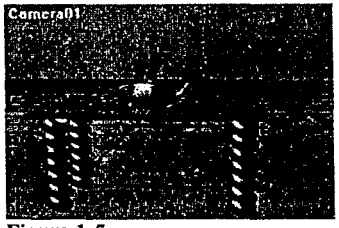


Figura 1-5
Escena en coordenadas mundiales construida a partir de tres objetos distintos. Figura cortesía de Autodesk.

Coordenadas mundiales

Una vez especificadas las formas de objetos individuales, podemos modificar su orientación, deformarlos y colocarlos en posiciones adecuadas en una escena al utilizar una estructura de referencia llamada sistema de coordenadas mundiales. La figura 1-5 nos muestra la escena ya formada a partir de una tetera, una pata y la superficie de una mesa.

Coordenadas de vista

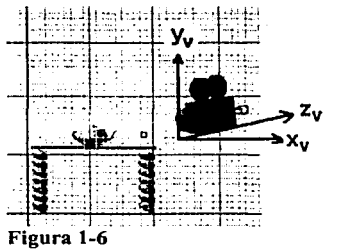


Figura 1-6
Esta escena nos muestra un observador, la cámara, definido dentro de un sistema de coordenadas que llamaremos de vista. El sistema de coordenadas de vista define la posición y orientación del observador. Figura cortesía de Autodesk.

Un sistema de coordenadas de vista se utiliza en paquetes de gráficas como una referencia para especificar la posición y orientación de visión de un observador. También nos sirve para especificar la posición del plano de proyección, que podemos considerar como análogo al plano de la película de la cámara. El sistema de coordenadas de vista es un sistema de referencia definido dentro del sistema de coordenadas mundiales, la figura 1-6 nos ilustra una escena tridimensional siendo filmada por una cámara que nos indica la posición y orientación de algún observador. Podemos notar que el sistema de coordenadas de vista tiene un origen o punto de referencia de vista y tres ejes perpendiculares y linealmente independientes entre sí que lo definen. El **plano de proyección** normalmente es perpendicular al **eje de vista** o **eje Z** del sistema de coordenadas de vista.

Coordenadas de proyección

Una vez definidos los objetos tridimensionales en coordenadas de vista, se pueden proyectar a un plano de proyección. Este plano de proyección es análogo a la película plana de la cámara. En resumen, se van a convertir coordenadas 3D a 2D.

Es común definir una pequeña ventana dentro de plano de proyección. Esta ventana determina un volumen de vista a partir de la cual algunos objetos serán visibles. A esta ventana se le conoce como la **ventana de proyección** o **ventana de visión**.

Existen dos métodos básicos de proyección, los cuales muestro en la figura 1-7. En la proyección paralela se transforman las posiciones de coordenadas en el plano de visión a

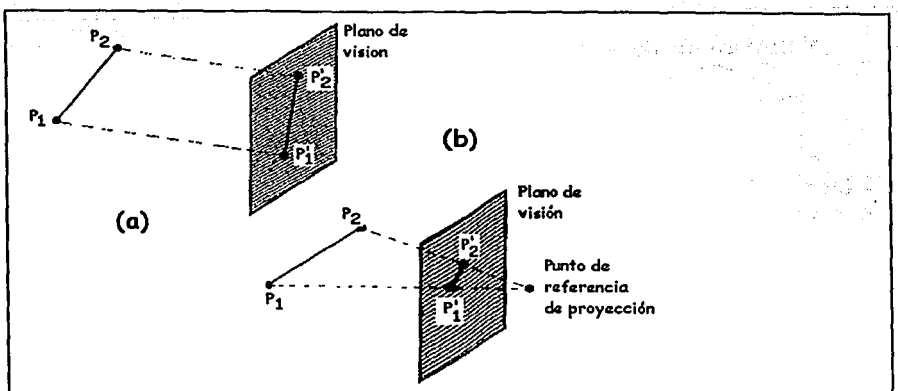


Figura 1-7
En (a) se muestra la proyección paralela de un objeto hacia el plano de visión. En (b) se muestra la proyección en perspectiva de un objeto hacia el plano de visión. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 1995, pg. 460-461.

lo largo de líneas paralelas, como se muestra en el ejemplo de la figura 1-7.a. Para una proyección en perspectiva, figura 1-7.b, se transforman las posiciones de los objetos en el plano de visión a lo largo de líneas que convergen en un punto que se denomina **punto de referencia** o **centro de proyección**. La vista que se proyecta de un objeto se determina al calcular la intersección de estas líneas con el plano de proyección. La figura 1-8 nos muestra entonces la proyección de las escenas sobre el plano de visión.

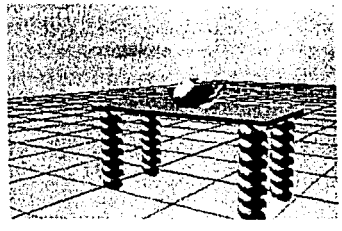


Figura 1-8
Escena captada por la cámara, el observador, y proyectada en el plano de proyección. Figura cortesía de Autodesk.

Coordenadas normalizadas

Para hacer que el sistema de gráficas sea independiente de los diversos dispositivos de graficación que se pueden emplear en una estación de trabajo, las coordenadas de proyección se expresan en coordenadas normalizadas, esto es en el rango de 0 a 1, antes de realizar la conversión final para especificar las coordenadas del dispositivo.

Coordenadas de dispositivo

Por lo general, en un dispositivo de despliegue de imágenes se define un **puerto de vista**, o sea, un área en la cual será desplegada una imagen y que generalmente es rectangular. Dentro de esta área se define un sistema de coordenadas que llamaremos **sistema de coordenadas de dispositivo.**

1-4 Volúmenes de vista y proyecciones

Ya sea que empleamos una cámara o nuestros ojos, tenemos un limitado campo de visión. En la síntesis de imágenes se trata de simular esta situación mediante analogías a la cámara fotográfica. Veamos, el tipo de lente que utiliza nuestra cámara es un factor que determina el porcentaje de la escena que se capta en la película, o sea, una lente gran angular capta una parte más amplia de la escena que una lente que regular.

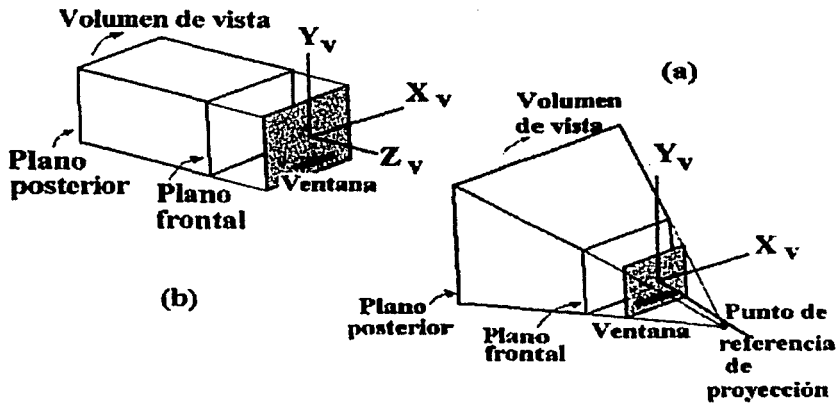


Figura 1-9
Volúmenes de vista que se limitan con los planos frontal y posterior y por los planos superior, inferior y lateral. Los planos frontal y posterior son paralelos al plano de visión en las posiciones z_{front} y z_{back} a lo largo del eje z_v. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 1995, pg. 471.

En analogía con el tamaño de la lente de una cámara, el porcentaje de la escena que se proyecta en el plano de visión se determina al especificar una ventanita que llamaremos **ventana de proyección** o **ventana de vista**. La figura 1-9 nos muestra la colocación de la ventana de visión dentro del sistema de coordenadas de vista para los dos métodos de proyección que ya mencioné en la sección anterior: proyección paralela y en perspectiva.

Ahora bien, dadas las especificaciones de la ventana de vista y de un modelo de proyección, podemos incluir los volúmenes de vista para los dos tipos de proyección ya mencionados. En la figura 1-9 se indica que para la proyección paralela, el volumen de vista es un paralelepípedo y para la proyección en perspectiva es una pirámide cuyo ápice está en el centro de proyección.

Podemos tener un volumen de vista finito al limitar la extensión del volumen en la dirección de vista. Ésta operación se realiza al especificar un plano frontal o cercano y un plano posterior o lejano, tal como se muestra en la misma figura.

1-5 La tubería gráfica estándar

La figura 1-10 nos muestra un ejemplo de una tubería gráfica adaptada a nuestras necesidades. En esta tubería distinguimos tres partes:

- La parte frontal.
- La parte posterior.
- Subsistema geométrico.

La parte frontal se encarga de suministro de los comandos que definen una escena y de su tratamiento geométrico segundo he descrito arriba. El resultado de esta etapa se almacena

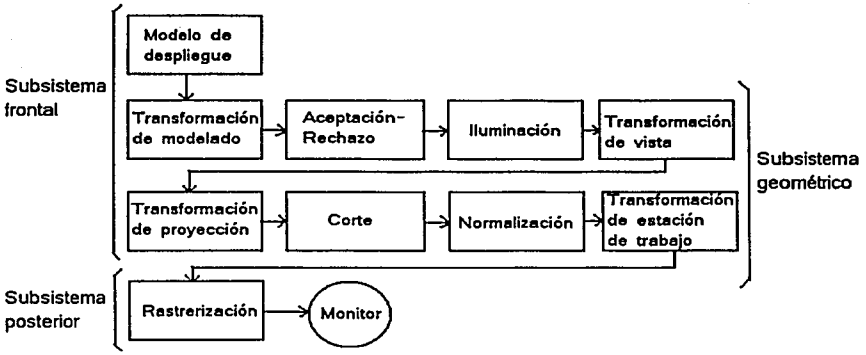


Figura 1-10
Ejemplo de tubería gráfica. Este esquema no representa el estado actual de la tecnología, no obstante puede ser considerado como un punto de partida para el diseño de otras tuberías. El estado actual de la tecnología permite al programador definir su propia tubería de representación.

en el búfer de estructura. La parte posterior se encarga de mostrar el contenido del búfer de estructura en el dispositivo de salida, la cual pudiera ser un monitor CRT o una impresora. En un capítulo futuro hablaré de la implementación hardware de estas etapas.

El modelo de despliegue

El modelo de despliegue se refiere a la forma en que las primitivas y algunos comandos son alimentados a la tubería gráfica. Hay dos tipos de modos, el modo inmediato y el modo lista. Al modo lista se le conoce en la bibliografía como ***retained mode.***

En el **modo inmediato,** el usuario puede construir una escena enviando cada primitiva o comando uno por uno y en cualquier momento. En el **modo lista,** como su nombre lo indica, se ha creado una lista con las primitivas y comandos que conforman la escena, así que la tubería procesa los comandos de la misma tal como se hace en una línea de montaje de autos. El modo lista tiene una peculiaridad, no es posible modificar las primitivas o comandos que contiene. En consecuencia si se desea modificar una figura, una nueva lista debe construirse con la figura ya modificada.

El modo lista también se caracteriza en que, la lista de primitivas puede ser almacenada en alguna memoria que sea local al procesador encargado de la tubería gráfica. De esta forma se libera algo de la carga de trabajo del procesador principal.

Aceptación-rechazo

Como ya se mencionó, podemos limitar la extensión del volumen de vista, de tal forma que podemos escoger las partes de una escena que deseamos ver y excluir objetos que se encuentran enfrente o atrás de la zona que nos interesa.

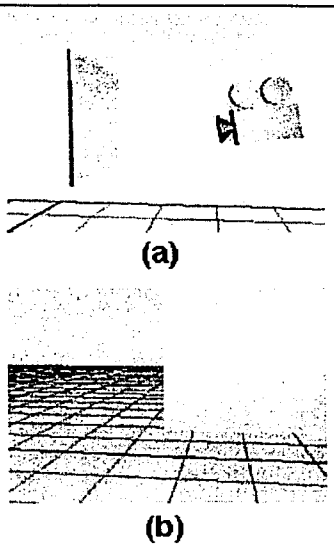


Figura 1-11
Imagen captada por la cámara. En (a) la cámara apunta a una esquina de un prisma rectangular. En (b) la porción del prisma que se ve en el plano de proyección es un cuadrilateral. Figuras cortesía de Autodesk.

Corte

De la etapa de aceptación-rechazo tenemos un conjunto reducido de primitivas que caben dentro de nuestro volumen de vista, sin embargo, algunas de esas primitivas no caben completamente, es decir, se pueden visualizar fragmentos de las mismas. El caso de una línea es simple ya que sólo se recorta su longitud, sin embargo, el caso de un polígono es diferente: el proceso de corte que se le aplica a una primitiva poligonal resulta en otro polígono. La figura 1-11 ilustra lo comentado. En 1-11.a tenemos un observador filmando un prisma rectangular y en 1-11.b tenemos una porción captada por la cámara. Notará que de imagen que debe proyectarse corresponde a un cuadrilateral plano.

Iluminación

Los paquetes más básicos cuentan con tres modelos de iluminación: ambiente, difusa y especular. Estos tres modelos definen las propiedades ópticas del material que representará la primitiva, así como las propiedades ópticas de la luz que ilumina el material. Las propiedades ambiente y difusa definen el color de la primitiva y de la luz que la ilumina. La propiedad especular nos permite definir aquellas manchitas brillantes que vemos en los objetos.

Para comprender esos conceptos requerimos de una explicación larga y compleja, por ello reservamos tal explicación para el futuro. Por ahora, si podemos decir que esta etapa se limita a aplicar los modelos de iluminación ambiente y difusa a cada vértice de cada primitiva, el modelo especular se aplica en una etapa posterior conocido como rastrerización.

Rastrerización

Esta etapa convierte las primitivas en valores de píxel, y generalmente almacena estos valores en un segmento de memoria conocido como el **búfer de estructura**[1] o (*frame buffer*). La rastrerización consiste de tres tareas, la conversión por rastreo (*scan conversión*), la determinación de superficies visibles y sombreado.

¿Qué implica la determinación de superficies visibles? Pues bien, antes de trazar un cuerpo geométrico, se determina cuáles de sus caras son visibles y cuáles están ocultas a la vista y que por tanto no serán trazadas.

Los procesos de conversión por rastreo y sombreado se unen en uno solo, así que una primitiva se traza de la misma forma en que se dibuja una imagen en la pantalla de un CRT, por líneas de barrido o más popularmente, líneas de rastreo. Ahora vamos a verlo más de cerca, el proceso determina la posición de cada píxel y también el color de los mismos. La figura 1-12, aunque no del todo relacionada nos ilustra el cómo se realiza el trazado de un triángulo mediante líneas de rastreo a la pantalla del ordenador.

El monitor

El tipo más común de monitor gráfico es el CRT de barrido, con base de la tecnología de la televisión. Un monitor de barrido monocromático opera dirigiendo un haz de electrones a lo largo de una trayectoria fija de líneas horizontales o trama, que comienza en la esquina superior izquierda de la pantalla. El haz traza una **línea horizontal o de barrido**

[1] Al búfer de estructura se le conoce mejor como *frame buffer* y esto, según la bibliografía anglosajona.

a lo ancho de la pantalla, baja una línea y se mueve al lado izquierdo de la misma para trazar la siguiente línea de barrido. La figura 1-12 ilustra lo dicho en este párrafo.

Conforme el haz de electrones se mueve a través de cada línea, se modifica la intensidad del haz para crear un patrón de manchas iluminadas. La definición de la imagen se almacena en un área de memoria llamada **búfer de repasado o búfer de marco o búfer de estructura**. Esta área de memoria contiene el conjunto de valores de intensidad para todos los puntos de la pantalla. Por cierto, cada punto en la pantalla se conoce como **píxel** o **pel**[2].

Según la tecnología de la televisión, la imagen en un CRT debe refrescarse o repasarse, esto es, regenerar la imagen entre 50 y 80 veces por segundo. En ocasiones los índices de repasado se describen en unidades de ciclos por segundo o mejor dicho hertz[3], donde un ciclo corresponde a un cuadro. Al utilizar estas unidades, describiríamos un **índice de repasado** de 60 cuadros por segundo sólo como 60 Hz.

Hasta aquí llegamos con el capítulo, ya que de otra forma me extendería demasiado explicando el CRT. Si desea mayor información puede consultar a *Foley, "Computer Graphics; Principles and Practice", 1990.* Este libro contiene amplia información sobre distintos tipos de monitores. Si lo que desea es información acerca de las nuevas tecnologías de panel plano, puede consultar a *Donald Hearn, M. Pauline Baker, "Gráficas por Computadora", Prentice Hall, 2ª Ed. 1995.*

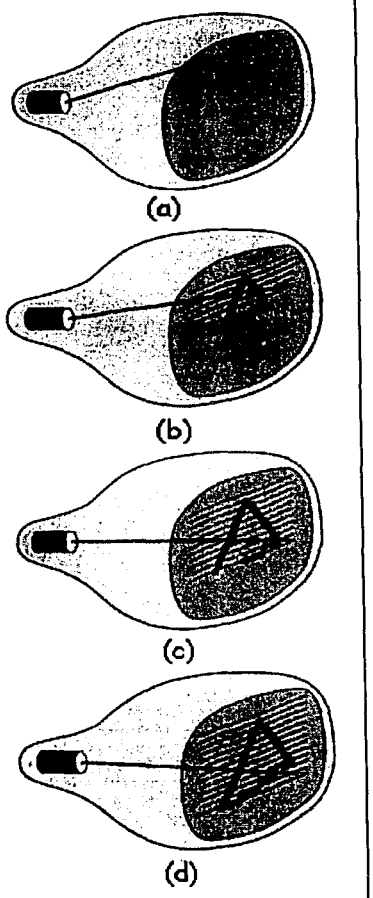
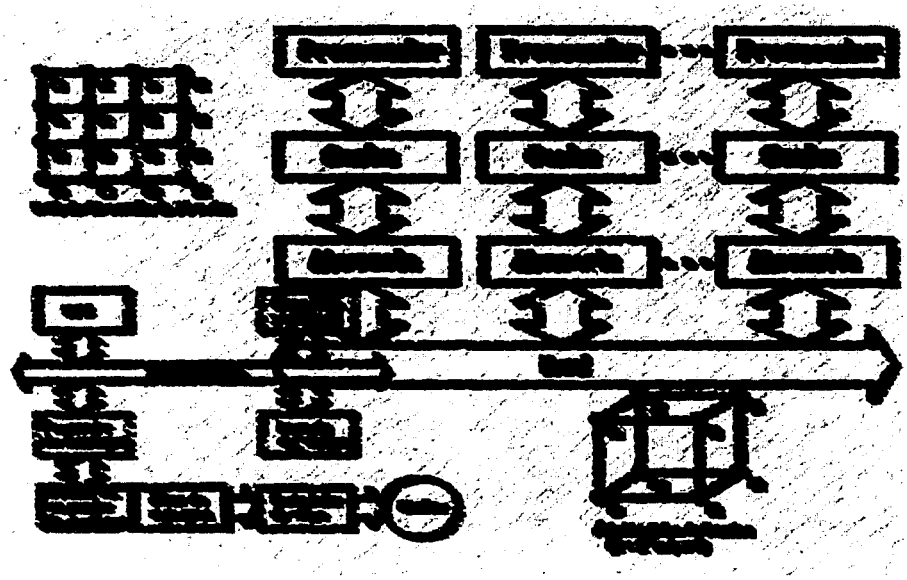


Figura 1-12
Un sistema de despliegue por líneas de rastreo. Un objeto se exhibe como un conjunto de puntos discretos a lo largo de cada línea de rastreo. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 1995, pg. 41.

[2] pel esta abreviatura de *picture element* o en español, elemento gráfico.
[3] hertz se abrevia como hz.



Capítulo 2 Arquitecturas genéricas: prototipos

Del capítulo anterior adquirimos un conocimiento previo de lo que es una tubería gráfica. De este concepto aprendimos que un programador proporciona todos los comandos necesarios para construir una escena. Pero alto, se trata de una sola escena y para que las capacidades de graficación propias de nuestro equipo sean de utilidad es mejor que la escena se pueda animar, es decir, que tenga movimiento.

La tendencia hoy día implica a un circuito dedicado que se encargue de la tubería gráfica. Este circuito es lo que llamamos el **subsistema gráfico**. Otro nombre que resulta común para referirnos a tal subsistema es el de **tarjeta de vídeo.** La tarjeta de vídeo se encarga tanto de la tubería gráfica como del proceso de desplegar la imagen en un monitor. Ahora bien ¿quién se encarga de calcular las posiciones de los objetos en escena? Pues bien, es un circuito que llamaremos el procesador principal. Como ejemplo ilustrativo puedo mencionar a los procesadores en **Pentium IV**[1], **Mips R10000**[2], **Power PC**[3], etc.

El procesador principal no se dedica sólo calcular posiciones de objetos en una escena. En realidad se encarga de realizar todo un proceso de simulación basado en los conceptos de la Física. Para ejemplificar, el sistema **Origin 2000**[4] es una red con soporte de hasta 128 procesadores, lo que significa una capacidad de simular vehículos dentro de un túnel de

[1] Pentium IV es una marca registrada de Intel, Copr.
[2] Mips R10000 es una marca registrada de Mips Technologies, Inc.
[3] Power PC es una marca registrada de Motorla, Inc.
[4] Origin 2000 es una marca registrada de Silicon Graphics, Inc.

viento. La precisión de la simulación con Origin 2000 es tal que no se requiere construir maqueta alguna para obtener información.

Los párrafos anteriores nos dan la excusa para la revisión de diversos tipos de arquitectura sobre computadoras. El estudio lo iniciamos revisando arquitecturas monoprocesador, sin embargo debo advertir que la mayoría son más un concepto de historia. Este inicio también nos mostrará la interacción que existe entre el trío procesador-memoria-tarjeta de video. El estudio continúa con las arquitecturas que soportan múltiples procesadores ya sean éstos de anillo con red.

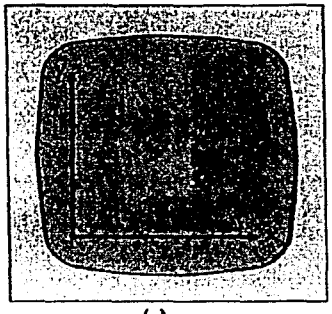


Figura 2-1
Sistemas de coordenadas de monitor. En (a) el sistema corresponde a una estación de trabajo. En (b) el sistema coordenada es para una PC. Donald Eran, M. Pauline Baker, *Gráficas por Computadora*, Prentice Hall, 2ª

2-1 Clasificación de computadoras según sus procesadores

En los años 60 se propuso un modelo de clasificación de computadora que todavía se utiliza. Un equipo se puede categorizar contando tanto el número de **unidades de control** como de las **tuberías de ejecución**. Alto, ¿qué es una tubería de ejecución? Para que un procesador moderno ejecute una instrucción, requiere pasarla por múltiples etapas, a éstas en conjunto se les conoce como la tubería de ejecución. Hemos reservado un capítulo completo para hablar de ello más adelante. La tubería no es sólo un estilo, es una técnica y se le puede comparar con la línea de montaje de autos de Henry Ford. Veamos la clasificación:

- Flujo único de instrucciones, flujo único de datos (SISD).
- Flujo único de instrucciones, flujo múltiple de datos (SIMD).
- Flujos múltiples de instrucciones, flujo único de datos (MISD)
- Flujos múltiples es instrucciones, flujos múltiples de datos (MIMD)

Un sistema SISD implica a un solo procesador, esto es, una unidad de control y una tubería de ejecución. Un SIMD en cambio implica una unidad de control y múltiples tuberías de ejecución, lo que es la tendencia actual. Las tuberías bien podrían interconstruirse con la unidad de ejecución, es decir, en un mismo encapsulado; o bien, podrían implantarse en distintos encapsulados y el usuario los iría comprando según sus necesidades. Un MISD implica múltiples unidades de control para activar una tubería de ejecución, lo que es un concepto tan extraño que no se implementa. Finalmente el MIMD, implica múltiples unidades de control y múltiples tuberías de ejecución. El MIMD normalmente se construye conectando varios procesadores SISD con un bus o con una red de comunicación.

2-2 El controlador de vídeo.

Debemos adelantarnos un poco en cuanto subsistemas gráficos. Esto se debe a que requerimos de esta información para comprender las ventajas y desventajas de las primeras arquitecturas que revisaremos durante el presente capítulo: las primeras arquitecturas son sistemas SISD.

Recordemos cómo se localizan los píxeles en un monitor de trama. Las posiciones de píxel en la pantalla se expresan en coordenadas cartesianas. En algunos monitores, como los que emplean las estaciones de trabajo, el origen de coordenadas se define en esquina inferior izquierda, según lo muestra la figura 2-1.a. En otros monitores, como los que se

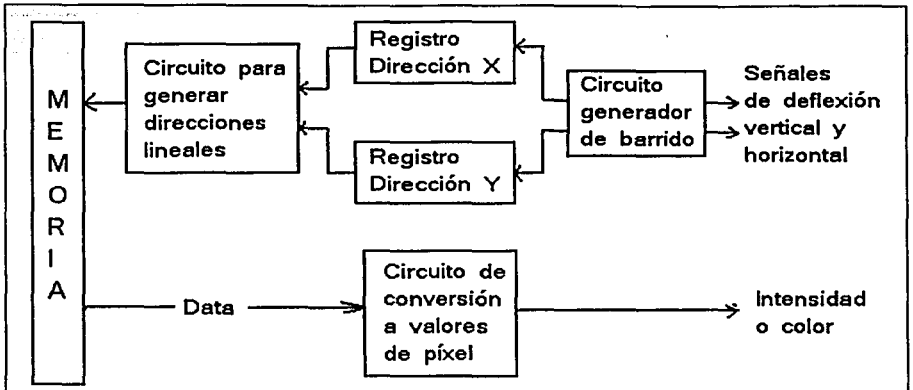


Figura 2-2
Organización lógica de un controlador de video. Las líneas de flechas no son representación de algún tipo de bus; se les emplea solo para indicar la dirección de las señales eléctricas. Foley, Van Dam, Feinderm Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, pg, 167.

emplean en una PC, el origen de coordenadas se localiza en la esquina superior izquierda, según la figura 2-1.b.

La figura 2-2 es un esquema simplificado de un controlador de vídeo. Sólo se muestran las partes que por ahora nos interesan. Ahora ¿cómo funciona este dispositivo? Pues bien, suponiendo que nuestro monitor emplea el sistema de coordenadas de la figura 2-1.a, el generador de barrido coloca en el registro *X* el valor inicial de *0* y en el registro *Y*, el valor de y_{max}. Ahora bien, los valores de píxel se almacenan en una memoria a la que llamaremos **búfer estructura**. Como los datos en el búfer de estructura se guardan en direcciones lineales, se hace necesario convertir las coordenadas *(X,Y)* a direcciones lineales de memoria. El valor almacenado en el búfer para esta posición de píxel se recupera y se usa para fijar la intensidad del haz del CRT. Entonces se incrementa el registro *X* en razón de 1 y se repite el proceso para el siguiente píxel en la misma línea del barrido. Luego de procesar el último píxel en la línea de barrido se determina el registro *X* como *0* y el registro *Y* se decrementa en *1*. Entonces se procesan los píxeles a lo largo de esta línea de barrido y se repite el procedimiento para cada línea de rastreo sucesiva. Luego de haber procesado todas las líneas de rastreo, el generador de barrido fija los valores que los registros *X* y *Y* para ubicar la primera posición de píxeles en la línea de rastreo superior y así, un proceso de retrazado de la imagen comienza de nuevo.

2-3 prototipos SISD

Cada una de las arquitecturas que vamos a mencionar responde a diversas necesidades según los criterios de costo-rendimiento. La primera arquitectura es la más simple y económica. Puede ser empleada cuando solamente se requiere sintetizar una escena fija. La segunda arquitectura es igualita a la primera, excepto en que incorpora el concepto de memoria multipuerto, lo que aumenta el rendimiento del sistema. La tercera arquitectura se ha especializado para síntesis de imágenes e implica un alto costo; no obstante presenta un buen rendimiento en cuanto a síntesis de escenas animadas. Por supuesto estos modelos de arquitectura son sólo prototipos y no se implementan como tales. Cada

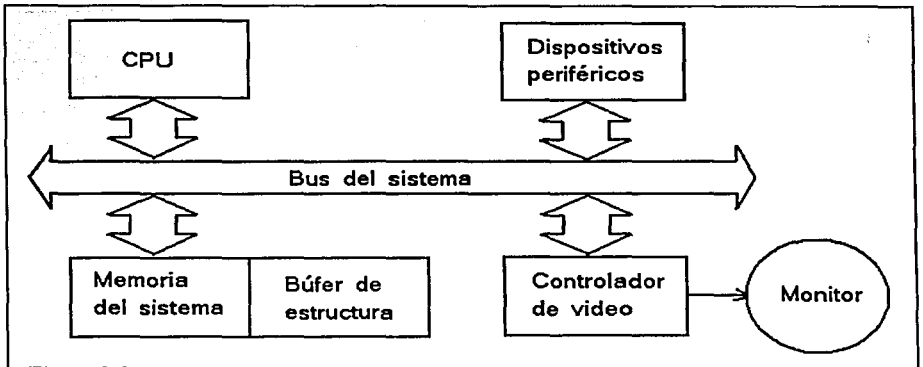


Figura 2-3
Arquitectura de un sistema gráfico simple de rastreo. Note que una porción de la memoria del sistema se reserva para el búfer de estructura. La información de la imagen es enviada entonces a través del Bus del sistema. El controlador de video hace lo suyo enviando las señales de deflexión al monitor: línea de flecha. Foley, Van Dam, Feinderm Hughes, *"Computer Graphics, Principles and Practrice"*, 2ª edición, pg, 167.

fabricante ha diseñado su propia variante con las mejoras que ha considerado necesarias. Más adelante, en un capítulo posterior, verificaremos algunas arquitecturas del mercado.

Sistema gráfico simple de rastreo

La organización más simple y común de un sistema de despliegue de barrido es mostrada de la figura 2-3. Note que una porción de la memoria del sistema se reserva bajo el nombre de búfer de estructura. El controlador de vídeo despliega la imagen ya definida en el búfer de estructura. El acceso del controlador a este búfer es a través del bus de sistema según se puede notar en la figura 2-3.

Esta arquitectura se caracteriza en que el sistema operante, los programas de aplicación y el paquete de subrutinas gráficas son ejecutados por el CPU. El paquete gráfico incluye rutinas que dibujan puntos, líneas, círculos y demás figuras fijando los valores de píxel en el búfer de estructura. La forma en como se realiza esto la veremos en un capítulo posterior. Note ahora la facilidad que tiene el CPU para sintetizar cualquier imagen fija, ya que este búfer se encuentra en el espacio de direcciones al que puede acceder directamente el CPU.

Esta arquitectura tiene una desventaja cuando la velocidad de refresco del monitor o el número de píxeles a desplegar aumenta, también aumenta el número de accesos a memoria por el procesador de vídeo, así que el CPU tiene limitado el número de accesos a memoria. Una consecuencia natural de esta situación, es la reducción del rendimiento del sistema en general.

Una posible ventaja, implica al costo de venta de este tipo de máquina. Si no me cree, revise la historia de Macintosh y de IBM en cuanto sistemas caseros.

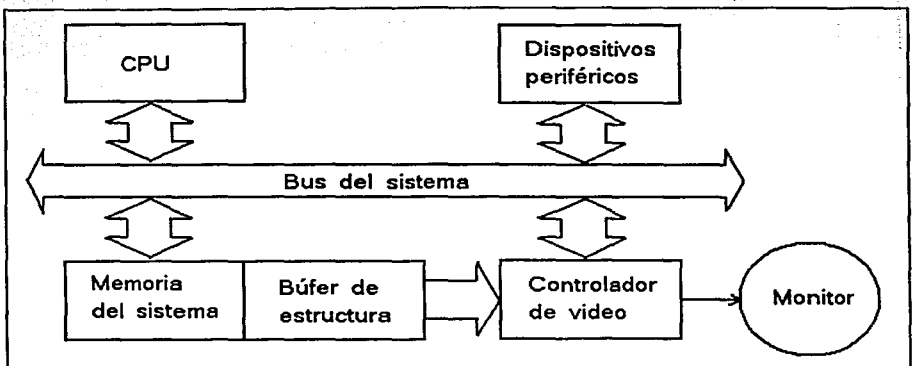


Figura 2-4
Arquitectura de un sistema simple de despliegue por rastreo. Esta arquitectura implica una memoria multipuerto, por lo que es posible implementar un bus secundario entre el búfer de estructura y el controlador de video. La existencia de este bus, libera el bus del sistema de una gran carga de información. El sistema de la figura reserva un espacio fijo de direcciones para el búfer de estructura. Foley, Van Dam, Feinderm Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, pg, 166.

Sistema gráfico simple de rastreo con bus dedicado

Este sistema es igualito al anterior, pero tiene en una diferencia en cuanto a su construcción, la figura 2-4 nos muestra que el controlador de vídeo puede acceder búfer estructura a través de un bus que los conecta directamente. Otra diferencia que quizá haya deducido, implica que el circuito de memoria del búfer de estructura tiene dos puertos, uno conectado al bus del sistema y otro al controlador de vídeo. El acceso a estos puertos puede realizarse idealmente en cualquier momento. Así que el rendimiento del sistema está limitado a la velocidad del CPU para generar la imagen.

El sistema de la figura 2-4, se caracteriza en que se reserva un espacio fijo de direcciones para el búfer de estructura. En la misma figura, por simpleza, no se ha dibujado el bus de direcciones entre el controlador de vídeo y el búfer de estructura. En este sistema, aún existe la limitante del ancho de banda de los circuitos de memoria, por lo que sí se desea aumentar la cantidad de pixeles a mostrar en pantalla, el rendimiento del sistema decrece.

Sistema gráfico de rastreo con procesador de despliegue periférico

La figura 2-5 muestra el diagrama de bloques de esta arquitectura. Se trata de una construcción que es más común en el mundo del PC y de la estación de trabajo. Tal diseño evita las desventajas de las arquitecturas arriba mencionadas introduciendo un procesador gráfico separado para realizar diversas funciones tales como el graficar puntos, líneas, círculos, elipses y otras figuras, ya sea sólo el perímetro o el área.

Así que ahora tenemos dos procesadores, uno es un procesador de propósito general que identificamos como el CPU y el otro es un procesador de propósito especial o **procesador de despliegue**. Adicionalmente contamos con tres áreas de memoria: la memoria del sistema, la memoria del procesador despliegue y el búfer de estructura. La memoria del sistema conserva datos más esos programas que ejecuta el CPU: programas de aplicación,

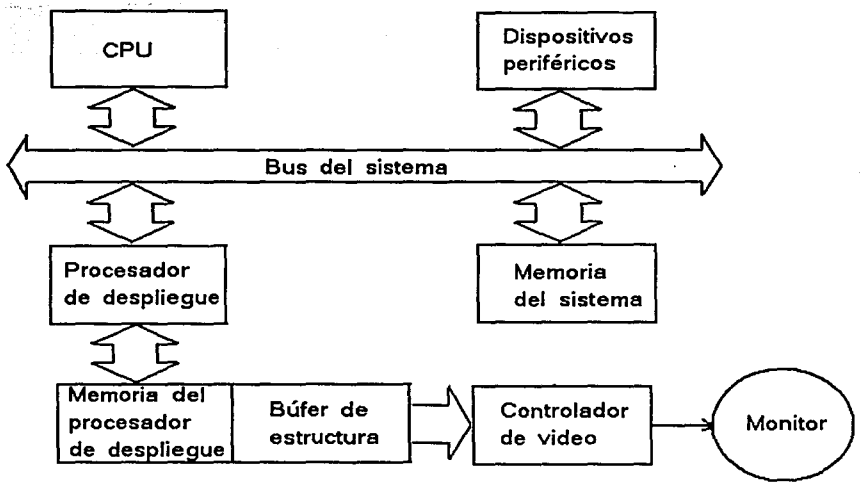


Figura 2-5
Arquitectura de un sistema con procesador de despliegue periférico. No que el sistema cuenta ya con dos procesadores: el CPU y el procesador de despliegue. Adicionalmente se tienen tres zonas de memoria: la memoria del sistema, la memoria de despliegue y el búfer de estructura. Foley, Van Dam, Feinderm Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, pg, 171.

el paquete gráfico y el sistema operante. Similarmente, la memoria del procesador de despliegue conserva algunos datos más el paquete gráfico y otros programas que controlan la generación de imagen. Alto ¿cómo que el paquete gráfico se encuentra tanto en la memoria del sistema como en la memoria del procesador de despliegue? Pues bien, en algunos casos, el procesador de despliegue era únicamente una lógica integrada para realizar el mapeo de coordenadas *(X, Y)* a direcciones lineales de memoria. En esos casos, las operaciones gráficas aún eran realizadas por CPU. Y ¿qué hay de los otros casos? Los otros casos, implican a un procesador de despliegue que acepta diversos comandos, tales como dibujar píxeles, líneas, círculos y circunferencias, polígonos, desplazar áreas rectangulares de la pantalla, etc. Estos comandos, originalmente residen en la memoria del sistema en forma de lista, pero luego son pasados al procesador de despliegue que los almacena en una pila local. Una vez que la pila se ha llenado, el procesador ejecuta esos comandos. Mientras el procesador ejecuta los comandos de una pila, el CPU está libre para realizar otras tareas, tal como atender al sistema operativo, ejecutar otra aplicación, o mejor aún, construir otra lista de comandos.

Concluyendo, esta arquitectura es especialmente conveniente para sintetizar imágenes 2D y 3D mediante primitivas.

Conclusiones sobre las arquitecturas básicas

Tal vez se pregunta ¿por qué establecer un bus secundario entre el controlador de vídeo y el búfer de estructura? La respuesta la obtendremos al calcular el flujo de información que se requiere para crear una imagen en el monitor. Suponga que tenemos un monitor cuya imagen debe refrescarse 60 veces por segundo: esto es con la finalidad de evitar parpadeos en la imagen. Suponga también que nuestro monitor debe mostrar 1024 por

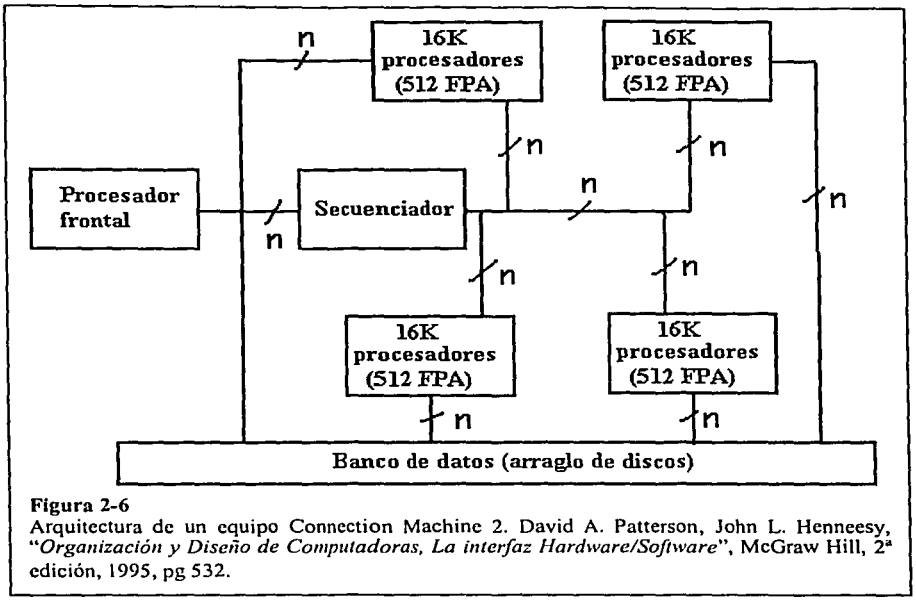


Figura 2-6
Arquitectura de un equipo Connection Machine 2. David A. Patterson, John L. Henneesy, "*Organización y Diseño de Computadoras, La interfaz Hardware/Software*", McGraw Hill, 2ª edición, 1995, pg 532.

768 píxeles, esto es un poco inferior a las capacidades de una estación de trabajo. Ahora bien, cada píxel se representa con tres o octetos o bytes. Así que por imagen tenemos 1024 por 768 por 3 igual a 2'359,296 bytes de información. Ahora la imagen se refresca 60 veces por segundo, así que requerimos de un ancho de banda de 141'557,760 bytes por segundo. Sencillamente abrumador y demasiado para cualquier bus de sistema. Más adelante, en un capítulo posterior, veremos que el controlador de vídeo no copia byte a byte la información desde el búfer de estructura, en vez de ello, es capaz de copiar varios bytes de manera paralela en un solo acceso; situación que permite a este bus dedicado estar dentro de las capacidades tecnológicas actuales.

2-4 Arquitectura SIMD

Estas arquitecturas se caracterizan en que:

- El procesador cuenta con una unidad de control y múltiples tuberías de ejecución.
- Cada tubería de ejecución tiene acceso a un segmento de memoria del sistema que usará como memoria local.
- Cada tubería de ejecución tiene sus propios registros de direcciones para localizar datos en su memoria local.
- Por ejemplo, queremos sumar 64 pares de números, entonces un programa envía 64 pares de números a 64 segmentos de memoria local de las 64 tuberías. Todas las tuberías responden a una sola instrucción de sumar que emana de un único contador de programa.

Ahora veamos un ejemplo de un equipo real, un equipo Connection Machine 2[5] o simplemente CM-2[6]. Este equipo está formado por 65,536 procesadores de un bit. Cada procesador tiene interconstruidas sus unidades lógica y aritmética. Para el ejemplo que nos interesa, la CM-2 tiene 2048 unidades aceleradoras de punto flotante que agrupan, cada una, a 32 procesadores de un bit. Alto ¿qué quiere decir esto? Quiere decir que la CM-2 tiene el equivalente a 2048 unidades de punto flotante.

Veamos una comparación, el procesador R2000 de MIPS, puede ejecutar una suma de 32 bits en 0.066 microsegundos, si los datos están almacenados en sus registros. Así que en un segundo puede ejecutar 15.152 millones de sumas, aproximadamente. Ahora bien, una unidad de punto flotante de la CM-2 realiza una suma en 21microsegundos, si los datos están almacenados en su memoria local. Dado que le es posible realizar las 2048 sumas a la vez, la CM-2 puede realizar 97.52 millones de sumas de 32 bits en un segundo. Para aumentar aún más la diferencia, el R2000, generalmente, realizará una suma en 0.98 microsegundos si los operados están en la caché.

La figura 2-6 muestra la organización de la CM-2. La operación del equipo está coordinada por un procesador SISD, denominado de comunicación o frontal o bien, *front-end* según la bibliografía anglosajona. El procesador frontal ejecuta un programa y ya que algunas de las instrucciones son SIMD, se envían a un circuito denominado secuenciador, que difunde las instrucciones a los 65,536 procesadores. Recuerde que la operación de 65,536 procesadores se controla con otros circuitos tales como las unidades aceleradoras de punto flotante.

2-5 Computadoras MIMD: Múltiples flujos de instrucciones, múltiples flujos de datos

La idea principal de los computadores MIMD es un computador potente que se construye conectando muchos computadores pequeños y se ha hecho, hoy día tenemos una red de computadoras por la cual se transportan cantidades inimaginables de información. Un ejemplo en el que se realizan procesamientos paralelos lo tenemos en la empresa Pixar[7]. Cuando se les encarga una animación, podemos ver a varios dibujantes trabajando a un tiempo en distintas computadoras; éstas se conectan por una red de comunicación a un conjunto de servidores que les da soporte para almacenamiento masivo, programas de aplicación, etc.

También existen computadoras que dentro de un solo gabinete incluyen una red de procesadores, tal como los equipos Origin de Silicon Graphics. Otras empresas que fabrican equipos multiprocesador son IBM y DEC.

Los computadores MIMD presentan una ventaja debido a un concepto llamado **escalabilidad del software** ¿qué significa éste término? Es muy simple, veamos, algunos MIMD soportan operaciones en presencia de hardware desigual; es decir, si uno de n procesadores falla, el sistema proporciona servicio con $n-1$ procesadores.

Hay que aclarar un par de términos, un computador MIMD es capaz de soportar a varios usuarios, cada uno ejecutando su propio programa. La capacidad de un computador de ejecutar múltiples programas es lo que llamamos **multitarea**. Otro término a verificar ese

[5] Connection Machine es una marca registrada de Thinking Machines.

[6] En los tiempos de la CM-2 el concepto de tubería de ejecución estaba aún en pañales.

[7] Pixar es una empresa que se dedica a la creación de dibujos animados por computadora. Sus animaciones se caracterizan por la presencia de movimientos faciales y musculares. La sede está en Estados Unidos de Norteamérica.

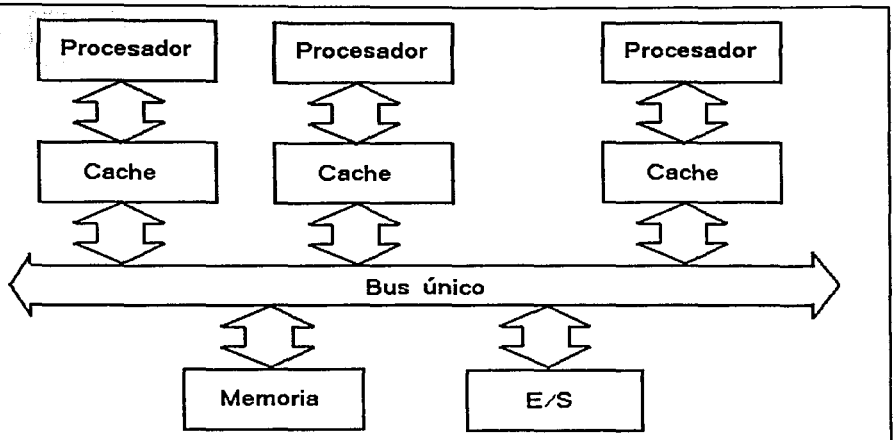


Figura 2-7
Arquitectura de un sistema mutiprocesador de un solo bus. El tamaño de este sistema oscila entre los 2 y los 32 procesadores. David A. Patterson, John L. Henneesy, "*Organización y Diseño de Computadoras, La interfaz Hardware/Software*", McGraw Hill, 2ª edición, 1995, pg 538.

procesamiento paralelo. Este término lo empleamos para referenciar a un único programa que corre en múltiples procesadores simultáneamente.

Los MIMD se construyen con dos estilos básicos: procesadores por un solo bus y procesadores conectados por una red.

Algunos de los inconvenientes de los MIMD son:

- El costo de cada procesador.
- Los defectos de las topologías de las redes.
- La indisponibilidad de los lenguajes de programación para operar estos sistemas.
- Respecto de este último punto, es el programador de aplicaciones quien que debe coordinar la comunicación entre los distintos procesadores.
- Aunque *n* procesadores pueden tener el potencial para terminar cualquier tarea con *n* veces más rapidez, los gastos de comunicaciones del grupo pueden evitarlo; aumentar la rapidez *n* veces llega a ser prácticamente improbable cuando *n* aumenta.

MIMD conectado por un solo bus

La figura 2-7 muestra el prototipo de un multiprocesador de un solo bus. Este tipo de sistemas pueden contener entre 2 y 32 procesadores, por cierto, 32 procesadores son el máximo que se ha ensamblado. Este esquema ofrece al programador un único espacio de direcciones, esto es, memoria compartida por todos los procesadores, o sea que se puede acceder a cualquier parte de la memoria.

Dado que los procesadores operan en paralelo, necesitan coordinarse cuando trabaja sobre datos compartidos; en caso contrario, el procesador podría comenzar a trabajar sobre un

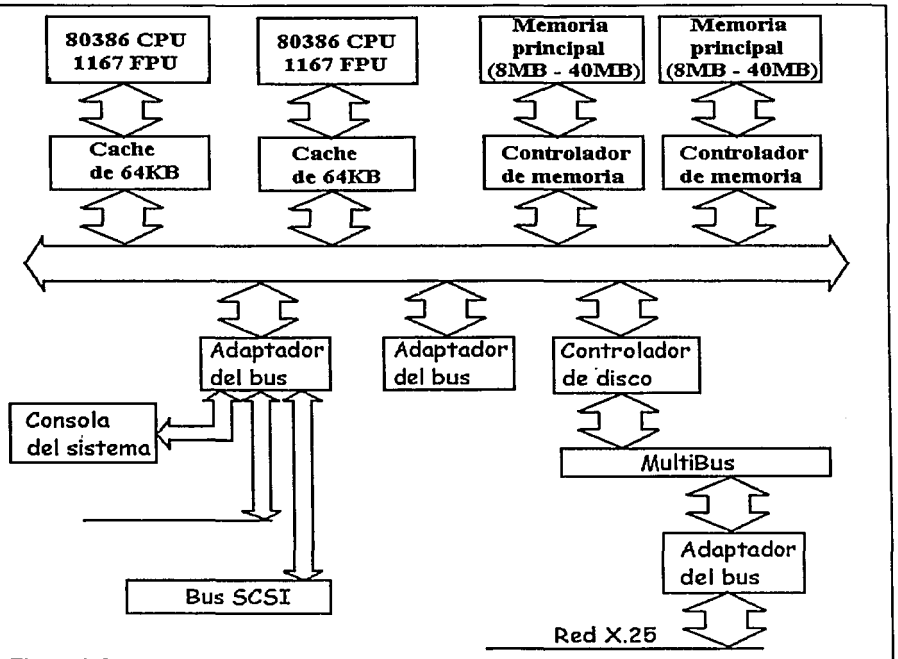


Figura 2-8
Arquitectura del multiprocesador Sequent Symmetry. Este equipo soporta hasta 30 microprocesadores, cada uno con 64 KB de caché. El ancho del bus de sistema es de 64 bits. Este prototipo fue implementado en los años 80. David A. Patterson, John L. Henneesy, "*Organización y Diseño de Computadoras, La interfaz Hardware/Software*", McGraw Hill, 2ª edición, 1995, pg 548.

dato antes que otro termine con éste. Esta coordinación implica un proceso denominado **sincronización**.

Note también que los procesadores emplean memoria caché para reducir el paso de información por el bus del sistema. ¿Qué ocurre cuando varios procesadores necesitan una copia del mismo dato en sus respectivas memorias caché y uno de estos procesadores necesita modificar el dato? Existen dos soluciones, una implica invalidar las copias en las otras memorias caché y otra implica actualizar la copia:

- **Invalidación de escritura**: El procesador que modifica el dato hace que en las demás memorias caché se invaliden las copias antes de cambiar su copia actual; entonces está libre para actualizar el dato. Este dato estará disponible cuando otro procesador lo pida.
- **Actualización de escritura**: el procesador que modifica el dato difunde el nuevo valor por el bus; entonces se actualizan todas las copias con el nuevo valor.

La figura 2-8 muestra el prototipo de multiprocesador Sequent Symmetry[8]. Los experimentos realizados con este equipo, al ejecutar 10 copias del mismo programa, demostraron que la técnica de invalidación de escritura reducía la cantidad de información

[8] Sequent Symmetry es una marca registrada de Sequent Computer Systmes, Inc.

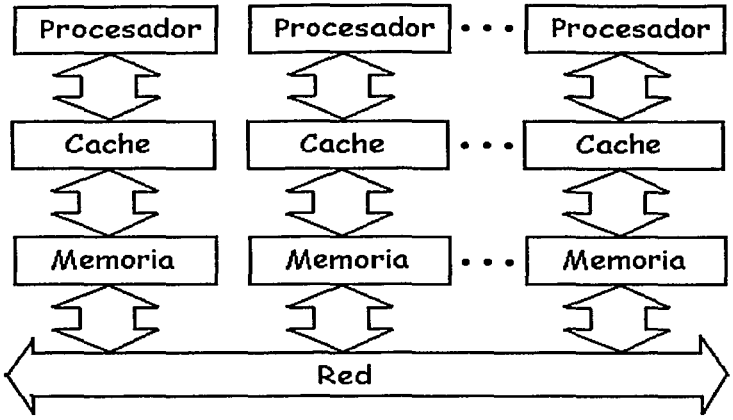


Figura 2-10
Arquitectura de un sistema multiprocesador conectado en red. El tamaño típico de estos sistemas oscila entre 32 y 1024 procesadores. Observe que a diferencia de la figura 2-7, la conexión de los multiprocesadores no es más larga entre la memoria y el procesador. David A. Patterson, John L. Hennecsy, "*Organización y Diseño de Computadoras, La interfaz Hardware/Software*", McGraw Hill, 2ª edición, 1995, pg 549.

que debía cruzar por el bus y que el rendimiento del sistema era semejante para cada tarea que debía ejecutar. La cantidad máxima de procesadores que soportó el experimento antes de degradar su rendimiento fue de 28.

Falta por mencionar una característica adicional acerca de los procesadores conectados en bus. Lo que nos trajo examinar los sistemas MIMD era nuestra necesidad de que varios procesadores ayudarán a nuestra tarea de simular un sistema físico. Para esto, requerimos que los procesadores trabajen sobre un mismo conjunto de datos, lo que significa que los procesadores deben comunicarse con la memoria continuamente. En un sistema en bus, sólo es posible establecer un único enlace procesador-memoria a la vez, situación que limita mucho el paralelismo de las operaciones. La figura 2-9 nos muestra un enlace de comunicación en donde dos procesadores comparten la información de sus respectivas memorias caché mientras otro espera el momento en que podrá hacer uso del bus.

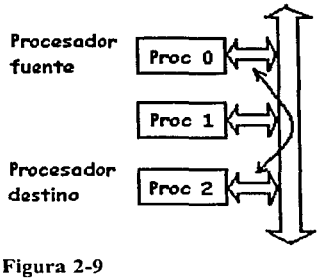


Figura 2-9
La figura muestra a dos procesadores empleando el bus. Un tercer procesador espera su oportunidad de usar el bus para comunicarse con otro procesador.

MIMD por conexión a red

La figura 2-10 muestra el prototipo de un sistema cuyos procesadores están conectados a una red. Es importante notar que la memoria está distribuida, así que tal característica permite aumento tras el número de procesadores sin degradar el rendimiento. Los números típicos de procesadores conectados en red varían entre 32 y 1024. Empresas como Cray Research, antes de su fusión con Silicon Graphics, ya construían este tipo de sistemas.

Nuestro estudio de una red MIMD va a ser fundamental, así que los elementos constituyentes básicos de una red son los siguientes:

- Nodo
- Conmutador

Figura 2-11
La figura representa una topología de red de anillo. Los puntos representan los conmutadores en tanto que cada cuadro es un trío procesador-caché-memoria.

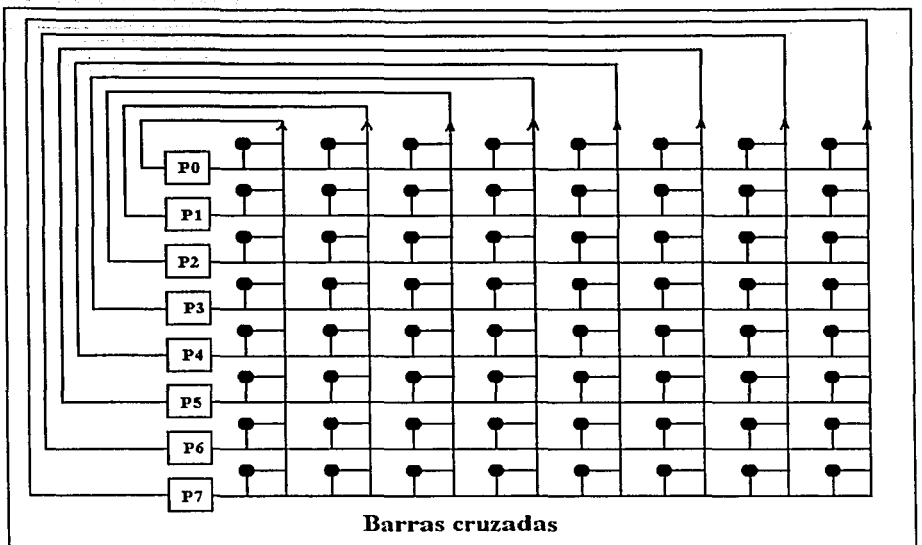
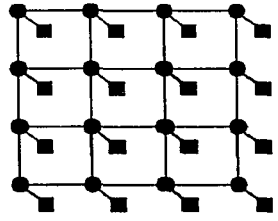


Figura 2-13
Topología de barras cruzadas. Este ejemplo requiere de 64 conmutadores que le permiten soportar cualquier combinación de mensajes entre procesadores. David A. Patterson, John L. Henneesy, "*Organización y Diseño de Computadoras, La interfaz Hardware/Software*", McGraw Hill, 2ª edición, 1995, pg 555.


Retícula 2D o malla de 16 nodos

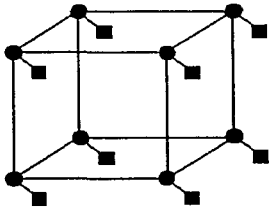

Árbol N-Cubo de 8 nodos
($8 = 2^n$ así, n=3)

Figura 2-12
Topologías de red. En (a) tenemos una retícula y en (b) tenemos una red 3-cubo. David A. Patterson, John L. Henneesy, "*Organización y Diseño de Computadoras, La interfaz Hardware/Software*", McGraw Hill, 2ª edición, 1995, pg 553.

- El enlace
- Procesador
- Memoria
- Caché

Las redes normalmente se dibujan como grafos, cada arco representa un enlace de la red de comunicación. El nodo es un trío procesador-caché-memoria y se muestra como un cuadrado negro. Las redes MIMD se parecen a las redes telefónicas, por tanto necesitamos conmutadores, es decir, circuitos que formen el camino por el cual viajará la información de un procesador a otro. El conmutador se muestra como un círculo sombreado. La figura 2-11 muestra una topología de anillo, que es la más simple.

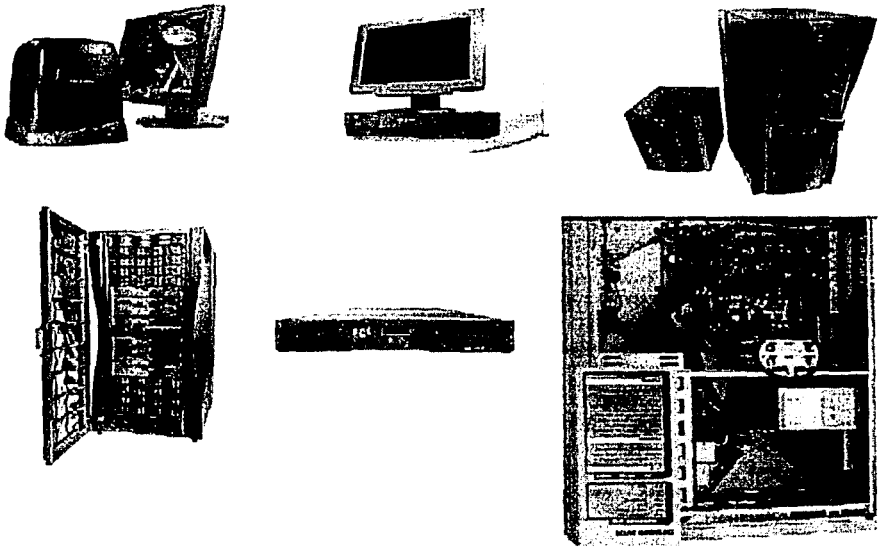
Al observar un poco la topología de anillo, podríamos pensar que es igual a un MIMD de bus, sin embargo, dado que los enlaces en una red son bidireccionales, podemos tener más de dos procesadores comunicándose a la vez. Bueno, algunos nodos no están directamente conectados, así que algunos mensajes tendrán que saltar a lo largo de nodos intermedios hasta que lleguen al destino.

Una red MIMD también se caracteriza en que la memoria, aunque está distribuida, es vista por los programas de aplicación como un único espacio de direcciones.

Entre el alto costo-rendimiento de una red totalmente conectada y el bajo costo-rendimiento de una red de anillo, existe un conjunto de redes que constituyen un amplio rango de compromisos. Los costos de la red implican el número de conmutadores, el número de enlaces y la longitud de los enlaces cuando una red tiene correspondencia con otro equipo remoto. La evaluación del éxito depende en gran parte de la naturaleza de la

comunicación y en la carga de trabajo de los programas paralelos que corren en la máquina. La figura 2-12 ilustra dos topologías populares. De esta figura nos es conveniente analizar un poco la figura 2-12.b. La topología *n-cubo booleano* es una interconexión n-dimensional con *2n nodos*, que necesita *n* enlaces por conmutador (mas uno para el procesador) y así, cada nodo está más próximo a los nodos vecinos. Las implementaciones comerciales con frecuencia añaden enlaces extra a esas sencillas topologías para mejorar el rendimiento y la fiabilidad.

Una alternativa considera un sistema de barras cruzadas, tal como se hacía en los sistemas telefónicos. La figura 2-13 muestra este sistema. Un sistema de barras cruzadas es equivalente a un sistema en el que cualquier procesador tiene conexiones a todos los demás. El sistema de barras cruzadas permite que cualquier nodo se comunique con cualquier otro nodo a través de la red.



Capítulo 3 Arquitecturas especializadas

El capítulo anterior nos mostró los conceptos básicos de arquitecturas monoprocesador y multiprocesador, por supuesto, en el caso multiprocesador, tuve que mostrar algunas implementaciones ya que no hay arquitecturas básicas que sirvan de modelo. Ahora nos toca revisar la arquitectura de algunas implementaciones de empresas como:

- Silicon graphics: O_2[1].
- Intel: Desktop 815 y Pentium III[2].
- Sun Microsystems: UPA[3].

En este capítulo me limitaré sólo a sistemas monoprocesador y aún, solo a mencionar las partes generales. El cómo funcionan sus sistemas gráficos lo mostraré en un capítulo posterior.

Debemos recordar que la arquitectura de una computadora se limita a un diagrama de bloques que muestra la relación funcional entre componentes y no incluye detalles sobre su implementación física y a veces, por protección de la empresa, ni siquiera datos de cómo se comunican. Por cierto, los diagramas que se muestran en el capítulo son tal y como se encuentran en las fuentes de información. Seguramente notará que las personas que realizaron estas gráficas no siempre diferencian entre un bus y una simple línea de comunicación.

---

[1] O_2 es una marca registrada de Silicon Graphics, Inc.

[2] Desktop 815 e Intel Pentium III son marcas registradas de Intel Corp.

[3] UPA (Ultra Port Architectur) es una marca registrada de Sun Microsystems.

Algo que también deseo del lector es que considere la cantidad de información mostrada. Intel libera toda clase de datos: mapas de tiempo, características eléctricas, modos de operación, etc. Toda su información llega a ser tan amplia y diversa que requiere de mucho tiempo para comprender de pleno como funcionan sus equipos. Silicon Graphics libera esta información únicamente a empresas registradas, no obstante, proporciona algunos datos que pudieran ser empleados por los fanáticos de computadoras. Sun Microsystems es aún más reservada, seguro es por su fama. Esta última empresa, a pesar de sus reservas de comunicación, también proporciona algunos datos básicos sobre su arquitectura y que espero sean suficientes considerando el nivel del presente trabajo escrito.

Cuando el lector haya terminado la lectura de este escrito sería bueno pensar en que una computadora, no sólo se forma con el microprocesador, de hecho los conceptos principales que deberá considerar todo aficionado son:

- Costos
- Características
    - Microprocesador.
    - El chipset o bien, la placa base.
    - Circuitos de memoria.
    - Subsistema gráfico.
    - Periféricos.
- Desempeño

Finalmente, los campos de aplicación de las computadoras monoprocesador, sin importar su marca, se emplean en:

- Servidores de red.
- Creación de contenidos digitales para la Internet.
- Producción y distribución de video. La distribución puede ser también a través de la Internet.
- Juegos de video y simuladores
- Diseño y desarrollo asistido por computadora.

3-1 Capacidades de un sistema O_2

Hay tres áreas importantes a las cuales se ha orientado el desarrollo de los sistemas O_2:

- Mapeo de texturas.
- Procesamiento de imágenes.
- Procesamiento de vídeo, composición y escenarios virtuales.

El sistema O_2 esta basado en la arquitectura de Memoria Unificada o bien, en forma abreviada UMA (UMA: *Unified Memory Architecture*) y en una memoria del tipo SDRAM (Synchronous Dinamic Random Access Memory) con un ancho de banda de 2.1GB por segundo. Memoria unificada significa que se tiene un único bloque de memoria sirviendo como búfer, caché, almacenamiento masivo, etcétera. También significa que el procesador principal tiene un único espacio de direcciones. El tipo de información que se almacena en la memoria es video, imágenes, texturas, aplicaciones, el sistema operativo y demás.

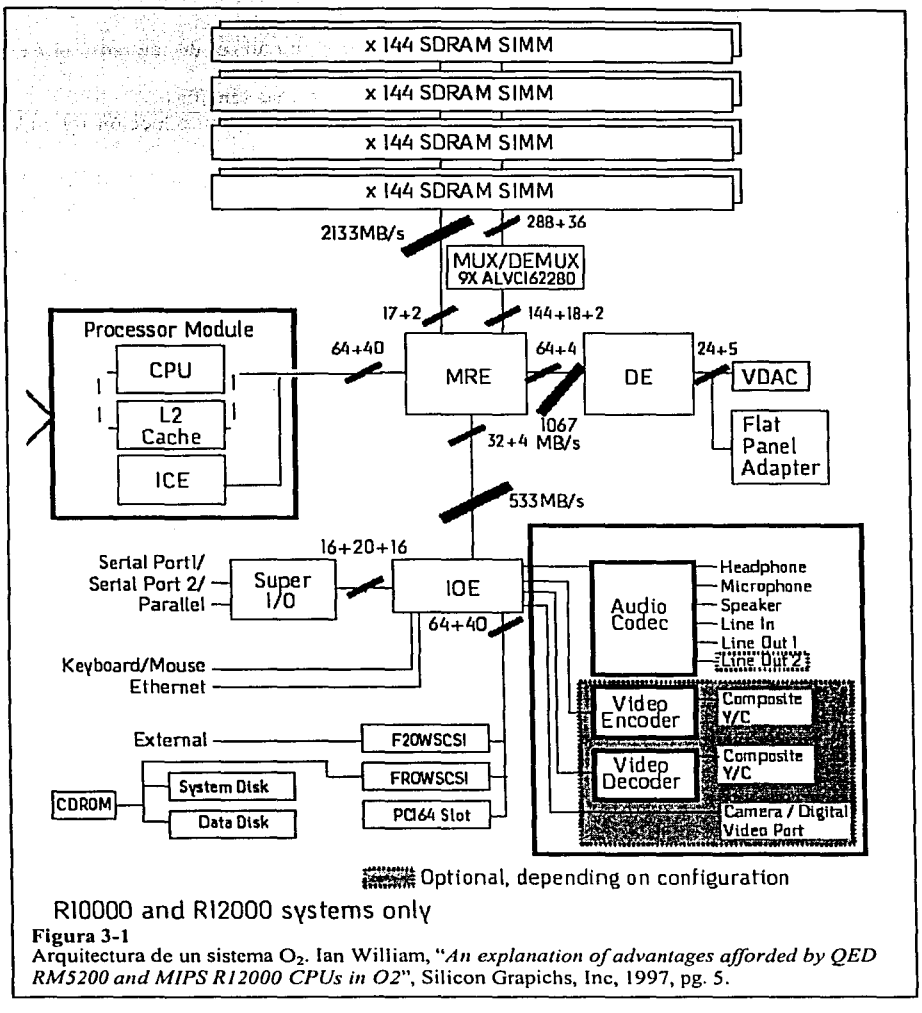


Figura 3-1
Arquitectura de un sistema O_2. Ian William, "*An explanation of advantages afforded by QED RM5200 and MIPS R12000 CPUs in O2*", Silicon Grapichs, Inc, 1997, pg. 5.

Cada tipo de dato reside en un segmento de memoria llamado búfer, así que todos los búferes comparte la misma memoria. Cada subsistema de O_2 tiene acceso a cada uno de estos búferes, según se requiera y sin la necesidad de realizar copias del contenido de alguno otro. Por ejemplo, se han almacenado en memoria datos de vídeo comprimido, por lo que estos datos pueden ser empleados como mapas de texturas en modelado 3D únicamente con pasar la dirección en memoria de los mismos.

La figura 3-1 nos muestra un diagrama a bloques generalizado de la arquitectura de un sistema O_2. De esta figura es necesario distinguir los siguientes elementos:

- El MRE que administra la memoria del sistema aunque de hecho es como un policía en un cruce de avenidas.
- Los bloques de memoria RAM.

- El módulo del procesador MIPS.
- El IOE es un circuito de interfaz con otros circuitos de entrada-salida y almacenamiento masivo.
- El DE que es un equivalente a las tarjetas de vídeo más sencillas.
- El ICE que nos da la oportunidad de disfrutar de la reproducción de vídeo comprimido.

El CPU

Vamos a empezar con el procesador del sistema. Si, ya sé, es algo que veremos en capítulos posteriores sin embargo en un momento verá porque incluyo información previa. Los sistemas O_2 consiguen su poder de cómputo de la familia de procesadores MIPS® IV: RISC R5000™ Y R10000™ y ya que está de paso, veremos porqué fue elegido.

- Implementa instrucciones de 64 bits.
- Tiene cinco tuberías para el procesamiento de instrucciones, dos tuberías para trabajar con enteros, otras dos para su trabajo con datos en punto flotante y la quinta relacionada con operaciones simples que implican el manejo de registros del procesador, etc.
- Puede terminar la ejecución de hasta cuatro instrucciones por ciclo de reloj.
- Su operación está fundada en el principio de suministro y ejecución dinámica de instrucciones.
- También tiene un predictor de saltos.
- Incluye dos memorias caché de nivel uno, una para datos, otra para código y cada una es de 32KB.
- La memoria caché de nivel dos puede ser de hasta 4MB.
- etc.

El ICE

El procesador MIPS, además de la caché de nivel 2, contiene otros circuitos como aquel que nos permite conectar dos procesadores en paralelo y un circuito que le hace un procesador muy especial. A este último circuito se le conoce como el ICE. Así es, dentro del encapsulado del procesador se incluye una unidad conocida como la *Image and Compresión Engine* o bien ICE (máquina de compresión de imágenes) la cual soporta:

- Compresión y descomprensión de vídeo MPEG el tiempo real.
- Codificación y decodificación de video en formatos NTSC, PAL y otros.

Con el sistema se incluye un paquete de software para soportar los algoritmos de compresión-descompresión MPEG-1, MPEG-2, Cinepack® e Indeo. Adicionalmente se ofrece compatibilidad entre los formatos de archivos de video para Apple Quick Time™, SGI y Microsoft® AVI.

Una consecuencia de incluir un módulo ICE en el mismo encapsulado del procesador implica la descarga de trabajo sobre el procesador principal aunque no sobre su bus de datos-instrucciones.

Gráficas 3D y el MRE

Si observa bien la figura 3-1 notará que el MRE está en el centro de toda la computadora, por lo que parte del desempeño de la misma depende de este circuito. El MRE incluye una unidad de control muy sofisticada para soportar hasta cuatro bancos de memoria y un total de hasta 2GB en RAM. El MRE permite una conexión casi directa entre el procesador y la memoria del sistema o bien, permite a algún dispositivo externo como el disco duro tener acceso directo a un segmento de memoria para transferencia masiva de datos.

El MRE también incluye circuitos que pueden procesar comandos gráficos propios de OpenGL a modo de tubería gráfica. La pregunta ahora es ¿donde se crea la imagen que se ve en el monitor? La respuesta, en la memoria principal. Así es, la memoria principal sirve como:

- Búfer de estructura.
- Búfer profundidad que almacena la distancia de cada píxel respecto del plano de la pantalla.
- Memoria de texturas.
- Búferes adicionales para máscaras y generación de imágenes en estéreo.

En general, dentro del MRE, se tiene un circuito por cada etapa de la tubería gráfica y algunos de éstos procesan porciones de polígonos de manera paralela en tanto que otras etapas procesan varios polígonos también en forma paralela. Falta algo, las texturas: a través de un trabajo software-hardware es posible aplicar texturas a superficies de polígonos y esto para aumentar el realismo de la imagen. El MRE cuenta también con un adaptador 3D estéreo para su uso con sistemas de realidad virtual. Bueno; en el capítulo dedicado a las tarjetas gráficas veremos cómo funciona este circuito de tubería gráfica.

Tal vez se pregunta la razón de la existencia de este circuito tan complejo, la respuesta implica el deseo de los productores de O_2 en crear un sistema básico que le permita a cualquier usuario iniciar su empresa. Para que esto sea posible el usuario, al menos, debe contar con las herramientas necesarias; O_2 no es un juguete.

Input/Output Engine IOE

En cuanto a procesamiento de vídeo, el IOE controla la entrada, la salida, la sincronización de datos de audio y vídeo y proporciona además funciones de procesamiento de vídeo que normalmente se encuentran en tarjetas opcionales especializadas para audio y vídeo. Estas tarjetas se conocen como *A/V Option Boards*.

Entre las capacidades interconstruidas en el IOE se proporciona un soporte nativo para el procesamiento de vídeo de entrada o salida. Suponga un flujo de datos (o secuencia de datos) que han sido capturados por una tarjeta de video en formato RGB. Durante la captura, se puede escalar la imagen horizontalmente o verticalmente, tal como se haría en una corrección de la relación de aspecto. Un entonces se realiza una conversión espacio-color entre los formatos RGB y YUV 4:2:2[4].

La secuencia de datos que sale del IOE es enviada a la memoria principal en un segmento llamado búfer, el envío se realiza a través de un canal llamado DMA[5]. Se supone que los datos son enviados directamente a memoria sin pasar por algún dispositivo hardware o

[4] Este formato también se conoce como un 4:2:2 YCrCb.
[5] Direct Accesss Memory: Acceso Directo a Memoria.

software; por supuesto el único dispositivo hardware es el que proporciona el servicio DMA. Según se requiera, los datos son almacenados en disco y ensamblados como un fichero de vídeo continuo, o bien, los datos son pasados al ICE, en el procesador principal, para su compresión en algún formato de vídeo.

Ahora bien, la mayoría de las aplicaciones de vídeo por computadora trabajan con una relación de aspecto correspondiente a píxeles cuadrados, sin embargo, las aplicaciones de vídeo profesional trabajan con píxeles no cuadrados. Por esta razón en IOE proporciona soporte nativo para la conversión en tiempo real entre ambos formatos de píxeles, permitiendo a las señales de video ser desplegadas con las relaciones de aspecto correctas.

Display Engine (DE)

El DE es lo que en el entorno de las computadoras personales llamamos el controlador de vídeo. Este circuito tomará la información del búfer de estructura, ubicado en la memoria RAM, para generar las señales eléctricas necesarias como las que alimentan a los DAC y las señales de sincronía para construir la imagen. El DE es capaz de desplegar imágenes de hasta 1280x1024 píxeles en 75Hz. A este subsistema se puede conectar un monitor CRT o bien un panel plano.

Si observa la figura 3-1, el ancho de banda que requiere el DE es de 1.067 GB/s, veamos si es suficiente: para ilustrar los requerimientos del sistema, considera una imagen en color verdadero de 1280x1024 píxeles, lo cual corresponde 3.9MB de datos. Aun cuando 1280x1024 píxeles pueden ser considerados como una imagen relativamente grande, corresponde a la resolución y capacidad de despliegue de la mayoría de las estaciones de trabajo así como también corresponde a la resolución de las cámaras digitales de alto desempeño. Igualmente, dibujar un modelo o escena 60 veces por segundo puede ser considerado demasiado optimista, sin embargo, esto es considerado el umbral bajo el cual el ojo humano no puede distinguir un movimiento discreto. Ahora bien, transferir una imagen entre la memoria principal y el DE a esas velocidades requiere de un ancho de banda de 236 MB/s (3.9MB x 60). Así que el ancho de banda del DE está sobrado o bien, se ha pensado en que pudiera soportar imágenes con una mayor cantidad de píxeles.

SDRAM

La configuración de los circuitos de memoria proporciona un ancho de banda de 2.1GB/s. Ahora, si observa la figura 3-1, notará que el DE sólo requiere de 1MB/s para hacer su trabajo, entonces ¿para qué una memoria con un ancho de banda tan amplio? La respuesta se da a continuación.

Ejemplo

Veamos, entonces, un ejemplo que explote realmente las capacidades de la O_2. Vamos a considerar un escenario virtual. Una persona es filmada contra un fondo verde. Ahora bien, la cámara se ha conectado a una O_2. El IOE realiza entonces la conversión de señales de NTSC a RGB; cada cuadro es enviado entonces a la RAM a través del MRE. Al mismo tiempo, el procesador principal está generando los vértices de un cubo que se tuerce en el tiempo. El MRE entonces toma la información de los vértices y de cada cuadro que se almacena en la RAM, los pasa por la tubería gráfica generando nuevos cuadros que almacena nuevamente en la RAM. La secuencia de cuadros es pasada al ICE para su compresión en formato MPEG–2. El resultado es almacenado nuevamente en la

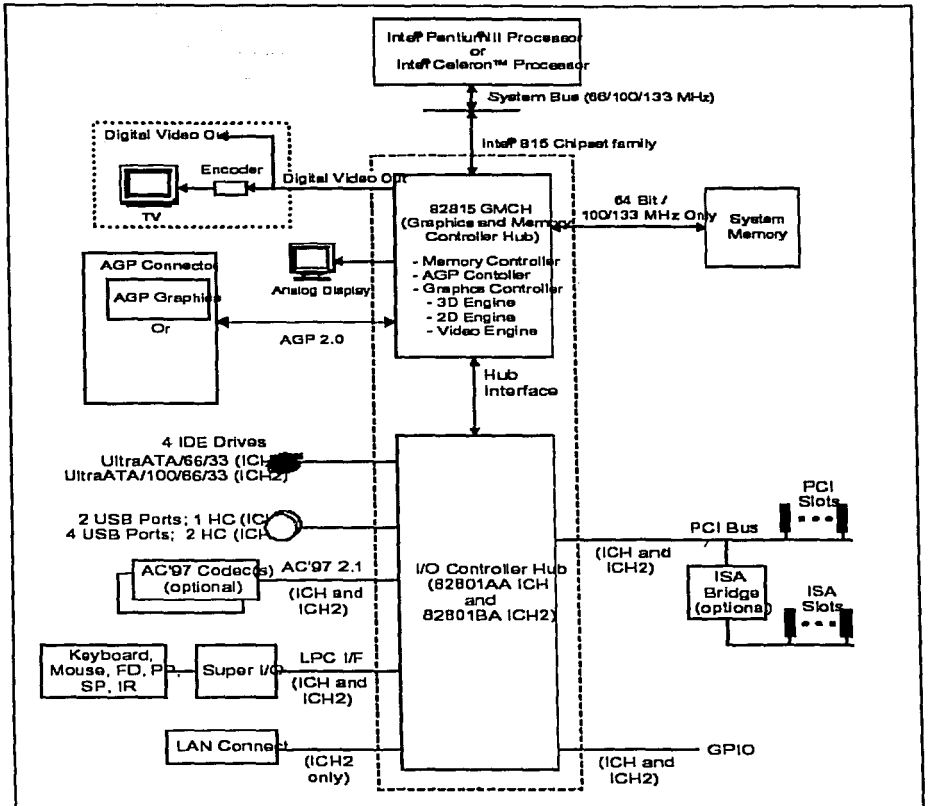


Figura 3-2
Arquitectura de un sistema Intel: Desktop 815 y Pentium III. *"Intel ® 815 chipset family: Graphics and memory controller Hub"*, Intel, Junio del 2000, pg. 15.

RAM de donde se copiaría a un archivo en un disco SCSI. Como puede ver la memoria RAM siempre está siendo requerida, de ahí, su enorme ancho de banda.

¿Cree que este ejemplo pueda hacerse con una PC? La respuesta, aunque es si, aunque el proceso no es tan eficiente como en una O_2.

3-2 Intel

La figura 3-2 nos muestra un diagrama de lo que es una computadora Intel. La figura es muy amplia dado que pretendo mostrar todos los elementos que un usuario puede conectar en el sistema. Veamos ahora que nos ofrece Intel antes de comentar lo referente a la síntesis de imágenes.

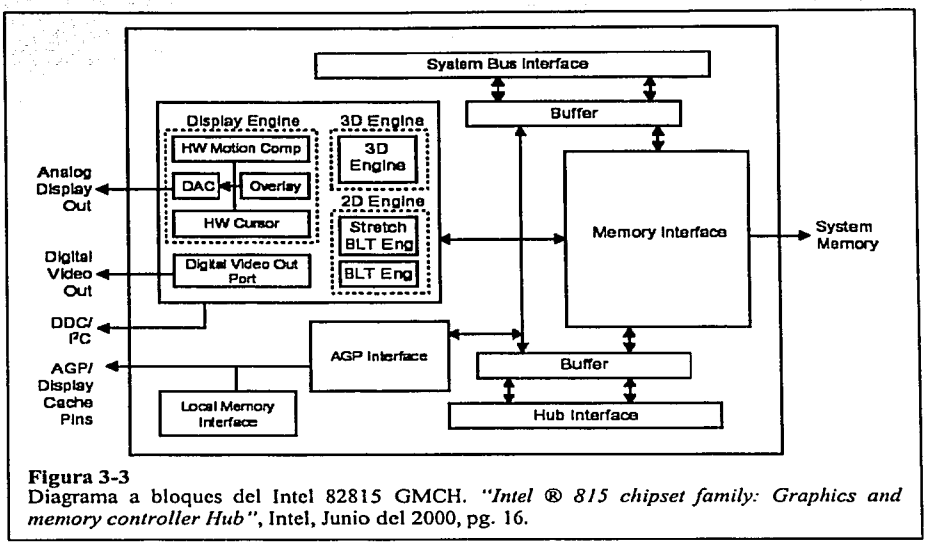


Figura 3-3
Diagrama a bloques del Intel 82815 GMCH. *"Intel ® 815 chipset family: Graphics and memory controller Hub"*, Intel, Junio del 2000, pg. 16.

En el centro de la figura notamos los dos circuitos que forman al *chipset*. Según la figura, hay dos miembros de la familia Intel®815 que son:

- Intel®815 chipset: este chipset contiene al 82815 ® GMCH y al 82801AA ® ICH.
- Intel®815E chipset: este chipset contiene al 82815 ® GMCH y al 82801BA ® ICH2.

Pero que son estos integrados GMCH e ICH, veamos

- El 82815 que es un controlador de gráficos y memoria y que en inglés se escribe como "*Graphics and Memory Controller Hub*" (GMCH) y
- El 82801 que es el controlador de entrada-salida y que en inglés es "*I/O Controller Hub*" (ICH).

El 82815 integra en un solo encapsulado un controlador para caché SDRAM que soporta un arreglo de memoria de vídeo de 32 bits del tipo SDRAM a 133MHz. Suficiente para un buen desempeño en imágenes 2D y gráficas 3D no muy complejas. También se multiplexan una interfaz para un dispositivo gráfico integrado o GPA y una interfaz AGP. Estos dos dispositivos son excluyentes, es decir, cuando conectamos una tarjeta AGP al sistema, el dispositivo gráfico integrado es deshabilitado. La figura 3-3 muestra un diagrama bloques del Intel 82815 ® GMCH.

Interfaz con la memoria del sistema

El GMCH integra un controlador de memoria del sistema que soporta un arreglo de circuitos SDRAM de 100/133MHz. Este controlador de memoria es configurable a través de registros de control. El GMCH soporta DIMMS de 64 bits de ancho con circuitos SDRAM. La forma de direccionar la información queda fuera de alcance del presente escrito, no obstante, nosotros debemos recordar que los circuitos de memoria se leen a manera de libro, es decir, por páginas. El GMCH puede tener acceso a cuatro páginas a la

vez. El GMCH también puede dar servicio hasta a tres circuitos DIMM y la detección del tipo de memoria y su método de acceso queda a cargo del BIOS.

Multiplexión del AGP y del GPA

El 82815 GMCH multiplexa una interfaz AGP con otra interfaz que Intel ha llamado GPA (*Graphics Performance Architecture*) y que se emplea con un dispositivo gráfico dentro del mismo encapsulado. Sólo una de las dos interfaces puede estar activa y esto se detecta a través del BIOS que programa de los registros del GMCH. En la figura 3-3 podemos notar cinco elementos importantes que conforman al GPA

- 3D Engine
- 2D Engine
- Display Engine
- Analog Display Out
- Local Memory Interface

Hay otros elementos, sin embargo, éstos nos bastan para identificar al dispositivo gráfico interno. Obviamente, estos 5 circuitos no son necesarios cuando se dispone de una tarjeta gráfica AGP externa. Debo aclarar que la existencia de GPA equipara una computadora Intel con una O_2. Acaso Intel ¿estará siguiendo el ejemplo de Silicon Graphics? De manera adicional, el que un sistema opere con GPA implica una relación costo-rendimiento que le hace atractivo ante las tarjetas gráficas más sofisticadas.

Normalmente, Intel coloca 4Mbytes en la placa base que llama *display cache* a modo de memoria de video que se usa más bien como búfer de profundidad o búfer Z. Cuando el usuario decide no instalar una tarjeta AGP, la memoria del sistema sirve entonces como búfer estructura y también como búfer de texturas y otras informaciones.

Interfaz AGP

En este capítulo no nos dedicaremos a revisar las distintas especificaciones y sus diferencias, tan sólo veremos los fundamentos de su operación y en un capítulo posterior revisaremos un poco más de esta especificación. Por lo pronto, veremos que existen dos especificaciones: AGP 1.0 y AGP 2.0. que pueden diferenciarse por su voltaje de operación, entre otros detalles.

- 3. 3V drive: Este modo es compatible con las especificaciones AGP 1.0 y 2.0.
- 1.5V drive: Este modo es compatible con especificación AGP 2.0.

La siguiente tabla muestra la relación de los niveles de voltaje con la velocidad de transferencia de información.

Velocidad	voltaje de la señal	
	1,5	1,3
1x AGP	Si	Si
2x AGP	Si	Si
4x AGP	Si	No

Más adelante, como ya comenté, nos introduciremos a la forma en que trabaja la especificación AGP 2.0. No será información muy técnica puesto que también sale del

alcance del presente escrito, sin embargo será suficiente para darnos una idea de su operación.

Interfaz Hub

Por tradición, los diseñadores de circuitos apodan Hub a aquellos que debe llevar una carga muy pesada de información. Esta interfaz Hub es un circuito que se comunica con el Hub del ICH. Intel la define como una conexión de carácter privado.

Soporte integrado para gráficas en el 82815 GMCH

Como ya he venido mencionando, el GMCH incluye dentro del encapsulado un acelerador gráfico. Su arquitectura consiste de máquinas[6] que realizan, cada una, operaciones en paralelo y cuyo resultado son gráficas 2D y 3D y compensación de movimiento para vídeo MPEG. Las máquinas 2D y 3D son manejadas por un preprocesador que al implementar una tubería gráfica permite un flujo sostenido de datos gráficos que serán finalmente desplegados.

Aunque más adelante hablaré sobre este tipo de sistemas gráficos, voy a mencionar que algunas de las etapas de la tubería del acelerador gráfico trabajan sobre distintas primitivas en paralelo, en tanto que otras trabajan en paralelo sobre porciones de la misma primitiva.

La máquina aceleradora gráfica de GMCH incluye soporte para la mapeo de texturas, para visión en perspectiva, filtrado Mip-Map, filtrado anisotrópico y trilineal, sombreado Gouraud, mezclas con el canal alfa, niebla y búfer de profundidad. Tiene también programadas otras funciones 3D compatibles con el estándar DirectX de Microsoft.

El GMCH también puede emplear el *display cache* (DC) sólo como búfer de profundidad, los búferes de textura y de despliegue están localizados en la memoria del sistema. Si el *display cache* no es empleado, el búfer de profundidad está localizado en la memoria del sistema.

El acelerador gráfico incluye a su vez capacidades 2D. Algunas de ellas están implementadas en las máquinas BLT y la STRBL ¿no es agradable trabajar con abreviaturas de nombres? Bueno; estas máquinas realizan operaciones lógicas y aritméticas con mapas de bits para generar efectos especiales. Algunas de las habilidades son mover una imagen sobre otra, componer imágenes, etc. Un hardware especial dibuja en la pantalla un cursor; a esta máquina le encontramos en la figura 3-3 como *hardware cursor*. El alto desempeño de las máquinas BLT de 64 bits proporciona una aceleración hardware para las operaciones más comunes en la construcción de ventanas en la pantalla.

La RAM

Así como en la O2, la PC Intel requiere de bancos de memoria, con las siguientes características:

[6] El término máquinas se traduce de la terminología anglosajona *engine* que Intel emplea para referirse este circuito.

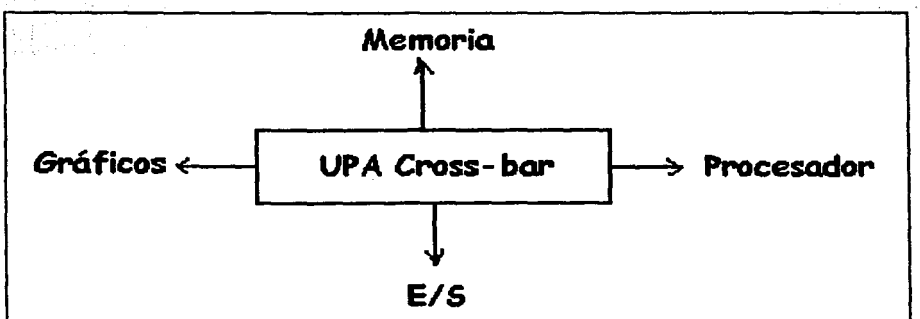


Figura 3-4
Concepción de una estación de trabajo según Sun Microsystems. *"UPA-Sun's High Performance Graphics Connection. Technical White Paper"*, Sun Microsystems, 1999.

- Las frecuencias de operación son de 100/133MHz con anchos de banda de 0.81/1.06GB/s.
- Soporta hasta tres bancos de circuitos y en total de 512MB de almacenamiento.

Si compara ésos datos con los respectivos de la O_2, la PC aún está a la mitad del camino, por así decirlo. Además, ¿512MB de RAM? Cualquier otra empresa de PC como ASUS o AOPEN fabrican placas base que soportan hasta 1. 5MB, es curioso ver a la empresa que ha impulsado el mundo del PC quedarse atrás ante la competencia, pero así son los negocios.

3-3 Sun Microsystems

Desde sus inicios, la empresa siempre se ha caracterizado por su muy particular definición de software y hardware, tal vez por las diferentes nacionalidades de sus tres creadores. Hoy día, la empresa sostiene contratos con diversas universidades: Sun Microsystems proporciona hardware actualizado y las universidades proporcionan cerebros.

Sun Microsystems ha diseñado una familia de estaciones de trabajo a partir de un concepto denominado *Ultra Port Architecture* o bien UPA, en español es *Arquitectura de Puerto Ultra.*

La figura 3-4 nos muestra que es lo que la gente de Sun Microsystems entiende por estación de trabajo. Como puede ver, hay cinco elementos en su tabla periódica: gráficos, procesador, entrada-salida, memoria y el quinto elemento comunica a los primeros cuatro.

La figura 3-5 nos muestra un diagrama que representa a la arquitectura UPA con su muy particular representación de líneas de bus. Cuatro de los cinco elementos de la computadora son fácilmente reconocibles, el quinto elemento es la esencia de lo que es la UPA. El elemento central o quinto se llama *UPA Cross-Bar Memory Interconnect.*

Aunque Sun Microsystems no proporciona un diagrama de la arquitectura del *UPA Cross-Bar* si nos da a entender que en su interior hay un circuito que se encarga de comunicar los periféricos que tiene conectados.

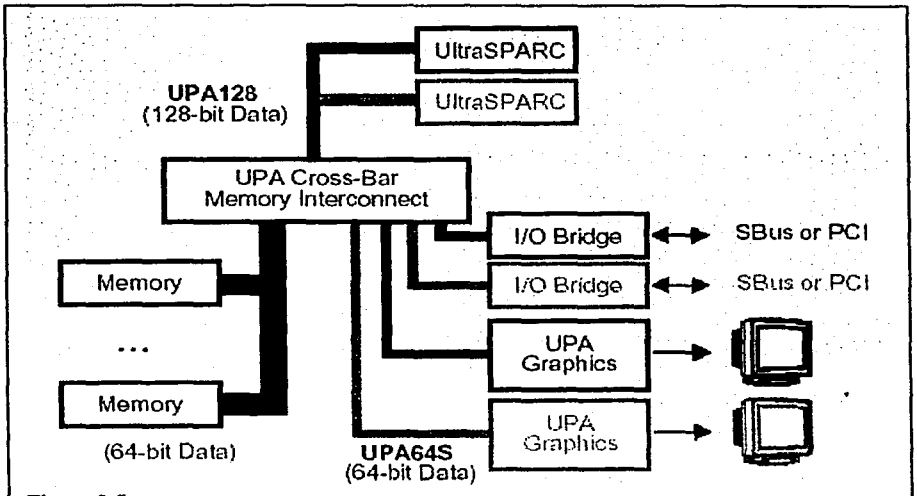


Figura 3-5
Diagrama a bloques de la arquitectura de un sistema Sun Microsystems. Esta arquitectura se denomina UPA o Ultra Port Architectura, según la bibliografía anglosajona. *"UPA-Sun's High Performance Graphics Connection. Technical White Paper"*, Sun Microsystems, 1999.

UPA se diseña con el propósito de proporcionar un sistema de bajo costo con habilidades gráficas 2D y 3D. Una tarjeta gráfica simple sería suficiente entonces ya que todas las operaciones pueden ser realizadas por el CPU del sistema. A pesar de esta cualidad Sun Microsystems mantiene contacto con proveedores de tarjetas gráficas que incluyen procesadores matemáticos que realizan todos los cálculos involucrados en síntesis imágenes en lugar del CPU.

La memoria

Veamos ahora las características de sus circuitos de memoria. Los circuitos que soporta son del tipo SDRAM PC 133 en configuración DIMM, con un bus de datos de 64 bits, con lo que el ancho de banda alcanza algo más de 2.0GB/s. Cualquiera de sus sistemas puede soportar hasta cuatro módulos de 128, 256 y 512MB. En total, podemos dotar este equipo hasta con 2GB de RAM.

El procesador

Sun Microsystems proporciona un potente procesador de 64 bits a 500MHz. El procesador contiene un conjunto de instrucciones para el manejo de gráficos en dos y tres dimensiones no obstante que la tarea se reserva a una potente tarjeta gráfica. El ancho de banda del bus que comunica al procesador con el *UPA Cross-Bar* varía de 1.0 GB/s (año de 1994) a 1.92 GB/s (año de 1998) según el año que se fabricó el modelo.

El CPU, conocido como UltraSparc, no es diferente de los demás, proporciona dos tuberías para operaciones con enteros otras dos para operaciones en punto flotante y una tubería para operaciones de carga y descarga de los registros del micro y otras operaciones. Tiene dos memorias caché de nivel 1, esto es, un caché para datos y un

caché para código. Cada caché soporta hasta 16 KB. En tanto, el caché de nivel 2 soporta hasta 256 KB entre código y datos.

Este CPU también soporta la decodificación y ejecución dinámica de sus instrucciones, o sea, las instrucciones se ejecutan tan pronto sus operandos están listos sin importar su orden. Ahora bien, apenas las instrucciones salen del proceso de tubería, un circuito se encarga de sincronizar los resultados de cada operación.

Las instrucciones multimedia que soporta el CPU UltraSparc involucran actividades como la graficación en dos y tres dimensiones, procesamiento de imágenes y la codificación y decodificación de vídeo MPEG en tiempo real.

UPA Graphics

Los ingenieros de Sun Microsystems han diseñado un bus denominado UPA64S que tienen bus de 64 bits y un ancho de banda de 960 MB/s. Por supuesto, sabemos que este ancho de banda es mucho más que suficiente.

Veamos ahora, 960 MB/s representa el pico de transferencia de información; sin embargo, el diseño del *UPA Cross-Bar* es tal que permite un valor sostenido de transferencia de información de más del 85% del valor pico.

3-4 En conclusión

Hemos revisado los aspectos fundamentales de tres equipos de cómputo de diferentes empresas: dos estaciones de trabajo y una PC. Los aspectos que, en general, debemos considerar son:

- El espacio de direcciones al que tiene acceso el CPU.
- El camino que sigue información, y en nuestro caso particular la información gráfica.
- Las capacidades del procesador y no sólo del CPU, para la síntesis y el procesamiento de imágenes.
- Los circuitos de RAM.

En cuanto a los CPU, todos comparten el principio de tubería o bien, como yo le llamo, la línea de montaje. La arquitectura Harvard[7] se puede ver en el uso de las memorias caché de nivel 1, la ejecución dinámica, la predicción de saltos, instrucciones especializadas para soporte de graficación 2D y 3D, el procesamiento de imágenes, codificación y decodificación de vídeo, etc.

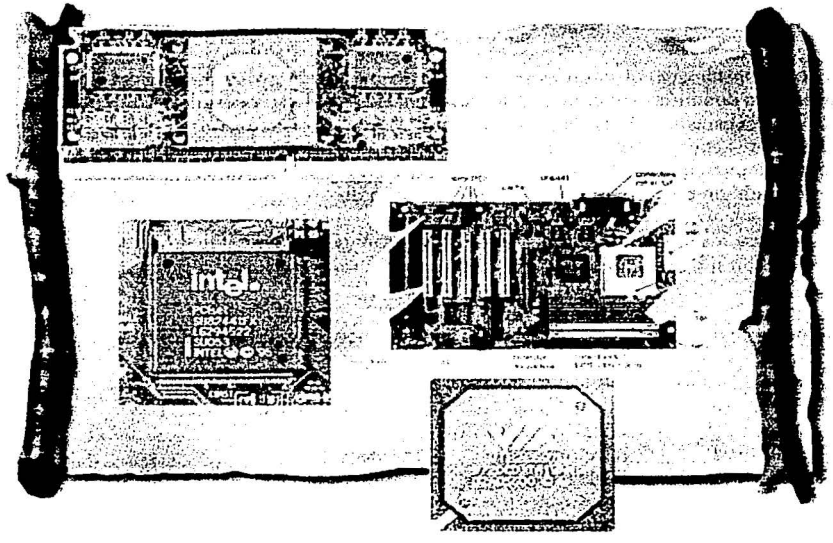
Las diferencias fundamentales entre equipos pueden ser:

- Una PC requiere de un procesador con bus de datos de 32 bits en tanto que una estación de trabajo requiere de otro procesador de 64 bits en un bus de datos.
- Los circuitos que se colocan en el mismo encapsulado que el CPU.

Algo que se ha quedado fuera de nuestras observaciones es la capacidad de almacenamiento masivo, es decir, los discos duros, en general, en una estación de trabajo

[7]Esta arquitectura se caracteriza que tiene un circuito de memoria del cual el CPU toma los comandos a ejecutar y en tener otro circuito de memoria en el que se almacenan los datos.

se prefiere la tecnología SCSI por su ancho de banda adecuado en la captura de vídeo. Las estaciones de trabajo no desprecian la tecnología IDE que actualmente fabrica los discos duros de mayor capacidad, buen rendimiento y bajo costo.



Capítulo 4 Los microprocesadores

Veamos, en 1947 **John Bardeen, William Shockley** y **Walter Brattain** entregan a sus jefes en ***Bell Laboratories*** un pequeño objeto que podría describirse como una pequeña cabeza con tres patas largas y que llamaron ***Transfer Resistor***.

Tal vez Shockley, a diferencia de sus jefes, pudo ver el potencial de su transistor, ya que decidió emprender su propio camino fundando una empresa.

Algunos de los empleados de William Shockley vieron su oportunidad de éxito, así que renunciaron y formaron su propia empresa, la ***Fairchild Semiconductor***. A su vez, otros empleados de Fairchild también vieron su oportunidad así que, dirigidos por **Ted Hoff,** formaron ***Intel***.

El resultado de esta descendencia entre 1947 y 1965 es el desarrollo de procedimientos para colocar en un solo encapsulado un circuito formado por muchos transistores.

En 1969, Ted Hoff, fundador de Intel, presenta su diseño de un microprocesador a una compañía fabricante de calculadoras japonesas. Este primer microprocesador era el **4004.** En 1971 Intel presenta su diseño para el **8008.**

Y la historia sigue ...

Como puede ver, la historia de las computadoras es muy interesante, no obstante, no es el objetivo del presente trabajo, esto es tan solo un homenaje a las personas que fundaron las bases de nuestro presente, después de todo, a quien no le gusta ser recordado. En Donald

H. Sanders, "*Informática presente y futuro*", puede encontrar mayor información al respecto.

La historia nos muestra que el desarrollo del circuito integrado no se detiene, así, del microprocesador se continúa con el microcontrolador, el DSP, los procesadores de video y otros de propósito especial. A su vez, las aplicaciones que requieren estos circuitos son muy variadas. En general, nuestro interés está en la síntesis de imágenes y en consecuencia en la simulación de sistemas físicos. Un ejemplo clásico sería la simulación de una cirugía donde un estudiante practica la extirpación de un segmento del hígado.

Pero ¿qué caracteriza a un microprocesador moderno? ¿Qué partes lo constituyen? Bueno; en este capítulo verificaremos principalmente la forma en que se procesan las instrucciones.

A continuación ofrezco una lista de fabricantes de microcircuitos para una u otra aplicación.

Phillips	Samsung	Cirrus Logic	3Dlabs (Permedia)
Intel	Motorola	Texas Instruments	Mips
ADLIB	Diamond	Hércules	Logitech
Matrox	Quantum 3D	Trident	Tseng
Video Logic	SiS	NewCom	

4-1 Von Newmmann Contra Harvard

Entre 1939 y 1944 **Howard Aiken**, dirigiendo un equipo de ingenieros entusiastas de la Universidad de Harvard y de IBM crearon una computadora que hoy día se considera la primera computadora digital. Esta máquina consistía de muchas calculadoras, las cuales trabajaban en partes del mismo problema bajo la guía de una unidad de control. Las instrucciones eran leídas de una cinta de papel y los datos eran proporcionados en tarjetas perforadas. Este equipo realizaba trabajos sólo en el mismo orden en que los recibía. Esta máquina se llamó **Mark I**.

Entre 1940 y 1950 **John von Neumann** y un equipo de entusiastas de la Universidad de Pensylvania se dedicaron a la tarea de construir una computadora con características innovadoras. Dos de estas características las listo a continuación:

- Una memoria de almacenamiento: Esta memoria podía almacenar código y datos.
- Una unidad central de procesamiento, la cual coordinaría las funciones del equipo.

Como referencia, esta computadora se llamó **EDVAC** (*Electronic Discrete Variable Automatic Computer* o Computadora Electrónica Automática de Variable Discreta) y serviría de modelo para computadoras durante 40 años.

Hasta aquí la historia, consideremos ahora una característica que distingue a la MARK I de la EDVAC. Me refiero a la forma en que se almacenan los datos y el código del programa. En la MARK I, las instrucciones se suministran al procesador en tiras de papel, en tanto que los datos se alimentan con tarjetas perforadas. En EDVAC, las instrucciones y los datos residen en el mismo almacenamiento (ya se habían desarrollado sistemas de memoria) y el procesador los toma de éste. En conclusión, la técnica de almacenar las

instrucciones y los datos es la característica más importante que distingue a ambas arquitecturas.

Ahora la pregunta obligada ¿Por qué los microprocesadores modernos emplean la arquitectura Harvard en vez de la von Neumann? En realidad los microprocesadores modernos mantienen una solución que es el híbrido de ambas arquitecturas. Las instrucciones y los datos residen en la RAM, sin embargo, al pasar al caché de primer nivel, las instrucciones se almacenan en su caché de código y los datos en su caché de datos.

Veamos, en la arquitectura von Newmann, ilustrada en la figura 4-1.a, el procesador recibe las instrucciones periódicamente hasta que debe recuperar un dato de la memoria, en este momento el flujo de instrucciones se interrumpe para leer el dato. Leído el dato, el flujo de instrucciones se reactiva. En la arquitectura Harvard, ilustrada en la figura 4-1.b, el procesador recibe instrucciones periódicamente hasta que debe recuperar un dato de la memoria, en este momento, el flujo de instrucciones **no** se interrumpe ya que el dato se encuentra almacenado en otro lugar. La conclusión implica que la arquitectura Harvard puede ejecutar más instrucciones por unidad de tiempo o bien, tiene un ancho de banda más amplio.

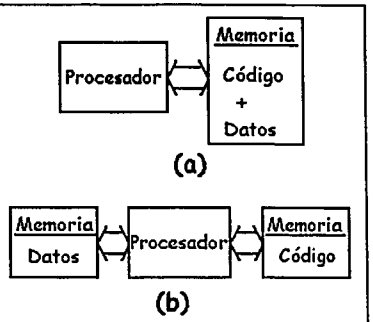


Figura 4-1
Arquitecturas de procesadores. La arquitectura von Neumann, representada en (a), se caracteriza por almacenar sus datos y código de programa en el mismo bloque de memoria en tanto que la arquitectura Harvard, representada en (b) se caracteriza por almacenar sus datos en un bloque de memoria distinto del de código.

4-2 El proceso de la tubería

Comencemos nuestro viaje preguntándonos ¿qué es una tubería? El proceso de tubería en un microprocesador se parece mucho a la línea de montaje de vehículos de Henry Ford (1863-1947). Tanto la tubería como la línea de montaje se caracterizan en su división del trabajo en etapas. En la línea de montaje, apenas el futuro auto ha salida de una etapa, otro entra en la misma. En la tubería es igual, apenas una instrucción deja una etapa, otra entra.

Probemos un ejemplo, en el procesador i486 de Intel, el procesamiento de una instrucción sigue cinco etapas:

PF Prefetch
D1 Decode 1
D2 Decode 2
EX Execute
WB Write Back

En la etapa *prefecth*, se trae un comando de la memoria al procesador. Una vez en el procesador, el comando entra en la segunda etapa, la *Decode 1*, que es la primera fase de la decodificación de la instrucción. El objetivo en este caso es analizar el comando y con ello determinar el tipo de acción que se debe emprender. Dependiendo del tipo de comando, a continuación, se deben determinar los operandos del mismo. Esta misión está encomendada a la segunda fase de la decodificación de la instrucción, denominada *Decode 2*. En la siguiente etapa, llamada *Execute*, es donde se lleva a cabo la ejecución del comando y el consiguiente acceso a memoria. En la etapa *Write Back* concluye la ejecución del comando para lo que el contenido de los registros del procesador y los registros de estado internos son llevados a su nueva situación.

La figura 4-2 ilustra el paso de instrucciones a través de una tubería. Note que las instrucciones requieren de cinco pulsos para completar su paso por todas las etapas, pero como la línea termina la ejecución de una instrucción distinta en cada pulso, aparentemente las instrucciones sólo precisan un pulso para su ejecución.

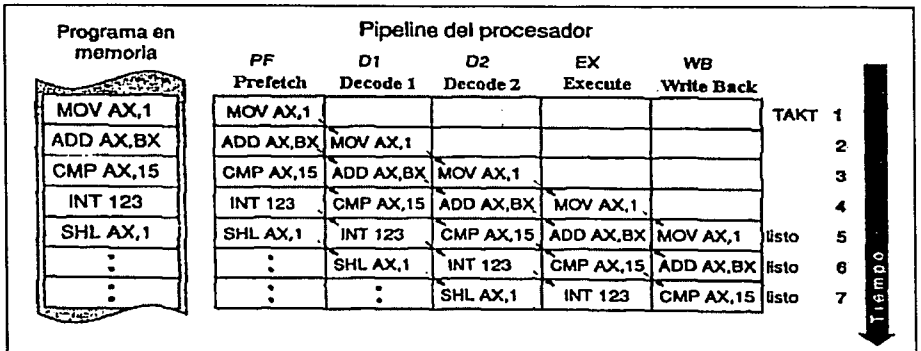


Figura 4-2
Tubería de un procesador i486 de Intel. La figura representa el tratamiento de una lista de instrucciones empleando el concepto de tubería de ejecución.

Saltos y ejecución especulativa

Considere lo que ocurre cuando el procesador debe ejecutar un comando de salto. En este caso, la ejecución del programa no continúa con la siguiente instrucción, sino con otra completamente diferente.

Cuando la instrucción de salto llega a la etapa de ejecución y es ejecutada, ocurre entonces que el procesador detiene la ejecución de instrucciones que se encuentran en la tubería y vuelve a cargar ésta con nuevas instrucciones. En este caso, la ejecución de una instrucción de salto precisa de dos a tres pulsos de reloj o ciclos de tubería.

Los saltos descritos en el párrafo anterior son del tipo no condicionados, sin embargo también existen los saltos condicionados, es decir, si tengo un grupo de instrucciones que deben ejecutarse de manera cíclica, continuarán su ejecución hasta que una condición se satisfaga. El que la condición se cumpla implica saltar a otro punto del programa.

Una forma de reducir el número de pulsaciones para un salto es almacenar en un búfer (*Branch Target Bufer*) la última instrucción de salto junto con su dirección de destino: fíjese bien, la dirección de destino no es siempre la que se proporciona como operando del comando, sino que es una dirección que resultó de evaluar el último comando de salto. Cuando una nueva instrucción de salto entra en la etapa de decodificación, se compara con la instrucción almacenada en el búfer, si son iguales, el procesador, en vez de cargar la instrucción que sigue a la instrucción de salto, modificará su contador de programa cargando la dirección de destino y así cargará la instrucción correspondiente: esto es lo que se llama **ejecución especulativa**. Adicionalmente, se conserva una copia de los registros del procesador que indican su estado antes del salto.

En caso de que el resultado de la condición no sea favorable, como muy tarde se determina en la etapa de ejecución, el procesador detiene la ejecución de las instrucciones que ya se encuentran en la tubería y regresa al estado que tenía antes del salto para continuar la ejecución de las instrucciones que corresponden. La consecuencia final implica que se perderán de dos a tres pulsaciones dependiendo del procesador.

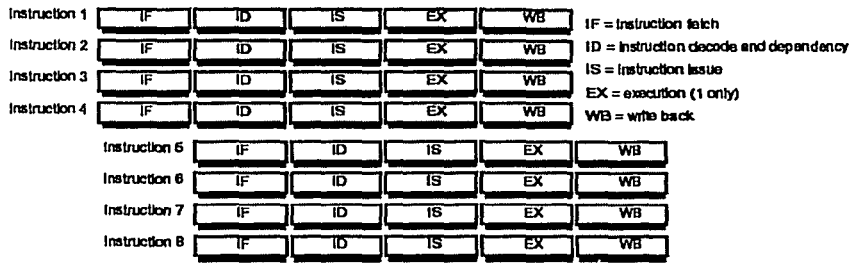


Figura 4-3
Estructura de una arquitectura superescalar de 4 vías. En cada etapa, cuatro instrucciones son manejadas en paralelo. Mips technologies Inc., "*Mips Microprocessor, User's manual*", Version 2.0, 1996, pg. 3.

4-3 ¿Qué es un procesador superescalar?

Pues es uno que busca, decodifica, ejecuta y completa más de una instrucción en paralelo. La figura 4-3 nos ilustra la estructura de una tubería superescalar de 4 vías. En cada etapa, cuatro instrucciones son manejadas en paralelo.

Como ejemplo, veamos al procesador MIPS R10000, el cual se instala en cualquier equipo Silicon Graphcis. La figura 4-4 muestra la tubería del R10000, note su extrema complejidad.

El R10000 tiene las siguientes características:

- Éste implementa instrucciones de 64 bits correspondientes a la familia de procesadores MIPS IV. Al conjunto de instrucciones se le conoce como ISA O *Instruction Set Architecture.*
- Éste puede decodificar hasta cuatro instrucciones en cada ciclo de tubería. Esto se logra acomodando cada instrucción en una de tres pilas.
- Tiene cinco tuberías de ejecución conectadas a las respectivas unidades de ejecución para datos enteros, en punto flotante y cálculo de direcciones.
- Emplea un método de **suministro dinámico de instrucciones** (*dynamic instruction scheduling*) que también se conoce como **ejecución fuera de orden** (*out-of-order execution*).
- También se recurre al **suministro especulativo de instrucciones** (*speculative branching*).
- Dentro del microcircuito se implementa un modelo de excepciones para interrumpir las tareas actuales del micro y atender otras que lo requieran.
- Empleas dos memorias caché de primer nivel: una es para datos y otra es para instrucciones.
- Se ha construido en el mismo microcircuito un controlador de memoria caché de segundo nivel.
- También existe un circuito controlador que funciona de interfaz para la conexión con otro procesador y permitir realizar tareas en forma paralela.

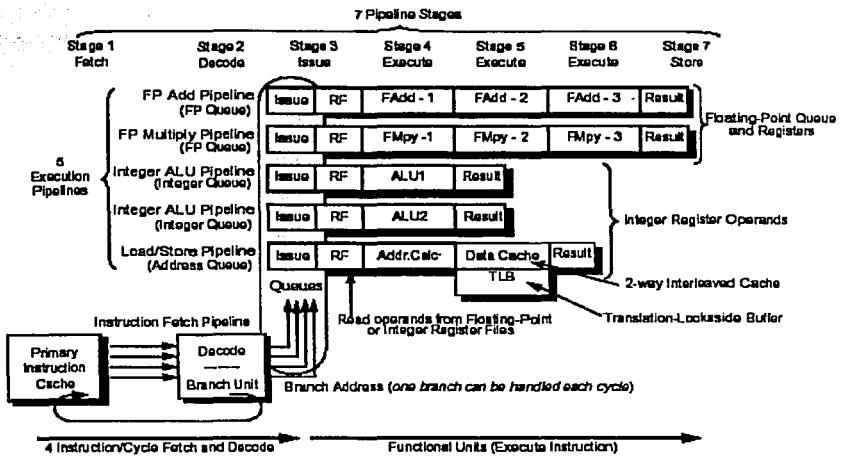


Figura 4-4
Arquitectura de tubería superescalar en el R10000. Mips technologies Inc., "*Mips Microprocessor, User's manual*", Versión 2.0, 1996, pg. 5.

El procesador R10000 es implementado empleando una tecnología CMOS VLSI de 0.35 micrones sobre una pastilla de 17 mm por 18 mm. El resultado es un circuito con 6.7 millones de transistores, incluyendo 4.4 millones de transistores para la caché de nivel 1.

R10000 tubería superescalar

El procesador superescalar R10000 busca y decodifica cuatro instrucciones en paralelo en cada ciclo de la tubería. Cada tubería incluye etapas para búsqueda (etapa 1 que se muestra en la figura 4-4), decodificación (etapa 2) suministro de instrucciones (etapa 3), obtención de los operandos (etapa 3), ejecución de instrucciones (etapas 4 a 6), y almacenamiento de resultados (etapa 7).

Pilas de instrucciones

Como se muestra en la figura 4-4, cada instrucción decodificada en la etapa 2 es añadida a una de las tres pilas de instrucciones:

- Pila de enteros
- Pila de direcciones (que se emplea para calcular direcciones)
- Pila de punto flotante

Cada pila tiene espacio para 16 entradas y las instrucciones no se acomodan en un orden en particular.

Tuberías de ejecución

Cada una de las tres pilas de instrucciones pueden suministrar una nueva instrucción por ciclo a cada una de las cinco tuberías de ejecución:

- La pila de enteros suministra instrucciones a las dos tuberías de la ALU de enteros.
- La pila de instrucciones suministra una instrucción a la tubería de la unidad Load/Store.
- La pila de punto flotante suministra instrucciones a las tuberías de las unidades de multiplicación y suma.

Se dice que existe una sexta tubería, la tubería de búsqueda, lectura y decodificación de las instrucciones, por supuesto se refiere a las etapas 1 a 3.

Tubería de la ALU de enteros de 64 bits

- Tiene una pila de instrucciones con 16 entradas. Cada entrada es de 64 bits.
- Las dos tuberías son atendidas por dos ALU de 64 bits:
    - ALU 1 contiene una unidad lógico-aritmética para realizar sumas. También hay registros de corrimiento y un comparador de datos numéricos de tipo entero.
    - ALU 2 contiene una unidad lógica-aritmética que entre otras funciones implica un multiplicador y un divisor de datos numéricos de tipo entero.

Tubería de punto flotante de 64 bits.

- Esta tiene una pila con 16 entradas.
- La primer tubería es atendida por una unidad de 64 bits para sumas, restas y operaciones diversas en punto flotante. Una suma se realiza en tres ciclos de tubería.
- La segunda tubería es atendida por una unidad de multiplicación paralela de 64 bits. Esta unidad también realiza operaciones para copia de información (instrucciones del tipo *move/load*). La tubería puede realizar una multiplicación en tres ciclos de la tubería.
- La segunda tubería también tiene interconectadas unidades separadas para divisiones y raíces cuadradas de 64 bits.

Tubería de Load/Store

- Tiene una pila con 16 entradas.
- Emplea un registro que sirve para los modos de direccionamiento base e índice.
- Esta tiene una unidad de cálculo de direcciones virtuales de 44 bits.
- Tiene una Translation Lookaside Buffer (TLB) que es una tabla con 64 entradas y que sirve para convertir direcciones de memoria virtual a direcciones de memoria física.

En el procesador MIPS, dadas las exigencias de multitarea, se requiere del manejo de la memoria virtual. La memoria física, para este fin, se divide en páginas o bloques que van de 4KB a 16 Mbytes.

Suministro dinámico de instrucciones

Durante la etapa 2, las instrucciones son almacenadas en tres pilas y entonces se determina cualquier dependencia entre ellas: si el resultado de una instrucción es el operando de la siguiente, o bien, si son instrucciones del mismo tipo. Si alguna o varias de las instrucciones tiene listos sus operandos, serán suministradas, cada una, a su respectiva tubería de ejecución sin importar cual llegó primero.

Puesto que las instrucciones salen de la tubería de ejecución también en orden diferente al que fueron suministradas, se hace necesario un mecanismo que permita ordenar los resultados de cada una. Este mecanismo implica una tabla de registros que guarda una copia del contador de programa para cada instrucción en la tubería de ejecución.

Ejecución especulativa

El concepto de ejecución especulativa ya lo he mencionado arriba y en casi todos los procesadores se implica la misma mecánica.

4-4 La Memoria Caché del Procesador

Debido a la gran velocidad alcanzada por los microprocesadores, la RAM del ordenador no es lo suficientemente rápida para almacenar y transmitir los datos que el microprocesador necesita, por lo que tendría que esperar a que la memoria estuviera disponible y así el trabajo se ralentizaría. Para evitarlo, se usa una memoria muy rápida, estratégicamente situada entre el micro y la RAM: la memoria caché. ¿Cuánto es "muy rápida"? Bien, unas 5 o 6 veces más que la RAM. Esto la encarece bastante, claro está, y ése es uno de los motivos de que su capacidad sea mucho menor que el de la RAM.

Vamos a dar un poco más de detalle. Cuando un ordenador trabaja, el micro opera en ocasiones con un número reducido de datos, pero que tiene que traer y llevar a la memoria en cada operación. Si situamos en medio del camino de la información una memoria intermedia que almacene los datos más empleados, los que casi seguro necesitará el micro en la próxima operación que realice, se ahorrará mucho tiempo del tránsito y acceso a la lenta memoria RAM; esta es la segunda utilidad de la caché.

Existe una pregunta que seguramente nos hemos planteado y que quizá no nos hayamos atrevido a hacer ¿por qué existen la memoria caché de nivel 1 y la caché de nivel 2? Ya sabemos porque existe la caché de nivel 2, ahora bien, recuerde que la caché de nivel 1 se divide en dos bancos; uno se emplea para contener los comandos a ejecutar y el otro banco almacena los datos. Este arreglo es lo que conocemos como la arquitectura Harvard y es una forma eficiente de suministrar información al procesador.

Es común que los circuitos de memoria caché sean del tipo SRAM (Static RAM) que es un circuito de memoria que permite un acceso a sus celdas en menor tiempo que una DRAM.

Efectividad de un caché

La calidad y efectividad de un caché se mide con la relación entre ***cache hits*** y ***cache mises***. Se habla de un *cache hit* o presencia cuando los datos requeridos por el procesador están disponibles en la caché y por ello se impide tener que acceder a la memoria

conectada a continuación, que resulta mucho más lenta. Un *cache miss* o ausencia significa que los datos no se encuentran presentes en la caché y por ello deben ser cargados primero en la caché desde la memoria conectada a continuación, antes de poder ser transmitidos al procesador. Cuanto mayor sea el número de aciertos en comparación con el número de ausencias, más a menudo podrá el procesador servirse de la memoria rápida y, por consiguiente, podrá trabajar más rápido.

Existen dos aspectos fundamentales en la utilización eficiente de una memoria caché:

- La estrategia de la caché, por lo que se refiere a los accesos de lectura y escritura y,
- La arquitectura de la caché, es decir, la manera en que se guardan las informaciones dentro de la caché.

Estrategias del caché

Adelantándonos un poco, cuando el CPU requiere un dato de la memoria, primero se verifica si está en la caché, de no ser así, un dato será copiado de la memoria a la caché y luego al procesador. Ahora bien, cuando se habla de accesos escritura de la CPU hacia la caché hablamos de

- Write Through o escritura directa
- Write Back o escritura posterior

Para simplificar la explicación de estas dos estrategias permítame considerar por ahora que solo existe un nivel de memoria caché, así que:

En el caso de Write Through, cuando el procesador requiere el dato de una variable, ésta se copia de la memoria a la caché y luego, de la caché al procesador, por lo que tenemos tres copias de una variable. Si el valor de la variable es modificado por el procesador, entonces, las copias de la variable en la caché y en la memoria principal son actualizadas; esto es lo que se llama coherencia. Esta estrategia de caché es muy empleada en sistemas multiprocesador.

En Write Back, también tenemos tres copias de la variable, una en memoria principal, otra en la caché y otra en el procesador. Cuando el procesador modifica la variable, ésta se modifica solo en la memoria caché hasta el momento en que, por un nuevo acceso de lectura, la CPU precisa espacio libre en la caché. Veámoslo de otra forma para precisar la utilidad de la estrategia. Si una posición de memoria caché cambia su contenido a menudo, con esta estrategia se ahorran los relativamente lentos accesos de escritura a la memoria principal, al menos hasta el momento en que el dato deba abandonar la caché. La estrategia Write Back, como posiblemente habrá notado, es muy empleada en sistemas monoprocesador.

Arquitectura de una caché

Normalmente, las memorias caché se organizan en las denominadas *Cache Lines* o **líneas de caché**. Una línea de caché es un registro que puede almacenar una cierta cantidad de Bytes. La anchura de cada línea, en Bytes, es igual al ancho de bus de datos del CPU. Por ejemplo, el 8036 tiene una anchura de 32 bits (un DWord o 4 Bytes), el 80486 tiene un bus de 128 bits (4 DWord o 16 Bytes) y en los pentium y otros procesadores modernos, se tienen 256 bits de ancho de bus de datos (4 QWORD o 32 Bytes).

Cuando el procesador realiza un acceso de lectura, de la memoria se copian los Bytes contiguos necesarios para llenar una línea de caché, aún cuando el CPU solo haya reclamado un Byte. El hecho de que se deba leer un bloque de Bytes contiguos de la memoria a la caché aprovecha el *Burst mode* o **modo ráfaga** de los circuitos de memoria; veamos entonces como funciona. El 80486 de Intel normalmente necesita dos pulsos para la lectura de un DWord desde la memoria, por lo que, para llenar una línea de caché son precisos 4x2 pulsos. Por el contrario, en el modo ráfaga sólo se requieren dos pulsos para el primer DWord y un pulso para cada uno de los tres siguientes. Por este motivo podemos hablar de un circuito de memoria Burst 2-1-1-1, que en conjunto, sólo necesita 5 pulsos en lugar de los 8 usuales.

Junto a los datos, en la caché también se han de guardar las direcciones de memoria que corresponden a estos datos, Así cada línea de caché viene vinculada a lo que se denomina **etiqueta** o según la terminología anglosajona "*tag*". En ella, la caché guarda la dirección del dato en la memoria del sistema. Las líneas de caché y las etiquetas se almacenan en circuitos de memoria diferentes pero de la misma tecnología, SRAM.

Organización de las líneas de caché (Direct Mapping) para el 80486

He elegido al 80486 por su relativa sencillez frente a los complejos procesadores modernos. De este circuito estudiaremos la caché secundaria, pero no de un golpe, empezaremos con un concepto base que es el Direct Mapping y finalmente mencionaré la necesidad del concepto de caché asociativo.

El 80486 trabaja con una caché de segundo nivel de 256 KB. El ancho de una línea de caché es de 128 bits o sea 16 Bytes, por lo que tenemos espacio para 16384 líneas de caché (256 KB / 16 Bytes). Entonces, el cometido del controlador de caché es relacionar unívocamente la dirección de la CPU con una de estas 16384 líneas de caché.

16384 es 2 elevado a la 14 y por eso, los 14 bits inferiores de la dirección de la CPU determinan el número de la línea de caché. De todos modos, no se trata de los bits 0 a 13 sino de los bits 4 a 17, pues los bits 0 a 3 se necesitan para representar el offset de cada Byte en la línea de caché correspondiente.

La figura 4-5 muestra el diagrama de lo que es la caché en el modo Direct Mapping. Note que los bits 4 a 17 se usan como índice de las líneas de caché. Notará también que el 80486 tiene un bus de direcciones de 32 bits, así que nos sobran los bits 18 a 31. Estos últimos deben ser guardados como etiquetas de las líneas de caché.

Ahora bien, hagamos cuentas, el 80486 puede direccionar 2 elevado a la 32 direcciones, o sea, 4.3 GB, sin embargo, en cualquier placa base que soporte al 80486 solo se pueden colocar hasta 64 MB, esto es 2 elevado a la 26. La conclusión de esto es simple, de los 32 bits para generar una dirección, solo empleamos los primeros 26 bits. Así que solo guardamos los bits 18 a 25 (8 bits) en la tabla de etiquetas de la figura 20.

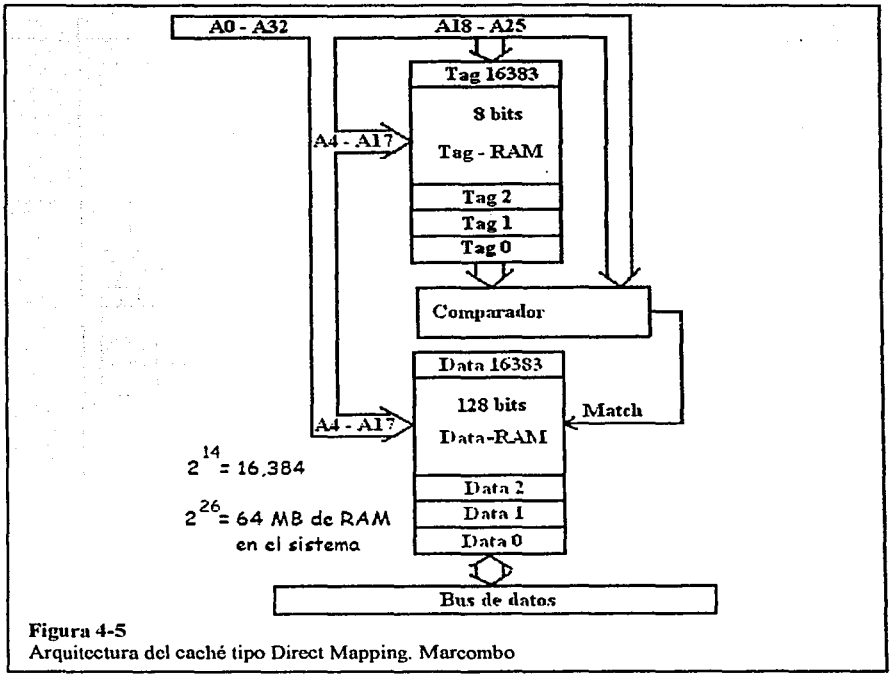


Figura 4-5
Arquitectura del caché tipo Direct Mapping. Marcombo

Localización de un dato en la caché

Cuando el CPU proporciona una dirección para un acceso de lectura, los bits 4 a 17 de la dirección, se emplean como índice a la tabla de etiquetas, Si los bits 18 a 25 coinciden con el valor de la tabla, entonces tenemos un acierto o caché hit y el dato puede ser recuperado de la caché.

Defectos de la estrategia Direct Mapping

Voy a plantear una situación hipotética: el CPU requiere de un dato y envía la dirección del mismo. Se toman los bits 4 a 17 que son el índice a la tabla de etiquetas y verificamos los bits 18 a 25 con la etiqueta almacenada. El resultado de la comparación es que ambas etiquetas, la proporcionada por el CPU y la almacenada en la caché son diferentes.

Por lo descrito en el párrafo anterior la línea de caché actualizará su contenido tomando los Bytes necesarios de la memoria que tiene conectada a continuación. Bueno, esta situación es correcta exceptuando en las siguientes dos situaciones

- La caché no está completamente llena, por lo que se desperdicia espacio.
- Dos aplicaciones están haciendo uso de la misma línea de caché.

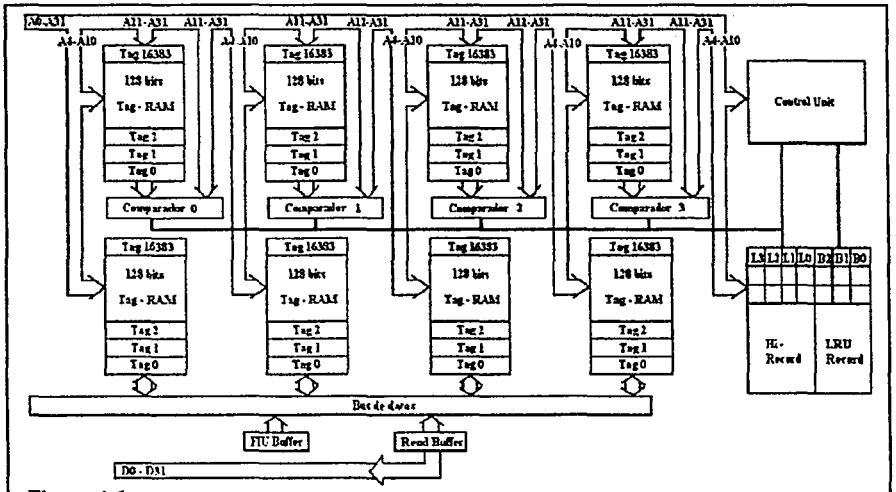


Figura 4-6
Arquitectura de una caché de primer nivel cuatro veces asociativa de 8 KB para el i486. Marcombo

La situación descrita en el último punto se empeora cuando los dos programas de aplicación requieren constantemente la línea de caché; así que resulta que la memoria intermedia estorba en vez de ayudar.

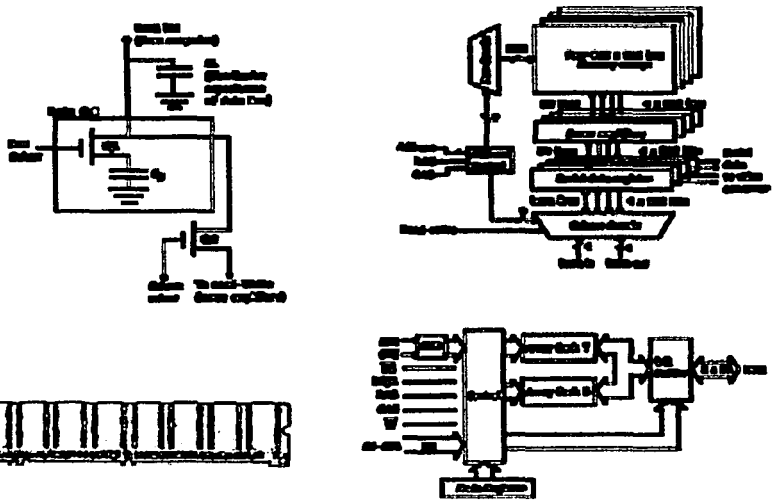
Para evitar estos malestares la caché se divide en bloques de tal forma que *a cada etiqueta almacenada le correspondan dos, cuatro e incluso ocho líneas de caché*. En este contexto, se habla de memorias caché dobles, cuádruples y ocho veces asociativas. Las figuras 4-6 nos ilustran el caso de una caché de primer nivel cuatro veces asociativa.

Así que en un caché asociativo, el controlador del mismo, en la búsqueda de una posición de memoria concreta, realizará comparaciones con dos, cuatro u ocho etiquetas, según el grado de asociatividad.

Concluyendo sobre la caché, aún falta por mencionar mucho acerca de las posibles estrategias de caché y en particular, lo que ocurre cuando la misma está llena y debe escribirse un dato más.

4-5 Concluyendo

Para hablar de microprocesadores requerimos de otra Tesis ya que sus tecnologías van mas allá de lo escrito en este capítulo. Como ejemplo, el IA-64 de Intel ya no tiene predictor de saltos, no obstante aún recurre a la ejecución especulativa cargando en sus tuberías las instrucciones que corresponden a ambas soluciones de una condición. Cuando el resultado de la condición está listo, solo se desechan las instrucciones de la condición falsa. La aparición del concepto multimedia obliga a los diseñadores a incluir circuitería adicional para el soporte de instrucciones multimedia. Algunas de esas instrucciones son del tipo SIMD, es decir, una sola instrucción obliga a varias unidades de ejecución a procesar un bloque de información.



Capítulo 5 Memorias DRAM

Existen diversas tareas que deben ser atendidas para lograr la síntesis de una imagen en computadora:

- Las operaciones en punto flotante propias las transformaciones geométricas.
- Las operaciones con datos de tipo entero propias del proceso de rastrerización.
- Los accesos a las memorias del sistema y del búfer de estructura.

En este segmento nos abocaremos a esa parte del hardware que conforma la memoria del sistema y a la memoria de vídeo; esta última la conocemos mejor como el búfer de estructura. Tal misión no es simple, los circuitos de memoria RAM han tenido un mayor desarrollo debido, principalmente al avance en la tecnología del microprocesador.

Vamos a considerar dos causas del desarrollo de la memoria RAM, la primera implica la cantidad de píxeles y la velocidad con la que se actualiza la imagen de un monitor para dar la sensación de una escena animada. La segunda causa implica al microprocesador de una computadora. Su amplio desarrollo también incluye un aumento en cuanto a la cantidad de información que puede suministrársele.

Ahora bien, a modo de ejemplo, consideremos la primera causa. Recuerde que el subsistema gráfico de su computadora debe tomar la información del búfer estructura 60 veces por segundo para formar una imagen en el CRT. Hagamos cuentas, considere un CRT de 1280 por 1024 píxeles, lo que corresponde una estación de trabajo con 32 bits por píxel y una actualización por pantalla de 60 Hz. Tal sistema requiere que cada acceso de 32 bits a su búfer de estructuras tarde 1/(1280x1024x60)=12.7 nanosegundos. Este número no toma en cuenta que el mapa de bits representativo de la imagen debe almacenarse primero en el búfer de estructura antes de ser desplegado. Tampoco toma en

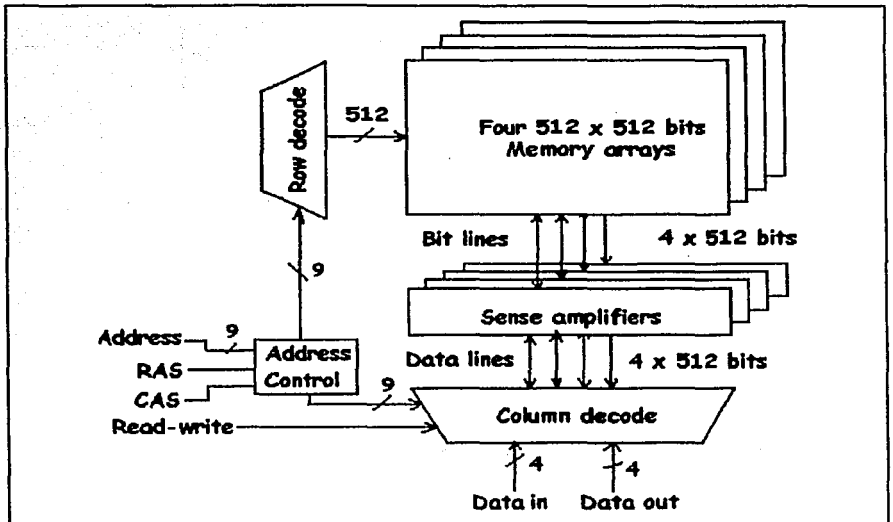


Figura 5-1
Circuito DRAM de 1 Mbit. Note que el circuito de memoria está dividido en 4 bancos de celdas. Los decodificadores de renglón y columna permiten acceso a 4 celdas al mismo tiempo, una por cada arreglo. Foley, Van Dam, Feinderm Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, 1995, pg. 857.

cuenta, la información adicional que se almacena en el mismo búfer: texturas y colores y cualquier información adicional que debe transferirse a la memoria de video y que se requiere para formar una imagen 3D.

La solución a todos estos problemas lo constituye un circuito que conocemos con el nombre de DRAM. Una DRAM es un circuito de memoria al que a cada momento hay que decirle "recuerda tu contenido". Diversas variantes de ese circuito constituyen una solución pasajera al problema de trabajar con más información.

5-1 La DRAM fundamental

El tipo de memoria preferida por los fabricantes de computadoras es la DRAM[1], cualquiera que sea su constitución. La ventaja de este tipo de memoria es la densidad de celdas que se pueden lograr en un solo encapsulado. La desventaja reside en la complejidad de los circuitos de lectura-escritura y en la necesidad de circuitos de apoyo, es decir, circuitos de refresco de la memoria; recuerde que las celdas de una DRAM pierden su carga en muy poco tiempo, por lo cual deben ser recargadas.

La figura 5-1 muestra el diagrama a bloques de un circuito DRAM de 1Mbit. Por supuesto, las memorias actuales son más complejas de lo que muestra este prototipo, sin embargo, el mismo nos sirve para entender el funcionamiento básico de un circuito de memoria. La información se almacena bit a bit en cuatro matrices cuadradas de 512x512 bits. Según la figura 5-1, las líneas verticales transfieran datos hacia y desde las matrices,

[1] DRAM significa *Dynamic Random Access Memory*

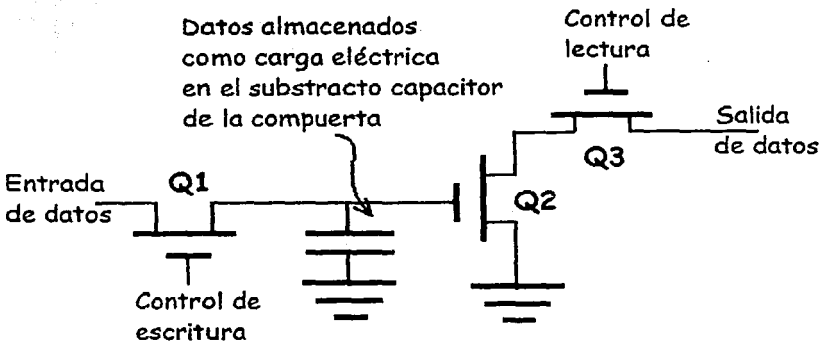


Figura 5-2
Calda de memoria MOSFET dinámica. Louis Nashelsky, "Fundamentos de Tecnología Digital", 2ª edición, 1993, pgs. 404 a 414.

una línea por cada columna de cada matriz; así que la longitud de palabra de la memoria es de cuatro bits.

Para tener un acceso de lectura o escritura a una celda específica de cada matriz requerimos de tres pasos. El primer paso es seleccionar el renglón, para esto se habilita la entrada *RAS*[2] y entonces se coloca la dirección del renglón de las terminales marcadas *Address*. El segundo paso es habilitar la entrada *CAS*[3] y entonces se coloca la dirección de la columna en las terminales marcadas *Address*. El tercer paso es definir si deseamos una escritura o una lectura desde la matriz. Una vez decidido, establecemos el voltaje adecuado en la terminal *Read-Write*.

La figura 5-2 muestra el circuito para una celda de memoria MOSFET básica para una DRAM, la cual emplea esencialmente un simple transistor MOSFET y un capacitor. Cada dato, un bit, se almacena como carga eléctrica en un capacitor muy pequeño, el cual debe recargarse o refrescarse miles de veces por segundo, o no se retendría la carga almacenada. Como puede notar, un transistor y un capacitor permiten una mayor densidad de celdas al formar una DRAM.

Veamos un poco más de cerca al capacitor. Éste se construye con un valor de 50 picofarads, según Louis Nashelsky, 1993, pg 409. El capacitor permanecerá cargado por un por algunos cientos de milisegundos, descargándose a través de la resistencia compuerta-drenaje de Q2. Para mantener la carga almacenada, es necesario refrescar o rescribir los datos periódicamente (al menos cada 2 ms). Actualmente, los circuitos de memoria agrupan sus celdas en bancos o páginas. Considerando ahora que el procesador principal requiere datos del mismo banco, sólo el 2% de todos los accesos a esas celdas será negado debido al proceso de refrescado.

Ahora, vamos a realizar una escritura de un bit a la celda. Una señal de control en la compuerta de Q1 lo enciende. En ese momento el voltaje del dato, suponiendo que sea VDD, cargará al capacitor vía Q1. Si el voltaje del dato es 0 volts (o nivel bajo) el capacitor se descarga vía Q1.

---

[2] en Foley, "*Computer Graphocs: principles and practice*", RAS es abreviatura de *row address strobe*.
[3] en Foley, "*Computer Graphocs: principles and practice*", CAS es abreviatura de column *address strobe*.

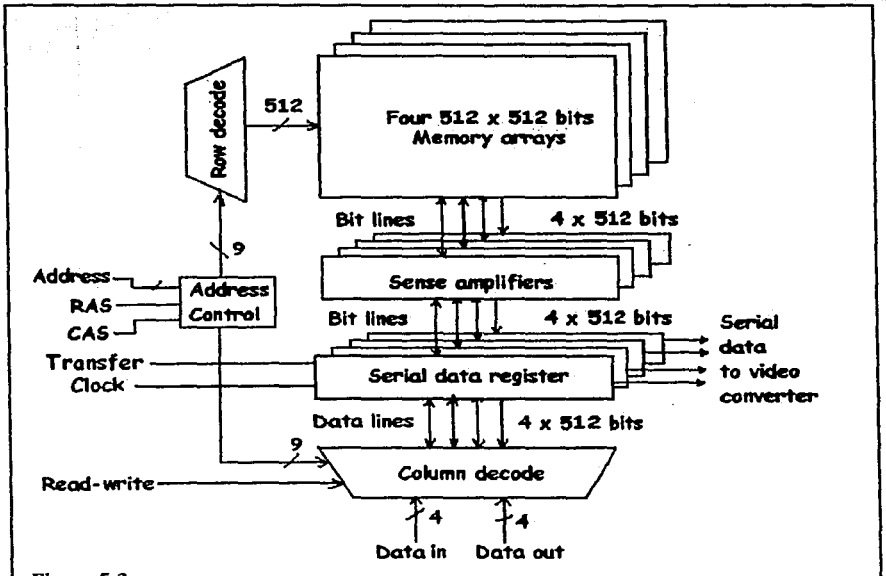


Figura 5-3
Diagrama de un CI VRAM de 1Mbit. Este circuito es muy parecido a una DRAM convencional. La diferencia estriba en 4 registros de carga paralela-salida serial capaces de cargar un renglón completo de cada matriz. Estos registros requieren de una señal de reloj de alta frecuencia. Foley, Van Dam, Feinderm Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, 1995, pg. 860.

El transistor Q3 se emplea para leer datos. Cuando la señal de control de lectura va a nivel alto, encendiendo Q3, el dato almacenado puede leerse en la salida de la celda. Ahora bien, a causa de que el capacitor se encuentra conectado en la compuerta de Q2 y de que Q2 proporciona la salida de datos en el drenaje, el bit del dato almacenado se invierte en la lectura.

Una carga eléctrica almacenada en el capacitor enciende a Q2 y así la salida es un nivel bajo. En tanto, un capacitor descargado mantiene a Q2 apagado y esto se interpreta por los circuitos de lectura como un nivel alto.

El refrescado de los datos se realiza cuando ambas señales de lectura y escritura se encienden, de tal suerte que los datos leídos se rescriben en la celda. Ahora bien, dado que la circuitería externa de refrescado mantiene un seguimiento del número par o impar de operaciones de refrescado realizadas, es posible determinar automáticamente si los datos son ciertos o están invertidos.

5-2 La Video RAM o VRAM

Considere ahora la situación siguiente: el controlador de vídeo necesita acceder al búfer de estructura 60 veces por segundo para generar las señales de control del monitor. A este proceso le llamamos **rastreo de salida de vídeo**. Ahora bien, la tubería gráfica genera

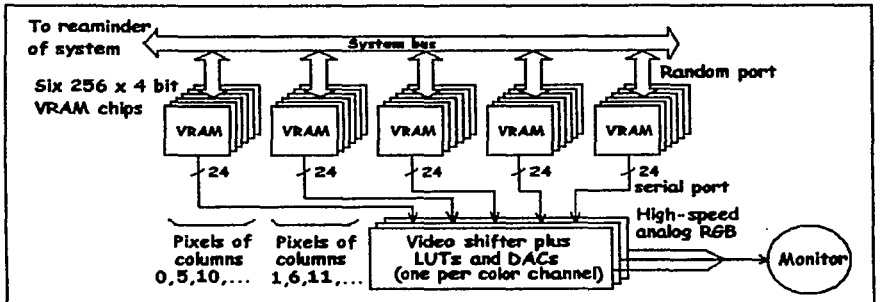


Figura 5-4
Diagrama a bloques de un búfer de estructura para una resolución de monitor de 1280x1024 píxeles. El circuito se construye con CI VRAM. Un registro de corrimiento proporciona píxeles a las altas velocidades que se requieren para formar la imagen en un monitor. Foley, Van Dam, Feinderm Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, 1995, pg. 861.

información continuamente para actualizar la imagen que va a desplegarse en el monitor, así que esta información se escribe en el búfer de estructura. Esta situación implica dos dispositivos turnándose el acceso a la memoria, lo que también implica que podríamos requerir un doble búfer de estructura: La imagen se construye en un búfer mientras el controlador gráfico accesa al otro para generar la imagen.

Una forma de simplificar esta situación es mediante una memoria que permita al controlador gráfico y a la tubería gráfica trabajar en forma paralela. La figura 5-3 es un prototipo de VRAM construido por *Texas Instruments* en 1983. Nuevamente recurrimos a los viejos modelos por su simplicidad en funcionamiento.

El circuito de la figura 5- 3 es similar a una DRAM convencional. La diferencia estriba en cuatro registros de **carga paralela-salida serial**, un registro por cada matriz. Las salidas de estos registros conforman un segundo puerto en el chip de memoria. Cada registro serie es tan ancho como la matriz de memoria, 512 celdas en el registro, así que un renglón completo de cada matriz puede ser copiado a su correspondiente registro polarizando el terminal *transfer*. Los registros tienen una terminal de reloj que les es común. Esta terminal recibe el nombre de reloj de alta frecuencia, lo que permite la transferencia de datos fuera del chip de memoria a alta velocidad. Las primeras versiones trabajaban con relojes de 30MHz.

Si este segundo puerto serie es empleado para las operaciones de rastreo de salida de vídeo, el proceso de rastreo puede ocurrir en forma paralela con las operaciones de lectura-escritura convencionales.

Ahora bien, considere esto, un solo chip VRAM puede soportar una resolución de 512 por 512 valores de píxel así entonces ¿qué hacemos si deseamos manejar una resolución de 1280 por 1024 píxeles? Obviamente se rebasa la capacidad de un solo chip, note también que al aumentar la resolución, es decir, la cantidad de píxeles a desplegar, se requiere también aumentar la velocidad del chip, lo cual, por lo general, no es posible. Una solución ya conocida es conectar múltiples bancos de chips VRAM a un registro de desplazamiento especial de alta velocidad, según se muestra en la figura 5-4. Múltiples valores de píxel, un grupo por cada banco de chips VRAM, pueden ser cargados en este registro de desplazamiento en forma paralela y entonces ser desplazados hacia fuera en forma serial a las velocidades adecuadas.

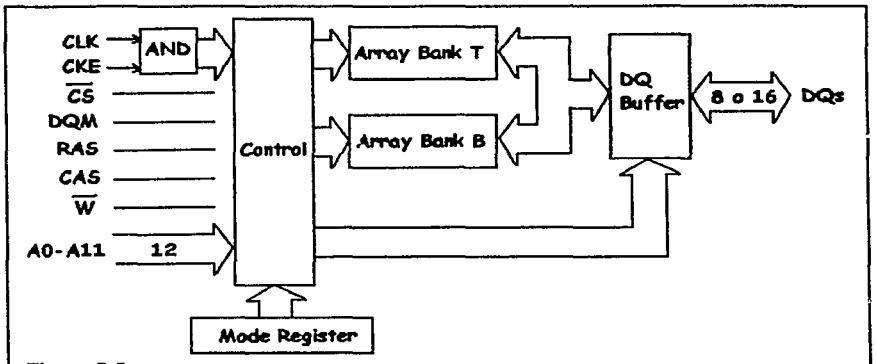


Figura 5-5
Diagrama a boques de un circuito SDRAM TMS626xx2 construido por Texas Instruments. El circuito puede almacenar hasta 16Mbit y puede operar en modo ráfaga, es decir, puede proporcionar un flujo continuo de bits. Texas Instruments, "*TMS626162, TMS626812 16M-bit Synchronous DRAMs Technical Reference*", 1996.

Respecto de la figura 5-4, note que una resolución de 1280 por 1024 son cinco veces la resolución de un sistema de 512 por 512, así que por esta razón empleamos cinco vías de información multiplexadas.

Normalmente, el registro de desplazamiento al cual alimentan los bancos de memoria se encapsula junto las con *tablas de búsqueda* (*look-up tables*) y convertidores digital a analógico y a todo integrado se le conoce como **RAMDAC**. Alto ¿tablas de búsqueda?, ¿Convertidores digitales a analógico? Por ahora no es una razón para preocuparse por estos términos, ya que he reservado un capítulo para hablar de este tipo de circuitos. Por lo mientras les diré que se trata del sistema necesario para crear una imagen en el monitor con la mayor calidad posible.

5-3 La SDRAM

Como consecuencia de la necesidad de equipos más potentes y confiables, los fabricantes diseñan, cada cual en su ramo, equipos que almacenan y procesan más rápido y mayor cantidad de información. Una situación ineludible a este respecto implica que los componentes de una computadora no pueden comunicarse por las diferentes velocidades a las que trasmiten o reciben información. Una alternativa implica sincronizar las operaciones de comunicación de todos los dispositivos con una sola señal de reloj. A esta señal le llamamos el **reloj del sistema**.

De manera semejante, los circuitos dentro del encapsulado de una DRAM funcionan todos con una sola frecuencia de reloj, que es la que proporciona el reloj del sistema. A consecuencia, a este tipo de DRAM se le llama **SDRAM** o bien **Synchronous DRAM**.

La figura 5-5 nos muestra el diagrama funcional de una **SDRAM TMS626xx2**, diseñada por Texas Instruments. La relativa sencillez de este circuito lo hacen ideal para explicar el funcionamiento básico de una SDRAM.

El dispositivo mostrado la figura 5-5 tiene una capacidad de 16Mbits y el acceso es aleatorio. Todas las entradas y salidas están sincronizadas con la transición positiva del reloj del sistema. Las celdas de bits se organizan en dos matrices o bancos. Para un acceso de lectura o de escritura hacia un banco, éste debe ser preactivado, luego entonces ya se puede suministrar la dirección del renglón y de ser el caso, la dirección de la columna. Los ciclos de refresco son suministrados alternativamente a cada banco.

El modo ráfaga

Por lo general, cuando vamos a leer de o escribir a una SDRAM, los datos se suministran en lo que llamamos **modo ráfaga** o bien, ***burst mode*** en cualquier bibliografía anglosajona. Vamos a tratar de explicar este concepto con un ejemplo sencillo: Deseamos escribir 256 bits en la memoria. Así que primero suministramos la dirección de inicio, es decir, especificamos la matriz y el renglón, pero no especificamos la columna (se sobreentiende que se trata de la primera columna) y a continuación suministramos los 256 bits. Lo mismo ocurre cuando vamos a leer de la memoria. También por lo general, los circuitos SDRAM incluyen una lógica por medio de la cual nosotros especificamos la cantidad de bits en ráfaga que deseamos leer o escribir. En el caso del TMS626xx2, podemos especificar 1, 2, 4, 8 y 256 bits en ráfaga.

La programación

El circuito TMS626xx2 no es sólo una memoria, se le ha incluido una lógica que permite programar o más bien fijar, su modo de operación. El modo de programación es mediante comandos y parámetros. Cada estado del circuito SDRAM es establecido por un comando diferente. Las terminales implicadas en la programación son las mismas que mostramos en la figura 5-5, excepto por la terminal DQM.

La operación con dos bancos

Antes de continuar, se hace necesario mencionar que el circuito TMS626xx2 es sumamente versátil ya que tienen una gran variedad de modos de acceso, tanto de lectura como de escritura. Para ejemplificar la aplicación de comandos que modifiquen la operación del CI SDRAM, revisaremos levemente una operación genérica de acceso entrelazado a los dos bancos.

Vamos a suponer que hemos almacenado información en los renglones de ambos bancos de manera alternativa. Ahora preactivamos uno de los bancos suministrando el comando correspondiente junto con el banco deseado estableciendo un voltaje para la terminal A11. A continuación, activamos un renglón estableciendo su dirección en las terminales A0-A10. Finalmente ejecutamos un comando READ (lectura) o bien, el comando WRT (escritura). Mientras la información se está transmitiendo hacia o desde la memoria, podemos preactivar el otro banco de celdas de la misma forma. El CI SDRAM comenzará sus operaciones con el segundo banco hasta que haya terminado con el primero. Como puede ver, podemos tener un flujo continuo de información.

Bueno; lo he descrito de forma muy simple, sin embargo, la realidad es muy distinta. Para cualquier información adicional puede referirse a "*TMS626162, TMS626812 16Mbit Synchronous DRAMs Tecnical Reference*" [4].

[4] Referido como el documento SMOU002.

Año	Tecnología	Frecuencia
1970	RAM/DRAM	4,77 mhZ
1987	FPM	20 MHz
1995	EDO	20 MHz
1997	PC66 SDRAM	66 MHz
1998	PC100 SDRAM	100 MHz
1999	RDRAM	800 MHz
1999-2000	PC133 SDRAM	133 MHz
2000	DDR SDRAM	266 MHz

Figura 5-6
Esta tabla muestra el año de lanzamiento comercial de varios tipos de CI RAM, el tipo de tecnología y la velocidad del bus de datos. Fuente: http://home.ctl.rr.com/eaa/MemoryTypes.htm

5-4 La evolución de las memorias

Ahora daremos un vistazo a la evolución de los circuitos de memoria que han acompañado a las computadoras por casi 30 años. La tabla de la figura 5-6 nos muestra la fecha del lanzamiento comercial, el tipo de memoria y frecuencia de reloj a la cual opera. ¿Quién no recuerda nuestra primera computadora casera de tipo PC? Un procesador IA8088 era el corazón del sistema. Esta computadora operaba a 4.77MHz y los circuitos de memoria RAM estaban soldados a la placa base. Por supuesto, a veces las memorias no tenían el dato a tiempo, así que el procesador debía esperar un par de ciclos hasta que la memoria tuviera listo el dato.

FPM (Fast Page Mode)[5]

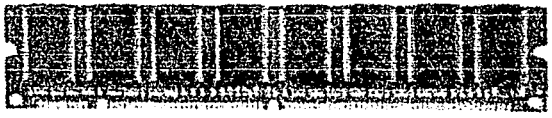
O bien, modo de paginado rápido. Un circuito de memorias FPM era como un libro. Para acceder a un dato, primero buscamos la página, luego buscamos el renglón y finalmente la columna. Por supuesto, si el siguiente dato a buscar ya se encontraba en la misma página y en el mismo renglón, sólo se debe especificar la columna. Una memoria FPM se construye con circuitos DRAM.

EDO (Extended Data Output)[4]

O salida de datos extendida. Este tipo de circuito de memoria se distingue de un FPM en que el dato contenido en las terminales del circuito, leído desde las celdas, permanecerá aún después de que se suministre otra dirección. El dato permanecerá en las terminales hasta que el circuito de memoria tenga listo otro dato. Tal vez nos preguntemos si esta situación existía ya en la FPM, la respuesta es no. Otra diferencia implica que las memorias EDO son un 10 a un 20 por ciento más rápidas que con memoria FPM. Una EDO se construye con circuitos DRAM. La figura 5-7.a muestra un circuito de memorias EDO.

[5]Fuente: http://home.ctl.rr.com/eaa/MemoryTypes.htm

(a)



(b)

Figura 5-7
Circuitos de memoria. En (a) tenemos un circuito de tecnología EDO. Los circuitos EDO se construyen con integrados DRAM. En (b) tenemos un circuito de tecnología SDRAM. Fuente: http://www.conozcasuhardware.com/articulo/ramcant1.htm#presentation.

DDR SDRAM[4]

DDR significa "*Double DataRrate*", y esto a su vez significa que la memoria funciona con las dos transiciones de un pulso de reloj en vez de con sólo una transición como en una SDRAM convencional. La consecuencia, es prácticamente una duplicidad en cuanto a la cantidad de información que puede transferirse al procesador principal.

SDRAM[6]

La figura 5-7.b nos muestra un circuito SDRAM. La memoria SDRAM, bien sea **PC66, PC100** o **PC133**, tiene un ancho de bus de datos igual 64 bits, lo que significa que en cada ciclo de reloj envía 64 bits, o sea, ocho bytes. De esta forma, su capacidad de transferencia de datos, es decir, su velocidad útil será:

- **PC66**: 8 bytes/ciclo x 66 MHz = 533 MB/s.
- **PC100**: 8 bytes/ciclo x 100 MHz = 0.8 GB/s.
- **PC133**: 8 bytes/ciclo x 133 MHz = 1.06 GB/s.

Cuánto más rápido se vuelven los microprocesadores más importante resulta tener un canal de comunicación fluido entre éstos y la memoria, algo que también es importante cuando se almacenan texturas en la memoria principal para que posteriormente sean transferidas al subsistema de vídeo.

[6] Fuente: http://www.conozcasuhardware.com/articulo/ramcant1.htm#presentation

RDRAM o Rambus DRAM[5]

La **RDRAM** o memoria **Rambus** se planteó como la solución de la necesidad de un mayor ancho de banda, veamos si es verdad. La Rambus tiene bus de datos más estrecho, de sólo 16 bits, o sea 2 bytes; no obstante que funciona a velocidades mucho mayores, esto es 300, 356 y 400MHz. Además, trabaja con las dos transiciones de un pulso de reloj, de forma que en cada ciclo de reloj envía 4 bytes en lugar de 2.

Debido este doble aprovechamiento de la señal, se dice que la Rambus funciona a 600, 712 y 800MHz equivalentes. Y por motivos comerciales, se la denomina PC600, PC700 y PC800. Por todo ello, su capacidad de transferencia es:

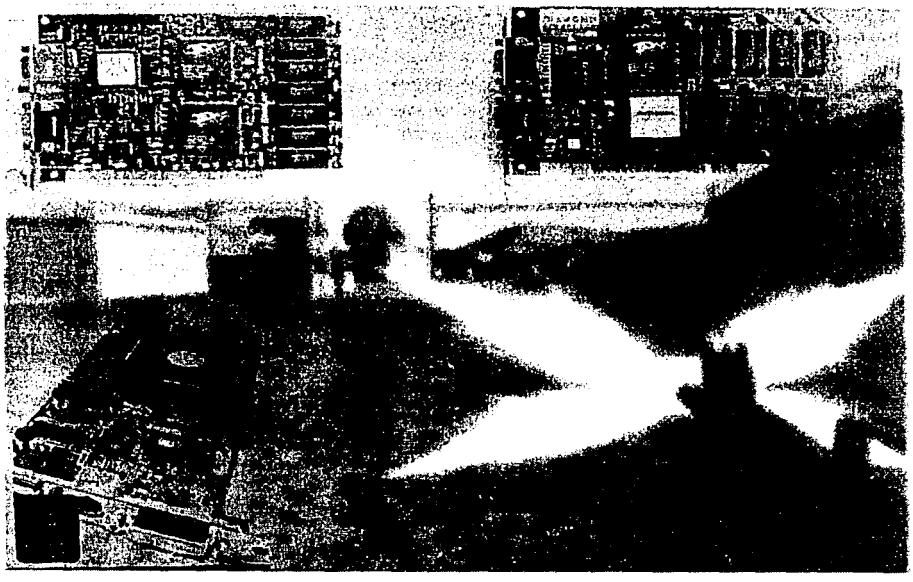
- **Rambus PC600**: 2 x 2 bytes/ciclo x 300 MHz = 1.2 GB/s
- **Rambus PC700**: 2 x 2 bytes/ciclo x 356 MHz = 1.42 GB/s
- **Rambus PC800**: 2 x 2 bytes/ciclo x 400 MHz = 1.6 GB/s

Como vemos, la Rambus más potente (la de 800MHz equivalentes) puede transmitir el doble de datos que la SDRAM PC100, lo que no es poco pero no es ocho veces más, como a muchos publicistas les gusta hacer creer.

5-5 Conclusiones

Nuestra necesidad de generar imágenes y animaciones más realistas ha originado el desarrollo de los componentes de las computadoras, entre ellos los circuitos de memoria RAM. Este capítulo es testimonio de esa evolución al contener información sobre circuitos que ya son obsoletos. Sin embargo circuitos como la VRAM han marcado la pauta para diseñar circuitos del tipo SDRAM con la característica adicional del modo ráfaga. En el momento en que este capítulo fue escrito los subsistemas de vídeo más socorridos para juegos ya incorporaban memorias síncronas.

Diseños como la DDR SDRAM son el último grito de la moda comercial, es decir, circuitos que antes eran privilegio de las estaciones de trabajo más sofisticadas, ahora ya son accesibles a muchos usuarios caseros. Bueno, también, al momento de escribir este capítulo, la memoria Rambus estaba siendo desplazada por la PC266.



Capítulo 6 Subsistema de video: las tarjetas gráficas

La tarjeta gráfica es la que se encarga de procesar la imagen que el procesador principal le envía a ésta y a su vez de enviarla al monitor. Hoy día las tarjetas realizan dos operaciones básicas:

- Interpreta los datos que le llegan del procesador, ordenándolos y calculando para poder representarlos en forma de un rectángulo compuesto de puntos (píxeles).
- De los datos que resultan del proceso anterior, los transforma en señales analógicas que pueda entender el monitor.

Estos dos procesos suelen ser realizados por un solo circuito, alto, ¿uno solo? La realidad implica que los fabricantes encapsulan todo junto, el procesador que se encarga de la tubería gráfica, el RAMDAC, el controlador gráfico, el RAMDAC[1], etc.

Hoy día se pueden encontrar en las tarjetas gráficas microprocesadores que llamaremos aceleradores gráficos. Éstos, se encargan de despejar casi todo el procesamiento de los gráficos del CPU haciendo que el computador, en general, opere más rápidamente. Las tarjetas gráficas que incluyen éstos aceleradores heredan el nombre de tarjetas aceleradoras y normalmente son más costosas que las tarjetas gráficas normales pero valen la pena. El mercado está lleno de circuitos que ofrecen un compromiso de rendimiento entre una imagen 2D, como la que se aprecia en Windows de Microsoft y una

[1] El RAMDAC es, o mejor dicho, son dos circuitos manipulados por el controlador gráfico. El primer circuito es una SRAM que identificamos mejor como la tabla de búsqueda. EL segundo circuito está formado por tres convertidores digital a analógico, los cuales convierten los datos binarios recuperados de la tabla de búsqueda en niveles de voltaje analógicos que controlan las intensidades de los tres haces rojo, verde y azul.

imagen como la de *Doom-Quake-Duke Nukem 3D*, misma que puede apreciar en la portada de este capítulo.

Las tarjetas gráficas y los aceleradores traen normalmente memoria RAM para guardar la información de las imágenes que se muestran en el monitor: búferes de imagen, coordenadas de vértices, texturas, comandos, etc. Una mayor cantidad de esta memoria montada sobre la tarjeta permite mayores resoluciones y mayores cantidades de colores en la pantalla. Hay tarjetas gráficas que tienen memorias más veloces, otras en cambio suplen las deficiencias en velocidad con un bus de datos más ancho, por ejemplo, los buses de datos que comunican al procesador gráfico con la memoria son de 32, 64 y 128 bits. El estándar de hoy día, en cuanto a cantidad de memoria interconstruida es de 8 MB de memoria para una tarjeta gráfica común y corriente, 32 MB para los aceleradores de 3D y hasta 64 MB para los aceleradores más especializados.

Veamos ahora cuáles son algunos procesadores gráficos que dominan el mercado hoy día:

- NV20 de nVidia
- 3dfx
- ATI Radeon
- SiS
- Trident
- Cirrus Logic

Lástima por los circuitos siguientes, sobre todo por Trimedia, que fue un procesador revolucionario en cuanto a generación de imágenes 3D y procesamiento 2D. Las siguientes empresas se han quedado rezagadas en el mercado:

- Permedia II de Permedia Labs
- Trimedia[2] de Phillips

6-1 Las tarjetas gráficas y la tubería de despliegue en

Los párrafos que vimos en la introducción de este capítulo están bien para una revista orientada al público en general. Nosotros, en cambio, necesitamos un poco más de detalle, por ejemplo ¿qué operaciones realiza la tarjeta aceleradora y cuáles quedan para el CPU? Para responder a ésta pregunta recurrimos a nuestro viejo concepto de tubería gráfica (también llamada tubería de despliegue). La figura 6-1 es una representación genérica que se ha obtenido del primer capítulo. Ahora bien, si recordamos lo que es una tubería de despliegue, ésta tiene básicamente tres secciones:

- Modelo de despliegue, que es la forma que se suministra información a la tubería.
- El subsistema geométrico.
- El subsistema posterior.

Veamos la primera etapa, o sea, el modelo de despliegue. Es el procesador principal el que se encarga de generar y suministrar los comandos de dibujo a subsistema gráfico, el cual se encarga del resto de la tubería.

Veamos más de cerca, el procesador principal, además de realizar la simulación, debe crear la descripción de un objeto 3D: posiciones de los vértices de sus superficie, color,

[2]Trimedia es una marca registrada de Phillips Inc.

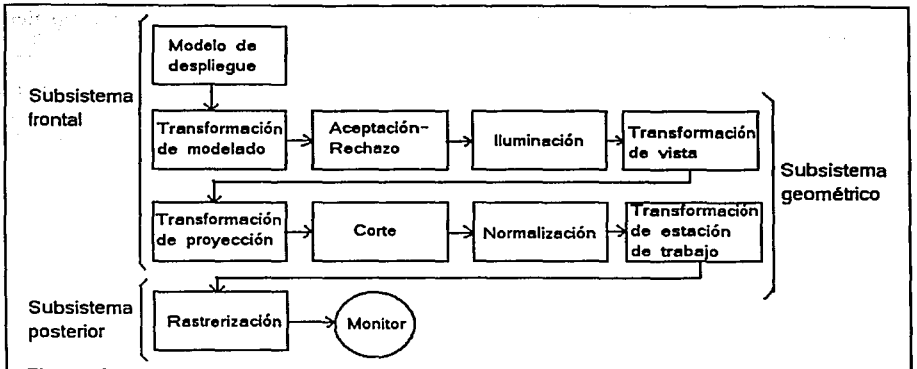


Figura 6-1
Diagrama a bloques de una tubería de representación gráfica. En el esquema se pueden observar los subsistemas geométrico y posterior. El subsistema geométrico se encarga básicamente de transformaciones geométricas, entre las cuales, las más básicas son la traslación, la rotación y la escala. Parte del trabajo del subsistema frontal es fragmentar los cuerpos ubicados en la escena en cientos y a veces miles de triángulos. El subsistema posteriores encarga de mostrar esos triángulos en pantalla empleando algoritmos de conversión por rastreo.

textura, ángulos de rotación de la figura, factor de escala, etc. Hecha la descripción del objeto, se pasa esta información a la tubería gráfica la cual representa al objeto en la pantalla.

Cabe mencionar que si el objeto 3D debe ser deformado o bien, debe explotar, el procesador principal creará una descripción diferente del objeto por cada paso de la animación.

Continuando con la explicación, el subsistema posterior tiene a su cargo la presentación de la imagen aplicando todo tipo de técnicas de sombreado. Tradicionalmente, este subsistema requiere de una aritmética entera, no obstante, las técnicas de sombreado actuales, la reflexión, la transparencia y la aplicación de sombras también requieren de una aritmética en punto flotante. Este subsistema, con todas o algunas de las técnicas de sombreado ya mencionadas, al momento de escribir el presente capítulo, se implementan ya en la tarjeta gráfica.

Algunas técnicas de sombreado o matizado que se pueden encontrar en las aplicaciones gráficas como simuladores y juegos de vídeo, son:

- Sombreado Gouraud.
- Texturas.
- Sombras.
- Reflexiones.
- Transparencias.

Ahora bien, debido a que continuamente surgen nuevos sistemas con más capacidades interconstruidas, los sistemas desplazados tendrían que ser remplazados para seguir la moda. Uno de los mejores inventos de este principio de siglo es la **Arquitectura Unificada de Dispositivos** (y en inglés UDA: Unified Driver Architecture). Esto significa que, para un mismo fabricante, no importa que procesador esté montado en el subsistema

de video: si es un procesador antiguo aún será útil ya que las funciones que no tiene interconstruidas, las implementa el procesador principal.

Voy a hacerle una pregunta ¿qué técnicas se emplean para presentar las superficies en una tarjeta gráfica moderna? Aunque no lo crea, aún se recurre a la aplicación de líneas de rastreo, ya sea por objeto o por imagen: esto es lo que se llama **conversión por rastreo.** En las siguientes secciones recordaremos un poco la técnica de conversión por rastreo y en la sección siguiente veremos como se implementa en un subsistema de vídeo.

6-2 La API y las tarjetas de vídeo

Los aceleradores 3D, como se les conoce a las tarjetas de vídeo actuales, son un tipo de tecnología muy especializada. Veamos a los favoritos: Voodoo 5[3] y GeForce[4]. Éstos sólo pueden dibujar en escena 3D si la misma es descrita en un lenguaje que sólo cada uno de estos sistemas entiende. No obstante, los programadores de aplicaciones realizan sus creaciones empleando una interfaz de programación que conocemos mejor como API. Las API más populares son Direct3D[5] y OpenGL[6]. La ventaja de las API implica una forma de comunicación entre el acelerador gráfico y el programa de aplicación. Cada empresa fabricante proporciona su versión de la API, pero no importa ya que los comandos que contienen, todos tienen la misma sintaxis.

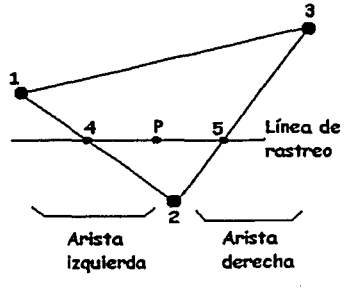


Figura 6-2
Algoritmo de conversión por línea de rastreo, esta técnica es muy empleada en sombreado Gouraud, aunque también se emplea en sombreado Pong, transparencias, mapas de sombras, etc. El algoritmo interpola de manes lineal todos los parámetros de los píxeles entre los puntos 4 y 5 ubicados sobre dos aristas del triángulo.

El caso de OpenGL es curioso. A pesar de haber sido creado con el objeto de formar un elemento común a todo tipo de estaciones de trabajo y sistemas caseros, se le ha relegado al mundo del PC. El lugar que OpenGL ocupaba en estaciones de trabajo hoy día lo cubre otra API llamada MesaGL[7] que es una versión libre del pago de derechos de autor y además es compatible con OpenGL.

Direct3D es un grupo de librerías desarrolladas originalmente por los ingenieros de Silicon Graphics y que fueran licenciadas por Microsoft para poder integrarlas en sus sistemas operativos. Direct3D es parte de otro, aún más grande conjunto de librerías llamado DirectX. La última versión de estas librerías se llama DirectX 8.1.

6-3 Tubería de rastrerización

La figura 6-2 muestra el algoritmo de rastrerización más común (también se llama algoritmo de conversión por línea de rastreo) y que se emplea con polígonos convexos. Ahora bien, antes de continuar, debemos repasar la información que define a un vértice. Un vértice se caracteriza por la posición (x, y), la profundidad, los valores R, G y B que corresponden con el color y el parámetro alpha que es un índice de transparencia.

Básicamente son tres las etapas del proceso de rastrerización que se emplean en los algoritmos:

- Procesamiento de polígonos.

---

[3] Voodoo 5 es un producto de 3dfx.
[4] GeForce es una marca registrada de nVidia Inc.
[5] Direct 3D es una marca registrad de Microsoft Corp.
[6] OpenGL es un productor controlado por el "*OpenGL Architectural Review Board*".
[7] MesaGL es un producto destinado al dominio público.

- Procesamiento de aristas.
- Procesamiento de líneas de rastreo.

El procesamiento de polígonos. Al entrar un polígono a esta etapa, de la información que lo define se generan pares de vértices que definen cada arista. A su vez, las aristas se clasifican en izquierdas y derechas: recuerde que las líneas de rastreo se trazan precisamente de izquierda a derecha. Para resumir, de un polígono que entra en esta etapa, la salida son definiciones de aristas izquierdas y derechas.

El procesamiento de aristas. Del grupo de aristas izquierdas y derechas, se obtienen parejas de vértices que definen, cada una, una línea de rastreo.

El procesamiento de líneas de rastreo. De cada línea de rastreo que entra a esta etapa, se generan, por interpolación lineal, los valores de píxel que son internos al polígono. Por si no lo recuerda, los valores de píxel son: *(x,y,z,R,G,B,alpha)*.

Por último, existen dos formas de aumentar el rendimiento de un sistema que rastreriza polígonos. Uno es aumentar la frecuencia del reloj del circuito y otro es agregar paralelismo en cada etapa. En la siguiente sección veremos un ejemplo aplicado a un sistema Silicon Graphics denominado *Power Iris*.

6-4 Power Iris de Silicon Graphics

Power Iris[8] es un sistema ya antiguo, no obstante, las técnicas que emplea son implementadas por los aceleradores gráficos de hoy. Primero, veamos algunas generalidades de este sistema. El Power Iris cuenta con un CPU compuesto de cuatro multiprocesadores que comparten un solo bus de memoria. Su subsistema gráfico puede representar 100,000 cuadrilaterales por segundo a todo color empleando sombreado Gouraud y búfer z. Un momento ¿cuadrilaterales? Así es, Silicon Graphics trabaja con polígonos de cuatro lados en vez de triángulos. La arquitectura del subsistema gráfico se muestra en la figura 6-3. En resumen, la figura nos muestra cinco subsistemas:

- **Subsistema CPU.**
- **Subsistema geométrico.**
- **Subsistema de conversión por rastreo.**
- **Subsistema de rastreo.**
- **Subsistema de despliegue.**

El subsistema del CPU

Este subsistema no nos interesa en este capítulo, tan sólo menciono que se encarga de determinar la posición de los objetos en la escena en cada paso de la animación y suministrar una lista de despliegue.

El subsistema geométrico

Este subsistema transforma, corta e ilumina las primitivas. En resumen, el subsistema está compuesto de cinco procesadores de punto flotante que trabajan en tubería. Observe la figura 6-3. Cada uno de esos procesadores es llamado **Máquina geométrica** o

[8] Power Iris es una marca registrada de Silicon Graphics Inc.

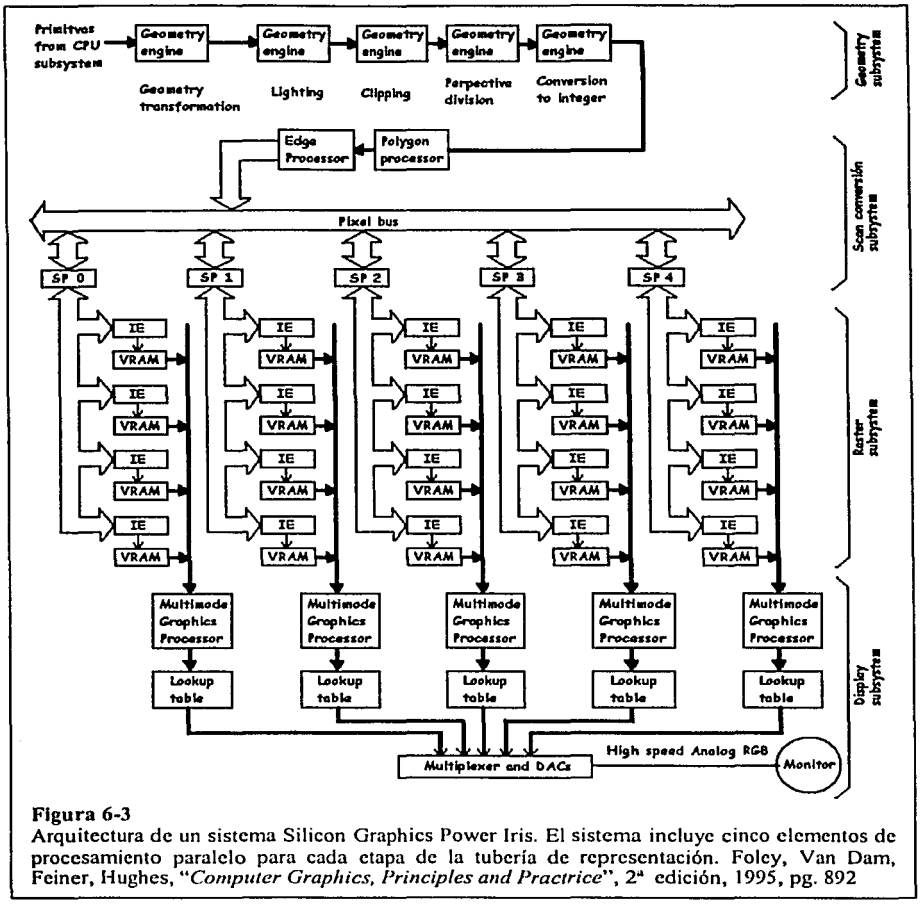


Figura 6-3
Arquitectura de un sistema Silicon Graphics Power Iris. El sistema incluye cinco elementos de procesamiento paralelo para cada etapa de la tubería de representación. Foley, Van Dam, Feiner, Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, 1995, pg. 892

simplemente GE[9]. Cada GE contiene una pila FIFO de entrada, un controlador y una unidad de punto flotante con capacidad de 20 megaflops. Cada GE es circuito comercial Weitek 3332.

El primer GE se encarga de una operación equivalente a la transformación de modelado (rotación, escala y traslación). Entre los elementos que sufren las transformaciones geométricas en este primer GE están los vértices y vectores normales de cada vértice. El segundo GE se encarga de las operaciones de iluminación. Este segundo circuito soporta hasta ocho fuentes de luz ¿No es una familiar el concepto de ocho fuentes de luz? El tercer GE ejecuta pruebas de aceptación-rechazo y corte de cada primitiva. El objeto de este GE es eliminar primitivas, polígonos, que no aparecen en escena y que por tanto no requieren ser representados. El cuarto GE se encarga de la transformación de vista y la proyección de las primitivas en el plano de proyección. El quinto GE se encarga de la transformación de estación de trabajo y además hace la conversión de datos de tipo punto flotante a datos de tipo entero.

[9] GE es una abreviación de *Geometry Engine* según la bibliografía anglosajona.

Como puede notar, el subsistema geométrico de Power Iris se parece un poco al que he mostrado en el capítulo primero. Los siguientes subsistemas ilustran el concepto de procesamiento paralelo de las primitivas.

Subsistema de conversión por rastreo

Este subsistema realiza la conversión de las definiciones de polígonos a definiciones de líneas de rastreo. Hay dos diferencias fundamentales entre Power Iris y un sistema moderno. La primera implica que Power Iris se trabaja con cuadrilaterales y no con triángulos. La segunda es aún más distintiva: las líneas de rastreo[10] son verticales y no horizontales como se acostumbra en una PC.

Observe la figura 6-3, un solo procesador de polígonos ordena los vértices de cada polígono de izquierda a derecha. Los vértices ya ordenados son empleados para descomponer el polígono en trapezoides alineados verticalmente (recuerde que Silicon Graphics trabaja con cuadrilaterales en vez de triángulos). La pareja de vértices superiores y la pareja de vértices inferiores de cada trapezoide son usadas para calcular las pendientes de las aristas que forman al cuadrilateral.

El procesador de aristas emplea la información que dan los vértices y las pendientes de las aristas para calcular las coordenadas (x, y) y valores de color de cada píxel que resulta de intersección de una línea de rastreo con las aristas de algún trapezoide. En resumen, de este proceso se obtienen dos puntos que definen un segmento de línea de rastreo dentro del trapezoide.

Observe la figura 6-3, lo que sigue del procesador de aristas son cinco **procesadores de línea de rastreo** o SP[11] trabajando paralelamente. Cada uno de los SP es responsable de una de cada cinco columnas en la pantalla. Por ejemplo el *SP0* maneja las líneas de rastreo *0*, *5*, *10* y así. Cada SP calcula las dimensiones z, R, G, B y alfa (recordemos que alpha se usa para transparencias y antialiasing[12]) para cada píxel en la línea de rastreo.

Subsistema de rastreo

Los SP se encargan de calcular la posición de cada píxel dentro de la línea de rastreo, sin embargo, el color de cada píxel, para cualquier técnica de sombreado o texturizado que se aplique es encargada a los IE o *Image Engine* según la bibliografía anglosajona.

Todo el subsistema de rastreo está compuesto de 20 máquinas de imagen o IE, cada una con sus respectivos circuitos de memoria VRAM que contienen, en conjunto, un búfer de estructura, un búfer de profundidad, un búfer alfa y otros.

Las 20 IE trabajan en paralelo sobre los píxeles verificando la profundidad con el búfer z. y verificando también la transparencia de los mismos con el búfer alfa.

---

[10] En la bibliografía anglosajona, al segmento de línea de rastreo entre dos aristas de polígono se le conoce como "*span*". No tiene equivalente en el español.

[11] SP es abreviación de *Span Processor* según la bibliografía anglosajona.

[12] El Antialiasing emplea una técnica de filtrado paso-bajas cuyos resultados en el dominio del espacio muestran imágenes sin bordes afilados.

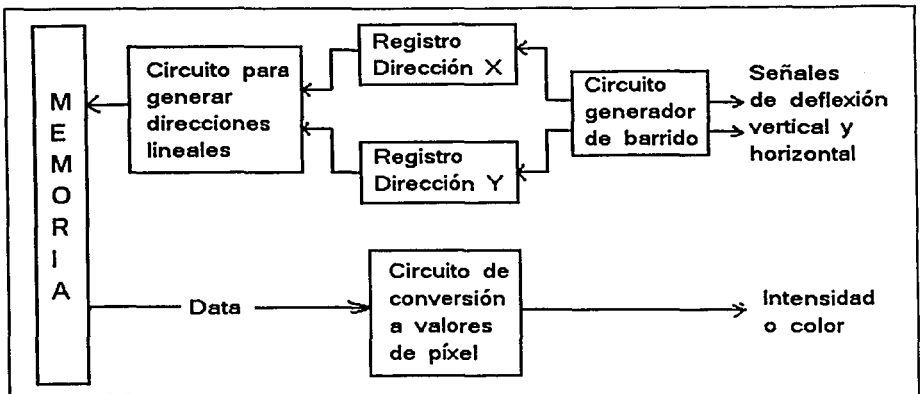


Figura 6-4
Organización lógica de un controlador de video. Foley, Van Dam, Feiner, Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, 1995, pg. 167.

Subsistema de despliegue

Este subsistema contiene cinco **procesadores de despliegue multimodo o MGP**[13]. Todos los MGP trabajan en paralelo leyendo la información del búfer de estructura. Los MGP también requieren de la información de las **tablas de búsqueda** para desplegar una imagen en modo RGB o pseudocolor. Los resultados de cada pareja *MGP-tabla de búsqueda* se multiplexan en un registro de corrimiento de alta velocidad el cual a su vez alimenta los tres DAC que impulsarán al monitor de vídeo.

Concluyendo

En este sistema podemos apreciar una de las aplicaciones de procesadores SIMP, es decir, varios procesadores que comparten el mismo código de programa y que a una orden, ejecutan la misma tarea sobre distintos datos. Si no se ha dado cuenta de a que circuitos me refiero, observe los procesadores SP, IE y MGP. Todos sincronizados para realizar la misma tarea.

Por supuesto, podemos tener cierto desperdicio de capacidades en este sistema ya que puede ocurrir que una de las cinco líneas de rastreo no tenga que hacer. Así que algunos procesadores estarán ociosos.

6-5 El controlador de video

La figura 6-4 nos muestra un esquema simplificado de un controlador de video. Típicamente el controlador lee el búfer de estructura unas 60 veces por segundo. Semejante frecuencia es para evitar el parpadeo en la imagen. El circuito generador de barrido genera las señales de deflexión que controlan la orientación de los haces de

[13] MGP es abreviación de *Multimode Graphics Processor.*

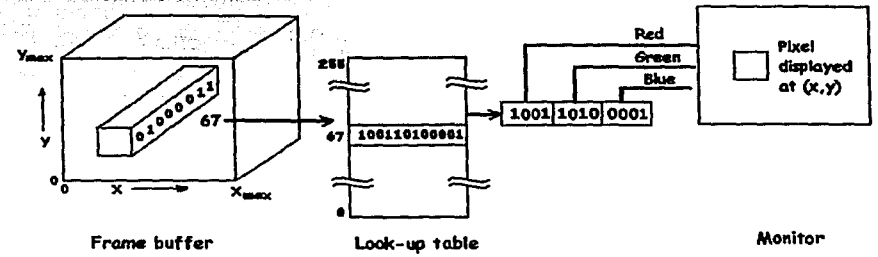


Figura 6-5
Organización de una tabla de búsqueda de video. Un píxel con valor 67 (binario 01000011) es desplegado en la pantalla con el cañón de electrones rojo a 9/15 del máximo, verde a 10/15 y azul a 1/15. Esta tabla de búsqueda es mostrada con 12 bits por entrada. Foley, Van Dam, Feiner, Hughes, "*Computer Graphics, Principles and Practrice*", 2ª edición, 1995, pg. 167.

electrones dentro del monitor. Este circuito también genera las coordenadas (x, y) que definen la posición de un píxel en el búfer de estructura. Un circuito que identificamos como *Linear-address* en la figura 6-4 se encarga de convertir las coordenadas en direcciones lineales.

El generador de barrido genera las coordenadas *(x,y)* de la siguiente forma. Los registros *X* y *Y*, mismos que se identifican en la figura 6-4, tienen inicialmente los valores $X=0$ y $Y=Y_{max}$. El generador modifica el registro *X* en incrementos de *1* desde *0* hasta un valor X_{max}. Para cada valor de *X* que se genera, se busca un valor de píxel en memoria y esto es lo que se llama **barrer una línea de rastreo** o bien, **barrido horizontal**. Entonces el registro *X* se inicializa a *0* y el registro *Y* se decrementa en *1* comenzando así el barrido para la segunda línea de rastreo. Una vez que se han barrido todas las líneas, los registros *X* y *Y* vuelven a sus valores iniciales $X=0$ y $Y=Y_{max}$. En general, al barrido de todas las líneas horizontales se le llama **barrido vertical**.

La tabla de búsqueda

Cada valor de píxel recuperado de la memoria pasa a la tabla de búsqueda, la figura 6-5 ilustra al respecto. Veamos, cada valor de píxel corresponde con una entrada en la tabla, la cual contiene valores que se emplearán para controlar la intensidad de los tres haces de electrones que dibujan la imagen tiene en el CRT. Para que comprenda mejor la razón de ser de este circuito, debemos conocer algunas de las necesidades que resuelve:

- Puede ocurrir que se requiere adaptar el brillo y el contraste a las condiciones de iluminación. Otras veces deseamos agregar algo de tinte a una imagen o bien, deseamos desplegar la misma con una escala de grises.
- Otra necesidad es la de crear un despliegue en pseudocolor a partir de una imagen en blanco y negro.
- A veces, para ahorrar memoria, configuramos nuestro equipo para que despliegue imágenes con menor cantidad de colores. Así que en la memoria se almacenarán índices a la tabla de búsqueda, en donde se localizan los valores de píxel equivalentes RGB

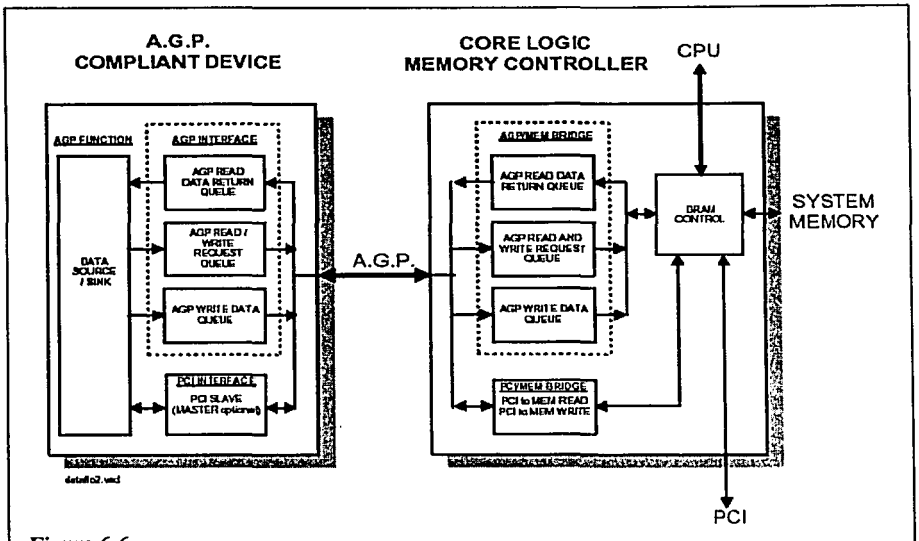


Figura 6-6
Arquitectura de la especificación 2.0 de la interfaz AGP. Intel Corp., "*Accelerated Graphics Port Interface Specification*", Revisión 2.0, Mayo de 1998, pg. 25.

Actualmente, en cualquier computadora, ya sea casera o estación de trabajo, es posible acceder a un panel que controla las características de la imagen que vemos: brillo, contraste, tinte, etc.

6-6 AGP

Dado que uno de mis objetivos es simular una bandera en una PC, es necesario que incluya información referente al bus AGP.

En precursor del bus AGP es el conocido bus PCI, el cual ha sido modificado de la siguiente manera:

- Se han agregado líneas de control para extender los modos de comunicación propios del PCI.
- AGP implica un bus de comunicación que se usa solamente para conectar dos dispositivos: el controlador de memoria y la tarjeta de video.

Debo aclarar que la especificación AGP implica modos de transferencia de información. El cómo se usen para generar una escena en el monitor depende del fabricante de la tarjeta, del diseñador del sistema operativo de la computadora y del programador de aplicaciones.

La especificación vigente de AGP es la 2.0 aunque, en el momento de escribir este capítulo se está comercializando con una nueva especificación, la 3.0.

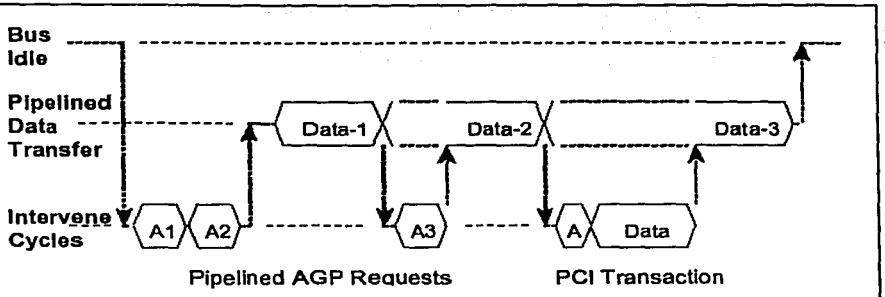


Figura 6-7
El mapa de tiempos indica la forma en que se realizan las peticiones AGP y la forma en que son atendidas. Al final del mapa se muestra una transacción PCI.

Modos de operación *DMA* y *ejecución*

AGP comprende dos modos de tratar la información necesaria para crear una escena 3D en un monitor: *DMA* y *ejecución*. También aquí debo aclarar que no estoy especificando las características de la transferencia de información.

En el **modo DMA** la memoria gráfica primaria es la memoria que está instalada en la tarjeta gráfica y a la que nos referiremos como **búfer de estructura local** o simplemente **memoria de video local**. Las estructuras 3D, texturas y demás informaciones son almacenadas inicialmente en la memoria del sistema y cuando se les requiere, son copiadas a la memoria local a través de una operación DMA. Una operación DMA implica que se transfiere un bloque de información sin interrupciones en su flujo, por lo que el bus AGP siempre estará ocupado.

El otro modo de operación se llama **modo ejecución** y parte de la siguiente consideración: *una memoria local pequeña implica un bajo costo.* Así que algunos segmentos de la memoria del sistema son dinámicamente reservados por el sistema operativo para ser empleados por el controlador gráfico y bajo el nombre de **memoria AGP** o **memoria de video no local**. El resultado neto implica que el controlador de video conserva la información más socorrida en su memoria local y cuando requiere más datos, estructuras 3D o texturas, puede acceder directamente a la información de la memoria AGP.

Finalmente, debo recordar al lector que la especificación AGP implica modos de transferencia de información. Lo descrito en los párrafos anteriores es solo una técnica para el manejo de las estructuras 3D, texturas y demás informaciones.

Transacciones PCI y AGP

La figura 6-6 nos muestra un diagrama a bloques de los circuitos empleados en los modos de comunicación de la especificación 2.0 de AGP. La especificación soporta dos modos de comunicación AGP y PCI. La figura solo muestra los circuitos que son necesarios para una transacción AGP, no obstante PCI cuenta con los mismos circuitos.

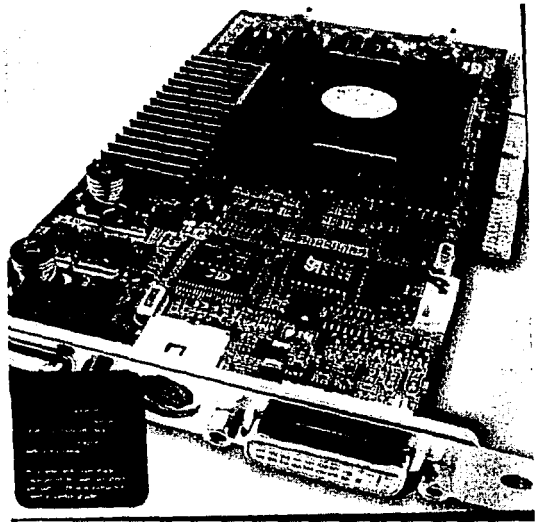


Figura 6-8
Tarjeta gráfica GeForce. Este sistema implementa una tubería gráfica completa, dejando al CPU el trabajo de calcular algunas posiciones, desplazamiento, ángulos de rotación, etc. Este sistema se puede programar con rutinas que realicen las tareas de aplicar sombras o reflexiones de superficie. Note también que su hardware incluye disipadores de calor tanto para el procesador gráfico como para la memoria. Maximum PC, Vol. 6, No. 64; editorial Imagine, Abril20001, PG. 32.

En el modo de transacción AGP, la tarjeta gráfica es el maestro y por tanto es quien realiza las peticiones de lectura y escritura. En el modo AGP, las peticiones de lectura y de escritura se almacenan en sus correspondientes pilas. Cada petición es atendida tan pronto le es posible al equipo enviar o recibir los datos.

Observe la figura 6-7. Se trata de un mapa de tiempos que nos indica que en el modo AGP es posible realizar una tercera petición apenas se ha terminado de enviar un bloque de datos para satisfacer la primera petición.

La misma figura nos habla del modo de transacción PCI. En este modo, una petición para nuevos datos debe ser atendida de inmediato, por lo que el bus no aceptará más peticiones mientras dure la transferencia de información.

Como hace el controlador de video para direccionar memoria del sistema

Cuando trabajamos en el modo ejecución, la tarjeta de video hace su solicitud de memoria al sistema operativo. Este último reserva la cantidad de memoria solicitada pero en páginas de 4KB no continúas. La pregunta ahora es ¿cómo le hace el controlador de video para direccionar estas páginas? La respuesta es simple, se emplea un GART, es decir, una **Tabla de Mapeo de Direcciones Gráficas** (GART es abreviación de *Graphics Address Remapping Table*). La tabla se implementa dentro del controlador gráfico o bien, en la memoria del sistema. La tabla, básicamente, contiene dos columnas, una de etiquetas o

índices y la otra con la dirección física de la página de 4KB. Ahora bien, generalmente, los bits más altos de una dirección de memoria son empleados como etiquetas o índices de la tabla. Una vez localizada la etiqueta en la GART, se recupera la dirección de la página. Finalmente, los bits inferiores de la dirección de memoria son empleados para direccionar el contenido de la misma página.

6-7 GeForce 3 de nVidia

El sistema GeForce 3, cuya fotografía se muestra en la figura 6-6, representa el nuevo estándar en cuanto a la implementación de la tubería gráfica: Toda la tubería se implementa en la tarjeta de video. Debido a este nuevo modo de representar una escena, las tarjetas de video son computadoras completas, que incluyen un procesador central, varios controladores de memoria, los circuitos de memoria que pueden variar desde los 32 MB hasta los 2 GB y circuitos adicionales que generan diversos tipos de señales de salida permitiendo conectar un monitor o un televisor.

El código clave del procesador gráfico GeForce 3 es NV20. Dentro del mismo hay 57 millones de instrucciones lo que lo hace el microcircuito más complejo jamás diseñado. Por comparación, el Pentium IV tiene 42 millones de transistores, mientras que el Pentium II tiene 9.5 millones.

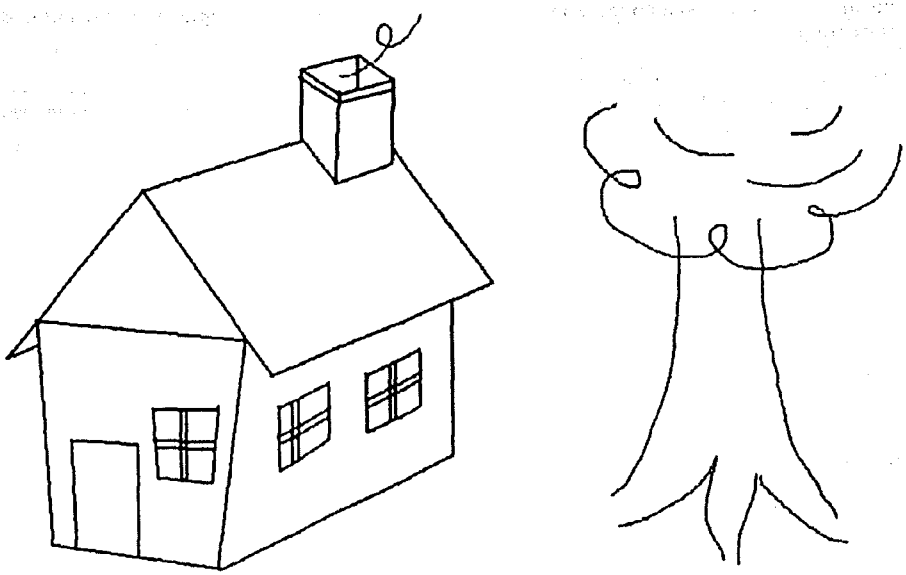
¿Velocidad de reloj? En la tarjeta se indica un tren de pulsos de 300 MHz alimentando al NV20. En tanto, la memoria funciona a 250 MHz. La memoria emplea la tecnología DDR o dobladora de reloj, por lo que podríamos considerar que funciona a 500 MHz efectivos.

Programabilidad

La **programabilidad** es la característica principal que separa a la GeForce 3 de cualquier otro acelerador de la generación anterior.

En las tuberías de representación 3D tradicionales, el CPU primero describe la forma del objeto, que va a ser representado, entonces pasa la información a la tubería del acelerador 3D, el cual describe la apariencia de la superficie del objeto. Que quede claro que operaciones como describir un objeto a partir de sus vértices, deformarlo, explotarlo, tornear un objeto y cualquier otra transformación que implique alterar la forma y propiedades de una superficie, son tarea del CPU.

En GeForce3, en vez de emplear al CPU para preparar los datos que serán pasados al acelerador 3D, un desarrollador puede cargar ahora el programa que genera la escena animada dentro del sistema GeForce y esto implica aquellas transformaciones mencionadas en el párrafo anterior.



Capítulo 7 Introducción a las primitivas de salida

Es posible describir una imagen de muchas maneras. Una de ellas implica describirla como un conjunto de objetos complejos: árboles y terreno o muebles y muros, colocados en posiciones de coordenadas específicas en escena. Las formas y los colores de los objetos se pueden describir, a nivel interno, como matrices de píxeles (texturas) y con conjuntos de estructuras geométricas básicas, como segmentos de líneas rectas y áreas de color formadas por polígonos. Entonces la escena se despliega al convertir por rastreo a las estructuras geométricas básicas en patrones de píxel.

Por lo regular, los paquetes de programación de gráficas ofrecen funciones para describir una escena en términos de estas estructuras geométricas básicas, que reciben el nombre de **primitivas de salida.** También es posible agrupar conjuntos de primitivas de salida en estructuras más complejas. Cada primitiva de salida se especifica con los datos de las coordenadas de entrada y otra información referente a la manera en como se debe desplegar ese objeto.

Los puntos y segmentos de línea recta son los componentes geométricos más simples de las imágenes. Los primitivos de salida adicionales que se pueden utilizar para crear una imagen incluyen circunferencias y otras secciones cónicas, superficies cuadráticas, curvas y superficies de *spline*, áreas de color de polígonos y cadenas de caracteres.

En el presente trabajo se describen algunos algoritmos básicos para desplegar primitivas de salida bidimensional, esto es líneas, circunferencias y superficies poligonales y amorfas. Estos algoritmos están orientados para su uso en sistemas de gráficas de rastreo, es decir, se convierte la especificación de una primitiva: coordenadas y otras

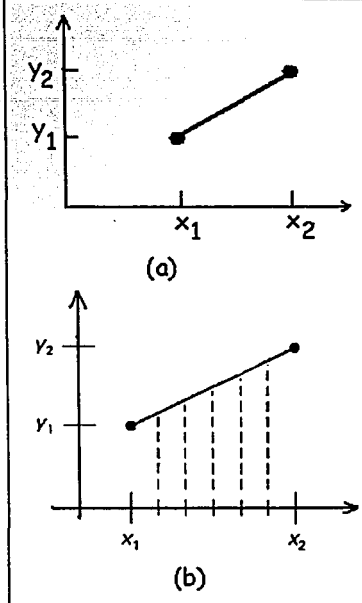


Figura 7-1
En (a) se muestra la trayectoria lineal entre las posiciones de extremos (x_1,y_1) y (x_2,y_2). En (b) se muestra un segmento de línea recta con cinco posiciones de muestreo a lo largo del eje de las x entre x_1 y x_2. Figura de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 89.

informaciones, en posiciones e intensidades de píxel que serán copiadas en el búfer de estructura.

Previamente, para cargar un valor de intensidad en el búfer de estructura, en una posición correspondiente a la columna x y a lo largo de la línea de rastreo y, supondremos que tenemos un procedimiento de bajo nivel disponible en la forma:

```
PonerPixel(x, y, intensidad : Integer);
```

A su vez requerimos de una función que nos devuelva el valor de un píxel en la posición (x, y):

```
LeerPixel(x, y : integer): integer;
```

Requerimos un procedimiento adicional que nos dibuje una línea horizontal y que llamaremos de línea rastreo. El procedimiento es el siguiente.

```
LíneaRastreo(x1, x2 , y : pmColor:Integer);
```

Se desea mayor información acerca de cualquier primitiva, puede consultar el capítulo 3 de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2ª edición, 1995.

7-1 Algoritmos para trazo de líneas

La ecuación pendiente y ordenada al origen de una línea recta es:

$$y = mx + b \tag{7-1}$$

Donde:

m : Pendiente de la línea.
b : Intersección de la línea con el eje Y.

Dado que una primitiva, como lo es la línea, se especifica con dos posiciones extremas (x_1, y_1) y (x_2, y_2) tal como indica la figura 7-1. a. Podemos determinar los valores para m y b como:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$
$$b = y_1 - m \cdot x_1 \tag{7-2}$$

Los algoritmos para trazar segmentos rectos se basan en los cálculos de las ecuaciones anteriores, o sea, la 7-1 y la 7-2. En general, existen dos algoritmos de conversión por rastreo altamente eficientes para el trazo de una línea en el búfer de estructura.

- El analizador diferencial digital o DDA.
- Algoritmo de línea de Bresenham

En ambos algoritmos se efectúa el muestreo de una coordenada en incrementos unitarios y se determinan los valores enteros correspondientes a la otra coordenada de tal manera

que las parejas de coordenadas estén lo más próximas a la trayectoria de la línea. Para ejemplificar, observe la figura 7-1.b.

Ambos algoritmos se desarrollarán a partir del concepto de una línea cuya pendiente varía de cero a cuarenta y cinco grados; de esta forma, a cada muestra de la coordenada *x* le corresponde uno y sólo un valor de la coordenada *y*.

Finalmente, es posible extender estos algoritmos para que describan segmentos rectos de cualquier pendiente, sin embargo, esta actividad se deja al lector.

El analizador diferencial digital (DDA)

Consideremos dos puntos muestra consecutivos (x_k, y_k) y (x_{k+1}, y_{k+1}). Entonces notamos que se cumple con la siguiente relación.

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \qquad (7\text{-}3)$$

Puesto que llevamos a cabo un muestreo unitario en *x*, es decir $x_{k+1} - x_k = 1$ podemos expresar la ecuación 7-3 como

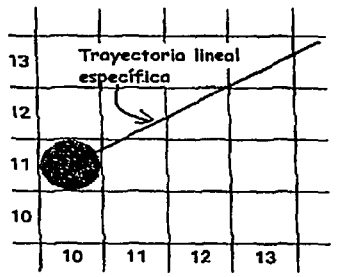
$$y_{k+1} = y_k + m \qquad (7\text{-}4)$$

Los valores iniciales serán $(x_k, y_k) = (x_1, y_1)$ y el subíndice *k* toma valores enteros consecutivos desde *1*. Para líneas con una pendiente positiva mayor de 45 grados, se realiza un muestreo unitario de *y*, es decir $y_{k+1} - y_1 = 1$ y se calcula cada valor sucesivo de *x* como:

$$x_{k+1} = x_k + \frac{1}{m} \qquad 7\text{-}5)$$

Bueno, el algoritmo de línea de rastreo es fácil de comprender y su implementación es sencilla; sin embargo, tiene la desventaja de trabajar con números en punto flotante. Tal situación ralentiza al algoritmo. A continuación tenemos el programa correspondiente.

```
Procedure LíneaDDA ( x1, y1, x2, y2 : Integer);
Var
    dx, dy, pasos, k    :   integer;
    Xinc, yinc, x, y    :   real;
Begin
 dx := x2 – x1;
 dy := y2 – y1;
 if abs(dx) > abs(dy) then
    pasos = abs (dx)
  else
    pasos = abs(dy);
 PonerPixel ( round(x), round (y), 1);
 for k=1 to steps do begin
   x := x + xInc;
   y := y + yInc;
 end;
end;
```



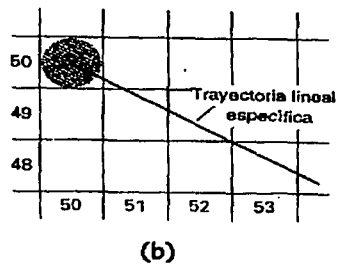


Figura 7-2
Secciones de una pantalla de despliegue donde se deben trazar segmentos de línea recta. Figura de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 89.

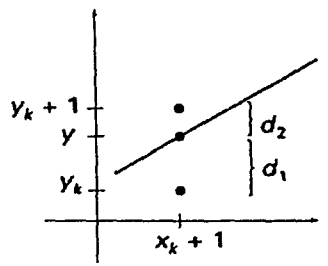


Figura 7-3
Distancias entre posiciones de píxel y la coordenada de *y* de la línea en la posición de muestreo x_{k+1}. Figura de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 93.

Algoritmo de línea de Bresenham

Un algoritmo preciso y efectivo para la generación de líneas de rastreo, desarrollado por Bresenham, convierte mediante rastreo, las líneas al utilizar sólo cálculos incrementales con enteros que se pueden adaptar para desplegar circunferencias y otras curvas. La figura 7-2 ilustra secciones de una pantalla de despliegue donde se deben trazar segmentos de línea recta. Los ejes verticales muestran las posiciones de las líneas de rastreo y los ejes horizontales identifican las columnas de píxel.

En este algoritmo, al realizar un muestreo de *x* en intervalos unitarios, necesitamos decidir cuál de dos posibles posiciones de píxel está más próxima a la trayectoria de la línea en cada paso del muestreo. Veamos como es esto, consideremos la línea de la figura 7-2.a. Iniciamos el muestreo desde el extremo izquierdo de la línea, es decir, en la posición *(10,11)*, ahora, la siguiente posición de muestreo corresponde a $x = 11$, en consecuencia, necesitamos decidir si trazamos el píxel en *(11,11)* o en *(11,12)*.

De manera similar, en la figura 7-2.b aparece la trayectoria de una línea con pendiente negativa que inicia a partir del extremo izquierdo en la posición de píxel *(50,50)*. En este caso ¿seleccionamos la siguiente posición de píxel como *(51,50)* o como *(51,49)*?

El algoritmo de línea de Bresenham responde a estas preguntas al probar el signo de un parámetro entero, cuyo valor es proporcional a la diferencia entre las separaciones de las dos posiciones de píxel respecto de la trayectoria de la línea. La figura 7-3 ilustra este concepto.

El algoritmo de Bresenham, para el caso de líneas con pendiente entre 0 y 45 grados se puede resumir en el siguiente listado:

```
Procedure LineaBresenham(x1, x2, y1, y2 : Integer);
Var
 dx, dy, x, y, xfin : Integer;
//
// p : parámetro de decisión
//
Begin
//
//Se determinan los extremos izquierdo y derecho de la línea. En el extremo izquierdo se
//almacenan en las variables (x,y) (representan un punto cualquiera de la línea) y la abcisa del
//extremo derecho se almacena en xfin.
  If x1 > x2 then begin
     x := x2;
     y := y2;
     xfin := x1;
  end else begin
     x := x1;
     y := y1;
     xfin := x2;
  end;
//
//Calcular constantes
  dx := abs ( x2 - x1 );
  dy := abs ( y2 - y1 );
//
//Se obtiene el valor inicial para el parámetro de decisión como
  p := 2 * dy - dx;
//
//Se traza el primer pixel
  PonerPixel(x,y);
//
  While x < xfin do begin
     x := x + 1;
     if p < 0 then begin
          PonerPixel(x,y);
```

```
        p := p + 2 * dy;  {entonces el valor siguiente del parámetro de decisión se calcula como}
    end else begin
        y := y + 1;
        PonerPixel(x,y)
        p := p + ( 2 * dy ) - ( 2 * dx ); {entonces el valor siguiente del parámnetro de decisión se
    calcula como}
    end;
  end;
//
End;
```

7-2 Algoritmo de circunferencia Bresenham

Una circunferencia se define como un conjunto de puntos que se encuentran, en su totalidad, a una distancia *r* de una posición central (x_C, y_C). La figura 7-4.a ilustra este concepto. Para nuestro algoritmo consideraremos que el centro de la circunferencia coincide con el origen del sistema de coordenadas, lo cual facilita la implementación del algoritmo.

Es posible reducir el cálculo de estas posiciones de píxel a lo largo de la circunferencia al considerar la simetría de la misma. Si consideramos una circunferencia cuyo centro coincide con el origen del sistema de coordenadas, notamos que existe simetría respecto de los ejes de las ordenadas y de las abcisas. Podemos llevar esto un paso más adelante y señalar que también hay simetría entre octantes: Las secciones circulares en octantes adyacentes son simétricas respecto de la línea de 45 grados que los divide.

Así pues, habiendo considerado la simetría de la circunferencia, podemos generar todas las posiciones de píxel calculando solo los puntos dentro del sector *x=0* a *x=y*. La figura 7-4.b muestra como el cálculo de un punto en un octante resulta en puntos en los otros siete octantes.

A lo largo de la sección circular de *x=0* y *x=y*, en el primer cuadrante, la pendiente de la curva varía entre 0 y 45 grados, por tanto podemos tomar pasos unitarios en la dirección positiva de *X* en ese octante y utilizar un parámetro de decisión, tal como se hizo en el algoritmo de línea de Bresenham, para determinar cuál de las dos posiciones posibles de *y*: (x_{k+1}, y_k) o (x_{k+1}, y_{k-1}), está más próxima a la trayectoria de la circunferencia en cada paso. Las posiciones de los puntos en los otros siete octantes se obtienen entonces por simetría. La figura 7-5 ilustra este concepto.

```
Procedure Circunferencia Bresenham (xCentro, yCentro, Radio: Integer);
Var
X, y, p : integer;      //p es el parámetro de decisión

Procedure GraficaPuntos;
begin
  PonerPixel( xCenter + x, yCenter + y, 1);
  PonerPixel( xCenter - x, yCenter + y, 1);
  PonerPixel( xCenter + x, yCenter - y, 1);
  PonerPixel( xCenter - x, yCenter - y, 1);

  PonerPixel( xCenter + y, yCenter + x, 1);
  PonerPixel( xCenter - y, yCenter + x, 1);
  PonerPixel( xCenter + y, yCenter - x, 1);
  PonerPixel( xCenter - y, yCenter - x, 1);
end;

Begin
  x:=0;
```

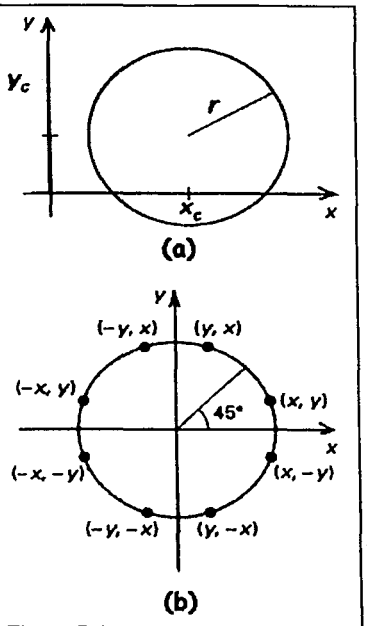


Figura 7-4
En (a) hay una circunferencia con coordenadas de centro (x_c, y_c) y radio *r*. En (b) del cálculo de un punto en la circunferencia se pueden obtener otros siete puntos siguiendo la simetría de la misma. Figura de Donald Hearn, M. Pauline Baker, "Gráficas por computadora", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pgs. 100 y 101.

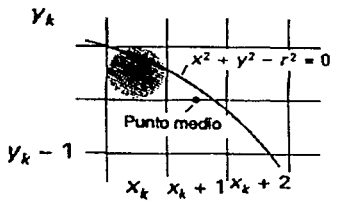


Figura 7-5
Punto medio entre píxeles candidatos en la posición de muestreo x_{k+1} a lo largo de una trayectoria circular. Figura de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 102.

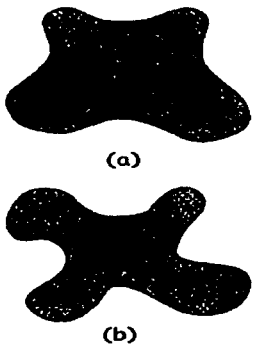


Figura 7-6
Ejemplos de fronteras de color para un procedimiento de llenado de fronteras. Figura de Donald Hearn, M. Pauline Baker, "Gráficas por computadora", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 133.

```
 y:=Radio;
 GraficaPuntos;
//
//Determina el valor inicial del parámetro de decisión
 p := p – radio;
//
 While x < y do begin
    If p<0 then
        x := x + 1;          //Punto dentro de la circunferencia
    else begin
        x := x + 1;          //Punto fuera de la circunferencia
        y := y – 1;
    end;
//
//Determinar el siguiente valor del parámetro de decisión
    If p < 0 then
        p := p + 2 * x – 1;
     else
        p := 2 * ( x + y ) + 1;
    GraficaPuntos
 end; //While
End;
```

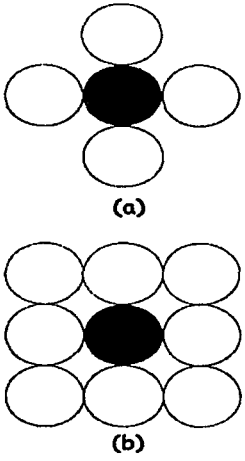


Figura 7-7
Métodos de llenado que se aplican en 4 regiones conectadas (a) y en 8 regiones conectadas (b). Los círculos en blanco representan píxeles que se deben probar desde la posición de prueba actual, que se muestra como un círculo en negro. Figura de Donald Hearn, M. Pauline Baker, "Gráficas por computadora", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 133.

7-3 Primitivas de llenado de área

Una primitiva de salida estándar, en los paquetes de gráficas generales, es un área de polígono de patrón o de color sólido. En ocasiones están disponibles otras clases de primitivas de áreas, pero es más fácil procesar polígonos que tienen fronteras lineales.

Hay dos planteamientos básicos para el llenado de áreas en sistemas de rastreo. Una manera de llenar un área consiste en trazar líneas de rastreo de una arista a otra. Otro método para el llenado de área es iniciar desde una posición determinada inicial y pintar hacia afuera desde ese punto hasta encontrar las condiciones de frontera específicas. El planteamiento de la línea de rastreo se utiliza, por lo regular en paquetes de gráficas generales para llenar polígonos, circunferencias, elipses y otras curvas simples. Los métodos de llenado que empiezan desde un punto interior son útiles con fronteras más complejas y en sistemas interactivos de pintura.

7-4 Algoritmo para llenar fronteras, método de las regiones conectadas.

Un algoritmo para llenar fronteras acepta como entrada una coordenada de un punto interior (x, y), un color de llenado y un color de frontera. Al iniciar desde (x, y), el procedimiento prueba regiones cercanas para determinar si son del color de la frontera. Si no es así, se pintan con el color de llenado y se prueban sus posiciones vecinas. Este procedimiento continúa probando todos lo píxeles hasta el color de la frontera del área. En la figura 7-6 aparecen algunos ejemplos de definición de regiones para llenar fronteras.

En la figura 7-7 se presentan dos métodos para proceder a probar píxeles cercanos desde la posición de prueba actual. En la figura 7-7.a se prueban cuatro puntos vecinos. Estas son las posiciones de píxel que se localizan a la derecha, a la izquierda, arriba y abajo del píxel actual. Este procedimiento se conoce como el método de las cuatro regiones conectadas. El segundo método se muestra en la figura 7-7.b y se utiliza para llenar

figuras más complicadas. A este último método se le conoce como el método de las 8 regiones conectadas.

En la figura 7-8 se ilustra una frontera a llenar con el método de las cuatro regiones conectadas. Note que la frontera no pudo ser coloreada en su totalidad con ese método. En cambio, de aplicarse el algoritmo de las 8 regiones conectas, la frontera sería correctamente coloreada.

```
Procedure LlenarFrontera (x, y, ColorRelleno, ColorFrontera : integer);
Var
    ColorActual:integer;
Begin
    ColorActual:0Getixel( x, y);
    if ( ColorAtual <> ColorFrontera) and (ColorActual <>ColorRelleno ) then begin
        PonerPixel (x, y, colorRelleno);
        LlenarFrontera (x + 1    ,y        , ColorRelleno, ColorFrontera);
        LlenarFrontera (x – 1    ,y        , ColorRelleno, ColorFrontera);
        LlenarFrontera (x              ,y + 1    , ColorRelleno, ColorFrontera);
        LlenarFrontera (x              ,y -1     , ColorRelleno, ColorFrontera);
    End;
End;
```

El método de las 8 regiones conectadas se deja como ejercicio al lector y debo aclarar que es realmente fácil de implementar una vez que se conoce el método de las cuatro regiones conectadas.

Dado que estos algoritmos son recursivos, pueden examinar todos los píxeles dentro de un área. El problema principal implica que, para examinar cada píxel, el algoritmo debe invocarse así mismo, en consecuencia, almacena una dirección de retorno por cada píxel. Las direcciones de retorno se guardan en un segmento de memoria llamado pila, que en total podría guardar entre 4000 y 8000 direcciones. Como ve, esto no resulta suficiente cuando el área a llenar es muy grande.

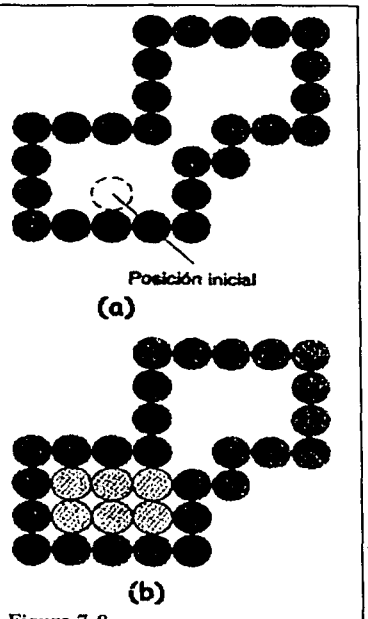


Figura 7-8
El área definida dentro de la frontera de color (a) se llena solo en parte (b) al utilizar un algoritmo para llenar fronteras de 4 regiones conectadas. Figura de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 134.

7-5 Llenado de polígonos con líneas de rastreo: estructuras de datos

Un tipo de algoritmo para llenar fronteras acepta como entrada una colocación de vértices, correspondientes a un polígono, y un color de llenado. El algoritmo traza líneas horizontales, que llamaremos **líneas de rastreo**, de una arista a otra hasta llenar la frontera. La figura 7-9, nos muestra un cuadrilateral cuya superficie se representa con líneas de rastreo.

Previamente a la descripción del algoritmo se requieren conocer algunas definiciones sobre las estructuras de datos[1] que se emplean. Todas las estructuras empleadas corresponden a la definición de clases[2].

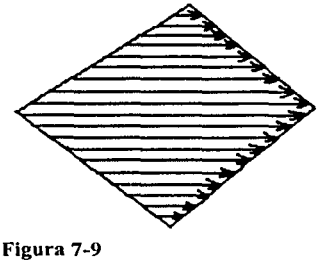


Figura 7-9
La figura muestra un cuadrilateral que ha sido llenado con líneas de rastreo.

[1] Una estructura de datos es un bloque de información y un conjunto de reglas de acceso a esa información.

[2] Una "clase" es un tipo de estructura de datos muy socorrido por los programadores de hoy e implica agrupar varios tipos de datos, para mayor información puede consultar el capítulo 25 del libro Herbert Schildt "*Turbo C/C++ manual de referencia*", McGraw Hill, 1995.

Vértice

Un vértice quedará plenamente especificado por una pareja de coordenadas (x, y), así que la definición de clase del vértice es la siguiente:

```
tVértice = class
x, y : Integer;
Constructor Create (x,y: integer);
End;
```

Por ahora nos vamos a ahorrar la información sobre el color.

Polígono

Un polígono quedará especificado por una lista de vértices, por lo que debemos considerar la siguiente definición:

```
tPolígono = class (tList)
```

En esta simple línea, nuestra definición de polígono está heredando todas las características de una clase ya definida. Esta clase se llama *tList* y actúa según su nombre, es decir, como una lista en la que podemos guardar la dirección en memoria de nuestra información importante. Adicionalmente se requieren métodos que faciliten el acceso a cada vértice en la lista, por lo que la definición completa es:

```
tPolígono = class (tList)
Procedure AgregaVértice (pmVértice : tVértice);
Function RegresaVértice 8Indice: Integer);
Property Vértice [Indc : Integer] tVertice read RegresaVértice;
End;
```

El método *RegresaVértice* y la propiedad *Vértice*, correspondientes a la definición de clase *tPolígono*, facilitarán el acceso a la información de cualquiera de los vértices de la lista. Para ilustrar veamos el siguiente ejemplo:

```
Program Prueba;
Type
tVértice = class
  // Definición de vértice
end;

tPolígono = class
  // Definición de polígono
end;

Var
  Vértice1, Vértice 2, Vértice3    : tvértice
  Triángulo                        : tPolígono;

Begin
//
//Definiendo vértices
  Vértice1 : = tvértice.Create(0,10);
  Vértice2 : = tvértice.Create(5,0);
  Vértice3 : = tvértice.Create(10,10);
//
//Agregando los vértices a la lista del triángulo
  Triágnulo.AgregaVértice(Vértice1);
  Triágnulo.AgregaVértice(Vértice2);
  Triágnulo.AgregaVértice(Vértice3);
//
```

```
End;
```

Se requieren otros dos tipos de estructuras de datos, las cuales son empeladas por el algoritmo y no por el usuario.

Una arista

Dado que una arista es uno de los lados del polígono, ésta se especifica con dos vértices y una etiqueta de clasificación. La definición de una arista también implica dos métodos que permiten encontrar la intersección de la misma con una línea de rastreo.

```
tTipoArista = (Izquierda, Derecha, Horizontal)

tArsita = class
//
  tipo        : tTipoArista;
  VérticeA    : tVértice;
  VérticeB    : tVértice;
//
  Constructor CrearArista(pmVérticeA,pmVérticeB : tVértice);
  Function x (x, y : integer) : integer;
  Function ICLR (y : integer) : bolean;
End;
```

Las definiciones de *VérticeA* y *VérticeB* especifican los extremos de la arista. La *función x* retorna la abcisa de la intersección de la arista con una línea de rastreo $y=k$. La *función ICLR* retorna un valor lógico indicando si hay intersección o no entre una arista y una línea de rastreo.

Tabla de aristas

Para el desarrollo del algoritmo se considera que las aristas deben ser clasificadas y enlistadas. Así que una tabla de aristas se hace necesaria. La definición de una tabla de aristas es como sigue:

```
tTablaArsitas = class (tList)
    Procedure AgregarArista ( pmArista : tArista);
    Function RegresaArista ( Índice : integer): tArista;
    Property Aristas [Índice : Integer] : tArista    read RegresaArista;
End;
```

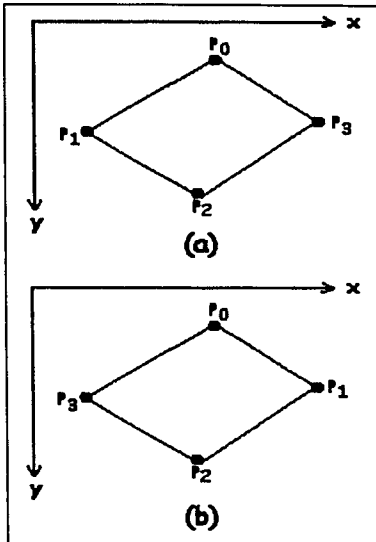


Figura 7-10
Numeración de los vértices de un polígono. La numeración de los vértices debe ser en orden ascendente según se muestra en los esquemas de la figura.

7-6 Llenado de polígonos con líneas de rastreo: Descripción del algoritmo

Este algoritmo acepta como entrada la definición de un polígono y un color de llenado. En este caso un polígono se define con tres o más vértices.

El algoritmo primero clasifica las aristas del polígono en izquierdas, derechas y horizontales. En el polígono de la figura 7-10.a las aristas P_0P_1 y P_1P_2 son izquierdas por que *miran* a la derecha. En tanto que las aristas P_2P_3 y P_3P_1 son derechas porque *miran* a la izquierda.

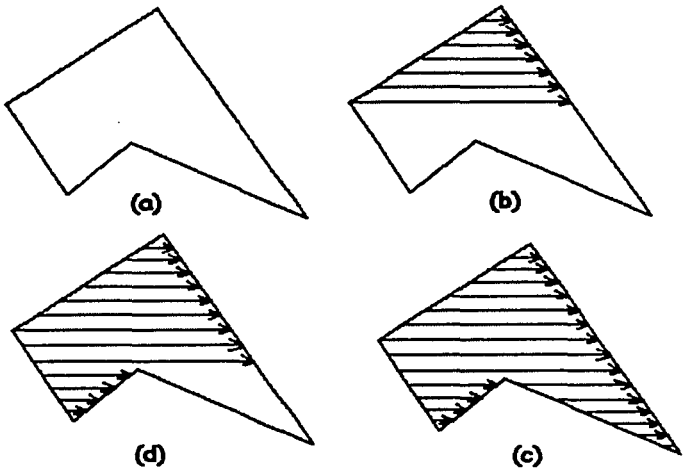


Figura 7-11
Los esquemas de la figura muestran el proceso de llenado de un polígono de aristas rectas. En (a) se muestra e4l polígono a llenar empleando una arista izquierda. En (c) ya se han empleado dos aristas izquierdas. Finalmente en (d) el polígono está completamente coloreado.

Note la forma en que se numeran los vértices. Resumiendo, los vértices de un polígono se deben numerar en orden ascendente según se puede apreciar en cualquiera de los polígonos de la figura 7-10.

Una vez clasificadas las aristas, se trazan líneas de rastreo desde las aristas izquierdas hasta la primera arista derecha que intercepten. Para ilustrar el algoritmo vamos a llenar el polígono mostrado en la figura 7-11.a. El polígono en 7-11.b ilustra el llenado del polígono considerando una sola arista izquierda. El polígono en 7-11.c ilustra el mismo proceso de llenado considerando la segunda arista izquierda. Finalmente, el polígono en 7-11.d ha sido llenado completamente.

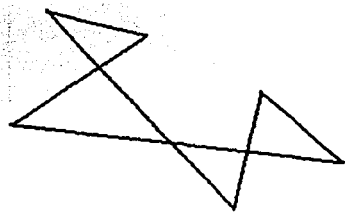


Figura 7-12
Polígono que se intercepta así mismo. El método de llenado de polígono mediante la clasificación de aristas falla con este tipo de figuras.

Este algoritmo tiene una limitación seria. Falla con polígonos que se interceptan a sí mismos, tal como el polígono de la figura 7-12.

Este algoritmo, así como cualquiera de los mencionados en este capítulo pueden ser encontrados en Donald Hearn y M. Pauline Baker, "*Gráficas por Computadora*", Prentice Hall, 2ª edición, 1995. En este libro, en su capítulo 3, están descritos todo tipo de algoritmos de salida, excepto el que corresponde al llenado de triángulos.

```
Procedure LlenarPolígono ( pmPolígono : tPolígono , pmColor: Integer);
Var
    Indc, Jndc, x1, x2    : integer;
    Uv                        : integer;    // último vértice
    TablaAristas              : tTablaAristas;
    AristaAuxiliar        : tArista;
    Buscar                    : Boolean;
Begin
//
//   Inicializar variables
    TablaAristas := tTablaArsitas.Create;
Indc := 0;
```

```
//
//  PRIMERA PARTE : clasificar aristas
//
    While Indc <= pmPolígono.NumVertices – 2 do begin
//
//      Crear una arista auxiliar
        AristaAuxiliar := tAristas.Create.
//
//      Clasificar las aristas
        If ( pmPoliogno.Vértice [ Indc ].y > pmPoliogno.Vértice [ Indc + 1 ].y then begin
            AristaAuxiliar.Tipo       := Izquierda;
            AristaAuxiliar.VérticeA   := pmPoliogno.Vértice [ Indc ];
            AristaAuxiliar.VérticeB   := pmPoliogno.Vértice [ Indc + 1 ];
        end else
        If ( pmPoliogno.Vértice [ Indc ].y < pmPoliogno.Vértice [ Indc + 1 ].y then begin
            AristaAuxiliar.Tipo       := Derecha;
            AristaAuxiliar.VérticeA   := pmPoliogno.Vértice [ Indc + 1 ];
            AristaAuxiliar.VérticeB   := pmPoliogno.Vértice [ Indc ];
        end else
//        ( pmPoliogno.Vértice [ Indc ].y > pmPoliogno.Vértice [ Indc + 1 ].y then begin
            AristaAuxiliar.Tipo       := Horizontal;
            AristaAuxiliar.VérticeA   := pmPoliogno.Vértice [ Indc ];
            AristaAuxiliar.VérticeB   := pmPoliogno.Vértice [ Indc + 1 ];
        End;
//
//      Agregar la arista auxiliar a la tabla
        TablaAristas.AgregarArista (AristaAuxiliar);
        AristaAuxiliar:=Nil;
//
        Indc := Indc + 1;
    End;
//
//  Crear una arista auxiliar
    AristaAuxiliar := tArista.Create;
//
//  Clasifica la arista que une los vértice primero y último
    uv := pmPolígono.NumVértices – 1; // Último vértice
    If ( pmPoliogno.Vértice [ uv ].y > pmPoliogno.Vértice [ 0 ].y then begin
        AristaAuxiliar.Tipo       := Izquierda;
        AristaAuxiliar.VérticeA   := pmPoliogno.Vértice [ uv ];
        AristaAuxiliar.VérticeB   := pmPoliogno.Vértice [ 0 ];
    end else
    If ( pmPoliogno.Vértice [ uv ].y < pmPoliogno.Vértice [ 0 ].y then begin
        AristaAuxiliar.Tipo       := Derecha;
        AristaAuxiliar.VérticeA   := pmPoliogno.Vértice [ 0 ];
        AristaAuxiliar.VérticeB   := pmPoliogno.Vértice [ uv ];
    end else
//    ( pmPoliogno.Vértice [ uv ].y > pmPoliogno.Vértice [ 0 ].y then begin
        AristaAuxiliar.Tipo       := Horizontal;
        AristaAuxiliar.VérticeA   := pmPoliogno.Vértice [ uv ];
        AristaAuxiliar.VérticeB   := pmPoliogno.Vértice [ 0 ];
    End;
//
//SEGUNDA PARTE: Llenar un área con líneas de rastreo
//
    For Indc := 0 to TablaAristas.NumAristas –1 do
        If TablaAristas.Arsitas [indc] = Izquierda then
//
//          Crear una línea de rastreo
            For y := TablaAristas.Aristas[Indc].VérticeA.y to TablaAristas.Aristas[Indc].VérticeB.y do
begin
//
//              Busca arista derecha
                Jndc := Indc;
                Buscar := True;
                While Buscar do begin
                    If Jndc = TablaÇAristas.NumAristas –1 then Jndc:=0 else Jndc:=Jndc+1;
                    If TablaARistas.Arista[Jndc].ICLR(y) then Buscar:=False;
                End;
//
```

```
//              Busca extremos de la línea de rastreo
                x1 := TablaAristas.Arista[Indc].x;
                x2 := TablaAristas.Arista[Jndc].x;
//
//              Traza Línea de rastreo
                LíneaRastreo(x1,x2,pmColor);
//
            end;
//
//  Borra la tabla de aristas
    TablaArsitas.Free;
End;
```

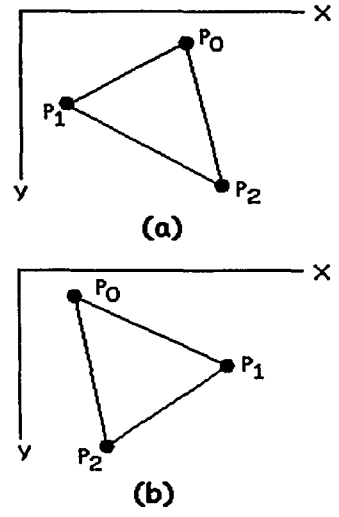


Figura 7-13
La figura ilustra la forma en que pueden numerarse los vértices de un triángulo.

7-7 Llenado de fronteras triangulares

Este segmento está dedicado a un tema que es de suma importancia en el mundo de los subsistemas de video. El estado del arte para cualquier tarjeta de video que se digne de ser 3D implica la capacidad de dibujar millones de superficies triangulares por segundo.

Este algoritmo acepta como entrada tres vértices, los cuales corresponden a un triángulo, y un color de llenado. El algoritmo primero clasifica las aristas del triángulo en izquierdas, derechas y horizontales. En el triángulo de la figura 7-13.a, las aristas P_0P_1 y P_1P_2 son izquierdas porque *miran* a la derecha. En tanto que la arista P_2P_0 es derecha porque *mira* a la izquierda. Note también la forma en que se numeran los vértices. Resumiendo, los vértices de un triángulo se pueden numerar en cualquier orden. El triángulo 7-13.b ilustra el otro caso posible.

Una vez clasificadas las aristas del triángulo, se reclasifican en aristas fuente y arista destino, sí, leyó bien, una arista destino. Las aristas fuente son aquellas desde las cuales parten las líneas de rastreo. La arista destino es aquella hacia la cual llegan las líneas de rastreo.

En un triángulo se presentan las siguientes situaciones:

- 2 aristas izquierdas, 1 arista derecha.
- 1 arista izquierda, 2 aristas derechas.
- 1 arista izquierda, 1 arista derecha y 1 arista horizontal.

La forma de reclasificar las aristas en fuente y destino es considerando su número. Si tenemos dos aristas izquierdas, éstas serán las aristas fuente. En caso contrario, si tenemos dos aristas derechas, éstas serán las aristas fuente.

A continuación, se puede llenar el triángulo con líneas de rastreo. Veamos un ejemplo, el triángulo de la figura 7-14.a debe ser llenado con líneas de rastreo. El triángulo en 7-14.b ilustra el llenado del triángulo considerando una sola arista fuente. Finalmente, el triángulo en 7-14.c ha sido llenado completamente.

```
Procedure LlenarTriángulo (VrtcA, VrtcB, VrtcC : tVértice);
Var
    AristasIzquierdas     : tTablaAristas;
    AristasDerechas       : tTablaAristas;
    AristasFuente         : tTablaAristas;
    AristasDestino        : tTablaAristas;
    Nai                   : integer;        //Número de aristas izquierdas
    Nad                   : integer;        //Número de aristas derechas
    Naf                   : integer;        //Número de aristas fuente
    Indc                  : integer;        //Índice
Begin
```

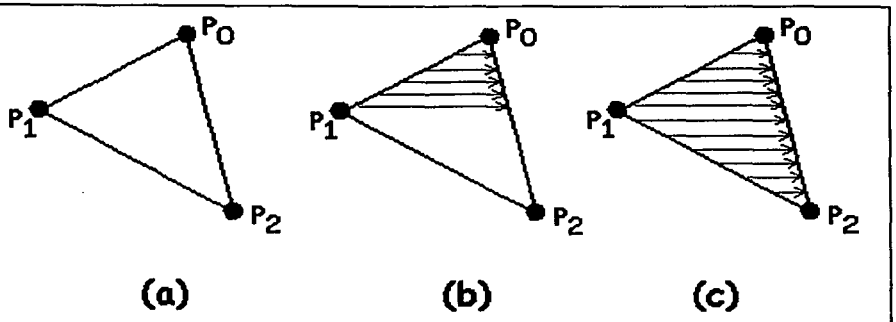
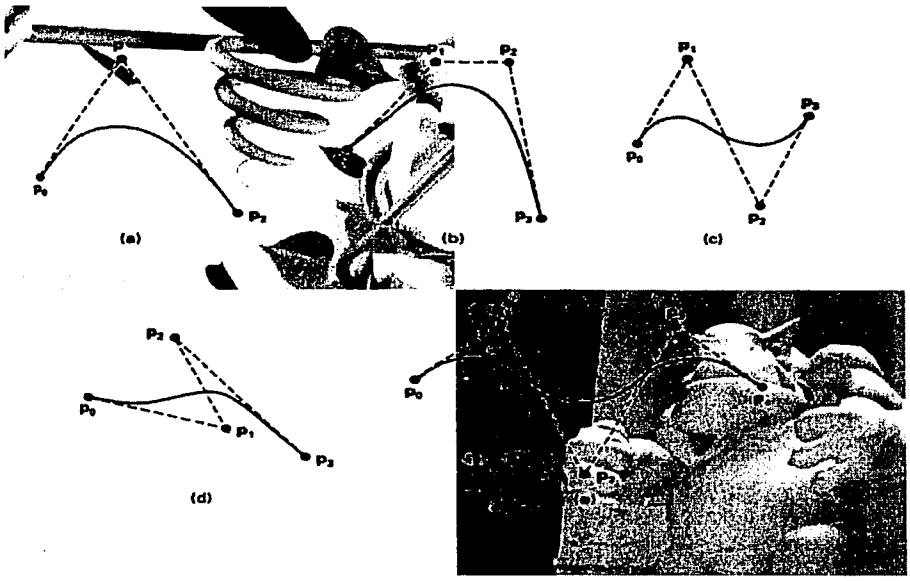


Figura 7-14
Esta figura ilustra las etapas del llenado de un triángulo. En (a) tenemos un triángulo que debe ser coloreado. En (b), el triángulo se ha coloreado considerando una sola arista izquierda. En (c), el triángulo está completamente lleno.

```
//
//   Iniciando variables
     nai      :=0;
     nad:=0;
     AristasDerechas    := tTablaAristas.Create;
     AristasIzquierdas  := tTablaAristas.Create;
//
//PRIMERA PARTE : clasificar las aristas en izquierdas y drechas
//
     If VrtcA.y < VrtcB.y then begin
          nai := nai + 1;
          AristasIzquierdas.AgregarAristas (tAristas.CrearArista(VrtcA,VrtcB);
     end else
     If VrtcA.y > VrtcB.y then begin
          nad := nad + 1;
          AristasDerechas.AgregarAristas (tAristas.CrearArista(VrtcA,VrtcB);
     End;
//
     If VrtcB.y < VrtcC.y then begin
          nai := nai + 1;
          AristasIzquierdas.AgregarAristas (tAristas.CrearArista(VrtcB,VrtcC);
     end else
     If VrtcB.y > VrtcC.y then begin
          nad := nad + 1;
          AristasDerechas.AgregarAristas (tAristas.CrearArista(VrtcB,VrtcC);
     End;
//
     If VrtcC.y < VrtcA.y then begin
          nai := nai + 1;
          AristasIzquierdas.AgregarAristas (tAristas.CrearArista(VrtcC,VrtcA);
     end else
     If VrtcC.y > VrtcA.y then begin
          nad := nad + 1;
          AristasDerechas.AgregarAristas (tAristas.CrearArista(VrtcC,VrtcA);
     End;
//
//   Verificar que dos o tres vértice del triángulo no estén concentrados en un punto
     If (nai + nad) <2 then bejín
          AristasDerechas.Free;
          AristasFuente.Free;
     End;
//
//SEGUNDA PARTE: Reclasificar las aristas en fuente y destino
//
     If (nai >= nad) then begin
          naf := nai;
```

```
        AristasFuente := Aristaszquierdas;
        AristasDestino := ArsitasDerechas;
    End else begin
        Naf := nad;
        ArsitasFuente := AristasDerechas;
        ArsitasDestino := ArsitasIzquierdas;
    End;
//
//TERCERA PARTE: Llenado de área
//
        For Indc := 0 to AristasFuente.NúmeroArsitas – 1 do
            For y := AristasFuente.Aristas[Indc].VérticeA.y to AristasFuente.Aristas[Indc].VérticeB.y
do begin
//              Encontrar extremos de la lñinea de rastreo
                x1:= AristasFuente.Arista[Indc].x(y);
                x2:= AristasFuente.Aristas[0].x(y),
                LineaRastreo (x1,x2,y,pmColor);
            End;
//
//  Borrar tablas de aristas
    AristasIzquierdas.Free;
    AristasDerechas.Free;
    AristasFuente.Free;
    AristasDestino.Free;
//
End;
```



Capítulo 8 Introducción a las curvas spline

En la terminología del dibujo mecánico, una *spline* es una banda flexible que se utiliza para producir una curva suave a través de un conjunto de puntos designados. Varios pesos pequeños se distribuyen a lo largo de la banda para mantenerla en posición sobre la mesa de dibujo mientras se traza la curva.

En gráficas por computadora, el término curva de spline ahora se refiere a cualquier curva compuesta, es decir, que se forma con secciones curvas que satisfacen condiciones específicas de continuidad en la frontera de las piezas.

Según lo anterior, podemos describir un tipo de esta curva en forma matemática como una función polinómica paramétrica cuyas primera y segunda derivadas son continuas a través de las distintas secciones de la curva.

Ahora bien, podemos escribir cada sección de spline como un conjunto de funciones de coordenadas paramétricas de la forma:

$$x = x(u), \quad y = y(u), \quad z = z(u); \quad u_1 \le u \le u_2 \qquad (8\text{-}1)$$

Las ecuaciones paramétricas 8-1 describen una curva que pasa por dos puntos de control p_k y p_{k+1}. La figura 8-1 nos ilustra al respecto.

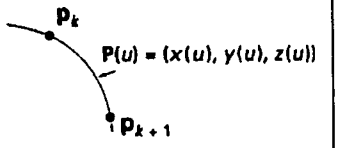


Figura 8-1
Función de punto paramétrico *P(u)* para una sección de curva de Hermite entre los puntos de control p_k y p_{k+1}. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2ª edición en español, 1995. pg. 338.

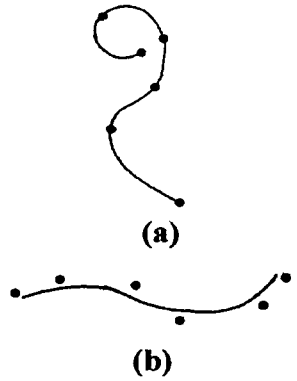


Figura 8-2
Conjuntos de 6 puntos que se interpolan en (a) y se aproximan en (b). En ambos casos se usaron secciones polinómicas continuas. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 332.

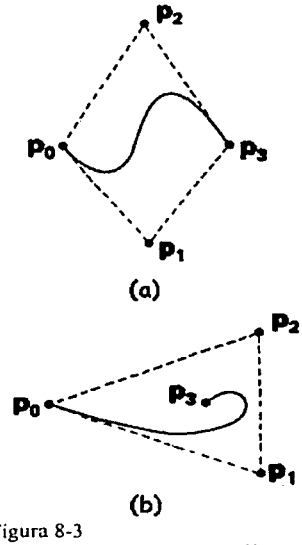


Figura 8-3
Formas de caso convexo (líneas de rayas) par dos conjuntos de puntos de control. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 333.

Puntos de control

Especificamos una curva de spline al proporcionar un conjunto de posiciones de coordenadas que se conocen como **puntos de control**. Estos puntos indican la forma general de la curva al ajustarse con funciones polinómicas paramétricas continuas

Interpolación y aproximación

Cuando las secciones polinómicas se ajustan de modo que la curva pasa a través de cada punto de control, como se indica en la figura 8-2.a, se dice que la curva que resulta realiza una interpolación del conjunto de puntos de control. Por otra parte, cuando los polinomios se ajustan a la trayectoria general de los puntos de control sin pasar necesariamente a través de alguno de ellos, se dice que la curva que resulta se aproxima al conjunto de puntos de control, figura 8-2.b.

Casco convexo

La frontera de un polígono que encierra un conjunto de puntos de control se conoce como **casco convexo**. Una forma de imaginar un casco convexo considera una banda de hule que se estira alrededor de las posiciones de los puntos de control de modo que cada uno de éstos se encuentra, ya sea en el perímetro del casco o dentro de éste. La figura 8-3 nos da un ejemplo. Por lo general, las curvas de spline están limitadas por el casco convexo, lo cual indica que los polinomios siguen con suavidad los puntos de control sin oscilaciones erráticas.

Gráfica de control

Por lo general, los puntos de control, para una curva de spline, se unen con líneas punteadas. Esto tiene el objeto de recordar al diseñador el orden de los puntos de control. Ahora bien, Este conjunto de segmentos de línea conectados se conoce como la **gráfica de control** de la curva. Otros nombres para la serie de secciones de línea recta que conectan los puntos de control en el orden específico son **polígono de control** y **polígono característico**. La figura 8-4 ilustra la forma de la gráfica de control para una secuencia de puntos de control dados en la figura anterior.

Condiciones de continuidad paramétrica

Una curva de spline la describimos a partir de varias secciones curvas más simples; ahora bien, si deseamos asegurar una transición suave de una sección a otra, podemos imponer varias **condiciones de continuidad** en los puntos de conexión. Si se describen dos secciones de spline sucesivas en la forma:

$$\begin{aligned} x_1 &= x_1(u), \quad y_1 = y_1(u), \quad z_1 = z_1(u); \quad u_1 \le u \le u_2 \\ x_2 &= x_2(u), \quad y_2 = y_2(u), \quad z_2 = z_2(u); \quad u_2 \le u \le u_3 \end{aligned} \tag{8-2}$$

Podemos establecer la **continuidad paramétrica** al comparar las derivadas de secciones curvas adyacentes en su frontera común. La **continuidad paramétrica de orden cero**, que se describe como continuidad C^0, implica que las secciones curvas sólo se unen. Es decir, los valores x, y y z que se evalúan en u_2 para la primera sección de la curva son

iguales, respectivamente, a los valores x, y y z que se evalúan en u_2 para la siguiente sección. La **continuidad paramétrica de primer orden**, que se denomina continuidad C^1, significa que las primeras derivadas paramétricas (líneas de tangente) de las funciones coordenadas en las ecuaciones 8-2, para dos secciones curvas sucesivas, son iguales en su punto de unión. La **continuidad paramétrica de segundo orden**, que se denomina continuidad C^2, implica que tanto como la primera como la segunda derivada paramétricas de las dos secciones curvas son las mismas en la intersección.

Ahora bien, **¿cómo podemos interpretar estas condiciones de continuidad paramétrica?** Con frecuencia, la continuidad de primer orden es suficiente para digitalizar trazos de curvas y superficies en aplicaciones de diseño, por ejemplo diseño automotriz. La continuidad de segundo orden es útil para establecer trayectorias de movimiento de objetos en una animación. Por ejemplo, una cámara que se desplaza a lo largo de la trayectoria de una curva, formada con secciones de spline, con continuidad de primer orden y con pasos iguales en el parámetro u, experimentaría un cambio brusco o aceleración no continua en la frontera de las dos secciones y provocaría discontinuidad en la secuencia del movimiento. No obstante, si la cámara se desplazara a lo largo de la trayectoria de una curva con continuidad C^2, la cámara experimentaría una transición suave en su movimiento a través de la frontera.

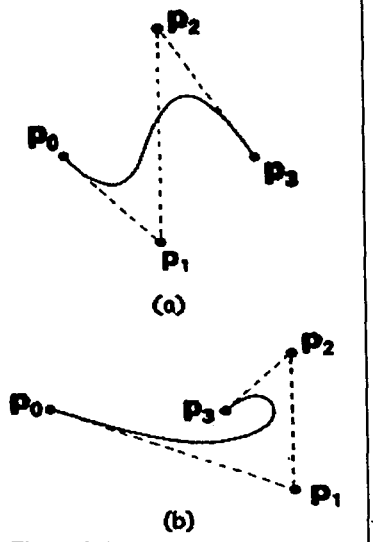


Figura 8-4
Formas de gráficas de control (líneas de rayas) para dos conjuntos diferentes de puntos de control. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 333.

Clasificación de curvas de spline

Splines
- Splines de interpolación
  - Splines cúbicas naturales
  - Splines de Hermite
  - Splines cardinales
  - Splines Catmull-Rom (o de Overhauser)
  - Splines Kochanek-Bartels
- Splines de aproximación
  - Splines Bezier
  - Curvas de B-Spline
    - Uniformes y periódicas
    - Uniformes y abiertas
    - No uniformes
    - Racionales
- etc.

Esta lista implica una clasificación muy general de las curvas de spline. Dado el carácter simplificado del presente trabajo escrito no es posible explicar cada caso, incluso se requiere de un libro completo para ello. Si se desea conocer algo más de este tipo de curvas, es recomendable acudir a Donald Hearn, M. Pauline Baker, "*Gráficas por Computadora*", Prentice Hall, 2ª edición, 1995, cap. 10. No obstante hago mención de dos ejemplos de curvas de spline: la interpolación de Hermite y la aproximación de Bezier cuyos conceptos son importantes en la implementación de curvas NURBS.

La implementación de este tipo de curvas, Hermite y Bezier es muy simple a comparación de otras curvas de spline. A consecuencia pasaremos casi de inmediato a la explicación de las características de una curva NURBS y como implementarla en OpenGL.

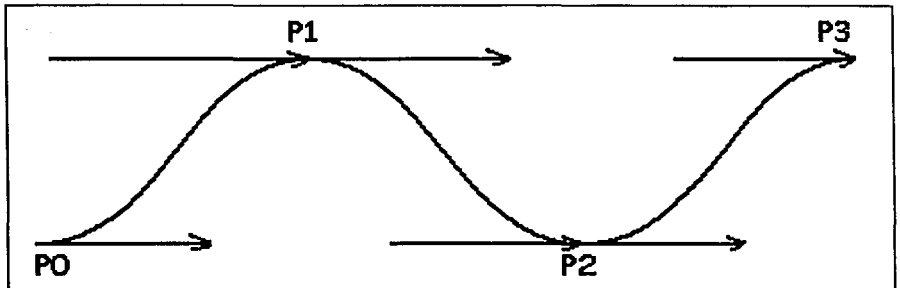


Figura 8-6
Curva de spline compuesta por secciones de Hermite. Cada sección de Hermite se especifica con dos puntos de control y pendientes de la curva en estos puntos. Para unir las secciones de Hermite se hacen coincidir los puntos de control y se igualan las pendientes para dar un aspecto suave a la curva.

8-1 Interpolación de Hermite

Figura 8-5
Curva de Hermite cuya forma se modifica ajustando los puntos de control.

Una **spline de Hermite**[1] es un **polinomio cúbico paramétrico** que genera una curva entre dos puntos que llamaremos de control. La ecuación correspondiente es:

$$P(u) = au^3 + bu^2 + cu + d; \quad 0 \le u \le 1 \qquad (8\text{-}3)$$

Donde el componente *x* de *P(u)* es $x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$ y de modo similar para los componentes *y* y *z*.

En cada uno de los puntos de control, el programador puede definir la respectiva pendiente de la curva, así que la forma que toma la spline sólo depende de las condiciones de frontera. La figura 8-5 muestra el concepto de las pendientes de la curva en los puntos de control. La figura 8-6 nos muestra varias secciones de Hermite unidas en sus puntos extremos para formar una sola curva. A consecuencia de la forma en que definimos nuestra sección de Hermite, se logra una continuidad de primer orden o C^1. Como es de esperar, un programador o usuario puede ajustar la forma de cada sección.

Las condiciones de frontera que nos interesan se expresan en las ecuaciones 8-4.

$$\begin{aligned} P(0) &= p_k \\ P(1) &= p_{k+1} \\ P(0) &= Dp_k \\ P(1) &= Dp_{k+1} \end{aligned} \qquad (8\text{-}4)$$

Donde Dp_k y Dp_{k+1} especifican los valores para las derivadas paramétricas (pendiente de la curva) en los puntos de control p_k y p_{k+1} en forma respectiva.

[1] Que se denomina así en honor del matemático francés Charles Hermite.

El equivalente matricial de la ecuación 8-3 es:

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{8-5}$$

y la derivada de la función 8-5 se puede expresar como

$$P'(u) = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{8-6}$$

Al sustituir los valores extremos de u en las dos ecuaciones anteriores, podemos expresar las condiciones de frontera de Hermite en forma matricial.

$$\begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{8-7}$$

Al solucionar la ecuación 8-7 para los coeficiente polinómicos, tenemos

$$\begin{aligned} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \times \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} \\ &= M_H \times \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} \end{aligned} \tag{8-8}$$

Donde M_H, es la matriz de Hermite y es el inverso de la matriz de restricciones de frontera. Así, es posible expresar la ecuación 8-5 en términos de las condiciones de frontera como

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \times M_H \times \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} \qquad (8\text{-}9)$$

Por último, podemos determinar las expresiones para formar **funciones de combinación de Hermite** al realizar las multiplicaciones matriciales de la ecuación 8-9 y recopilar los coeficientes para las restricciones de frontera a fin de obtener la forma polinómica.

$$P(u) = p_k H_0(u) + p_{k+1} H_1(u) + Dp_k H_2(u) + Dp_{k+1} H_3(u)$$

$$\begin{aligned} H_0(u) &= 2u^3 - 3u^2 + 1 \\ H_1(u) &= -2u^3 + 3u^2 \\ H_2(u) &= u^3 - 2u^2 + u \\ H_4(u) &= u^3 - u^2 \end{aligned} \qquad (8\text{-}10)$$

8-2 Aproximación de Bezier [2]

Las splines de Bezier tienen varias propiedades que hacen que sean muy convenientes para el diseño de curvas y superficies. Una curva de Bezier se aproxima a un conjunto de puntos de control, es decir, de todos los puntos de control, la curva sólo pasa por el primer y último puntos de control. Esto puede observarse en las curvas de Bezier de la figura 8-7

Suponga ahora que tenemos $n+1$ puntos de control, si un punto cualquiera en R^3 es $P_k=[X_k, Y_k, Z_k]$ con k en el rango de 0 a n, entonces la función polinómica de punto paramétrico es:

$$P(u) = \sum_{k=0}^{n} p_k BEZ_{n,k}(u), \quad u \in [0,1] \qquad (8\text{-}12)$$

Las funciones de combinación de Bezier, $BEZ_{n,k}(u)$ son los polinomios de Bernstein:

$$BEZ_{n,k}(u) = C_{n,k} u^k (1-u)^{n-k} \qquad (8\text{-}13)$$

[2] *Pierre Bezier*, ingeniero francés, desarrolló este método de aproximación de splines para utilizarlo en el diseño de carrocerías de automóviles *Renault*.

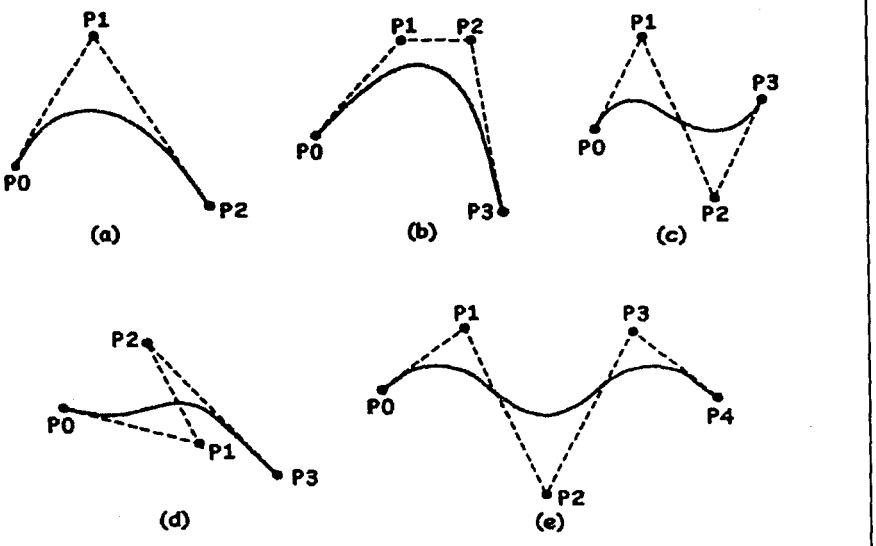


Figura 8-7
Ejemplos de curvas Bezier bidimensionales que se generaron a partir de tres, cuatro y cinco posiciones de puntos de control. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° edición en español, 1995. pg. 345.

Recuerde que la función de combinación $C_{n,k}$ se define como:

$$C_{n,k} = \frac{n!}{k!(n-k)!} \qquad (8\text{-}14)$$

La figura 8-7 también nos sirve para verificar la apariencia de algunas curvas de Bezier para varias selecciones de puntos de control en el plano *xy*.

Algunas propiedades correspondientes a las curvas de Bezier son:

- Una curva de Bezier es un polinomio de grado uno menos el número de puntos de control que se usan: un punto de control genera un punto, dos puntos de control generan una recta, tres puntos de control generan una parábola, etc.
- La curva de Bezier siempre pasa a través del primero y último puntos de control.
- Las funciones de combinación son todas positivas y su suma es siempre es uno

$$\sum_{k=0}^{n} p_k BEZ_{u,k}(u) = 1; \quad u \in [0,1] \qquad (8\text{-}15)$$

- Así, cualquier posición de un punto sobre la curva es la suma ponderada de las posiciones de los puntos de control.
- La tangente para la curva en un extremo está a lo largo de la línea que une ese extremo al punto de control adyacente.
- El número de funciones de combinación es igual al número de puntos de control.

- Una curva de Bezier no permite un control local sobre las secciones que la forman, es decir, si se modifica la posición de un punto de control, toda la curva cambia.

A modo de ejemplo, supongamos que tenemos 4 puntos de control, lo que da una curva de Bezier cúbica, es decir, de tercer grado. Las cuatro funciones de combinación son entonces:

$$
\begin{aligned}
BEZ_{3,0}(u) &= (1-u)^3 \\
BEZ_{3,1}(u) &= 3u(1-u)^2 \\
BEZ_{3,2}(u) &= 3u^2(1-u) \\
BEZ_{3,3}(u) &= u^3
\end{aligned}
\qquad (8\text{-}16)
$$

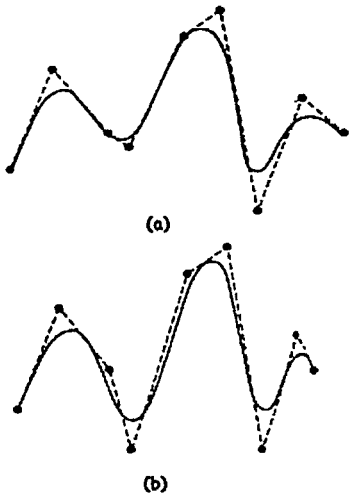


Figura 8-8
Modificación local de una curva B-spline. Al cambiar uno de los puntos de control de (a) se produce la curva en (b). Note que sólo se modifica la curva en las áreas cercanas al punto de control alterado. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2ª edición en español, 1995. pg. 352.

8-3 Curvas B-Spline

Esta es la clase de splines de aproximación que se utiliza con mayor frecuencia. Las curvas B-spline tienen dos ventajas sobre las curvas de Bezier:

- El grado del polinomio B-spline se puede establecer de manera independiente de la cantidad de puntos de control.
- Las curvas B-spline permiten un control local sobre la forma de curva.

Sin embargo, también hay desventajas, las curvas B-spline son sumamente complejas aún cuando se les compara con las curvas Bezier. Finalmente la figura 8-8 nos muestra una curva B-spline en la que uno de los puntos de control ha cambiado su posición. Note que la forma del resto de la curva no se altera.

Podemos escribir la expresión general para el cálculo de las posiciones de las coordenadas a lo largo de una curva B-spline como indica la ecuación 8-17.

$$
P(u) = \sum_{k=0}^{n} p_k B_{d,k}(u); \quad u_{\min} \le u \le u_{\max}; \quad 2 \le d \le n+1; \qquad (8\text{-}17)
$$

Ahora bien, los paquetes de diseño gráfico emplean un tipo especial de curva B-spline, cuya ecuación se muestra en la ecuación siguiente y que se llama B-spline racional o NURBS[3].

$$
P(u) = \frac{\sum w_k p_k B_{d,k}(u)}{\sum w_k B_{d,k}(u)}; \quad u_{\min} \le u \le u_{\max}; \quad 2 \le d \le n+1;
$$

La razón del empleo de este tipo de curvas implica que son invariables con respecto de las transformaciones geométricas, por ejemplo, al aplicar una transformación de perspectiva en los puntos de control de la curva racional, se obtiene la vista correcta, a diferencia de las B-splines que no son racionales: No nos avocaremos a verificar completamente este tipo de curvas ya que se requiere de mucha teoría matemática para la cual no tenemos

[3] NURBS es abreviación de Non Uniform Rational B-spline. Es español podemos nombrarla como B-spline racional no uniformes.

espacio, además ya se encuentran implementadas en OpenGL. Ahora veamos que significan las literales de la ecuación 8-17:

- p_k son un conjunto de $n+1$ puntos de control.
- $B_{d,k}$ son las $n+1$ funciones de combinación y la curva final se describe con estas combinaciones.
- d es el orden de las funciones de combinación y puede variar de 2 hasta $n+1$.
- $d-1$ es el grado de las funciones de combinación y varía de 1 hasta n. Esto quiere decir que si tengo una B-spline de grado 1, se trazará una recta que una los puntos de control. Si tengo una B-spline de grado 2 se trazará una curva, ya sea parábola, elipse, circunferencia, etc. Una B-spline de grado 3 se trazará como una curva cúbica y así sucesivamente.
- Observe la ecuación 8-17, el rango de u varía de un valor mínimo a un valor máximo.

Las funciones de combinación de las curvas B-spline son difíciles de formular, aún siguiendo las escasas instrucciones de los libros. Por esta razón nos limitaremos a mencionar algunas propiedades importantes y claro que esta información no es inútil. Las propiedades que anoto a continuación son importantes porque deben ser suministradas a OpenGL para que genere la curva deseada. Los siguientes puntos son importantes.

- El rango de u se divide en $n+d$ subintervalos.
- Cada función de combinación está definida para d subintervalos.
- La forma de la curva que generan las funciones de combinación se controla mediante un conjunto de valores que llamaremos **nudos**. Al conjunto de estos valores se les llama **vector de nudo** y debe contener $n+d+1$ valores. Más adelante veremos como se genera el vector de nudo y como afecta a la curva.
- Existe continuidad de orden C^{d-2}. Esto quiere decir que si deseo continuidad de segundo orden o C^2, debo definir una curva de cuarto orden, o bien, de tercer grado: curva cúbica.

Es seguro que al haber leído tantos detalles Ud. se haya perdido; pues deberá volver a leer porque la información proporcionada es importante y aún falta más: le advertí que las curvas B-spline eran complejas.

El vector de nudo

De párrafos anteriores sabemos que forma de un curva B-spline es controlada por un vector de nudos. En los segmentos siguientes veremos como generar el vector y la forma gráfica de las funciones de combinación que corresponden a estos vectores. Finalmente veremos como se implementan en OpenGL las curvas B-spline.

Existen tres tipos de vectores de nudo:

- Uniforme periódico
- Uniforme abierto
- No uniforme

El último tipo de vector de nudo es lo que da origen a los que comúnmente conocemos como curva NURBS o bien, B-spline no uniforme.

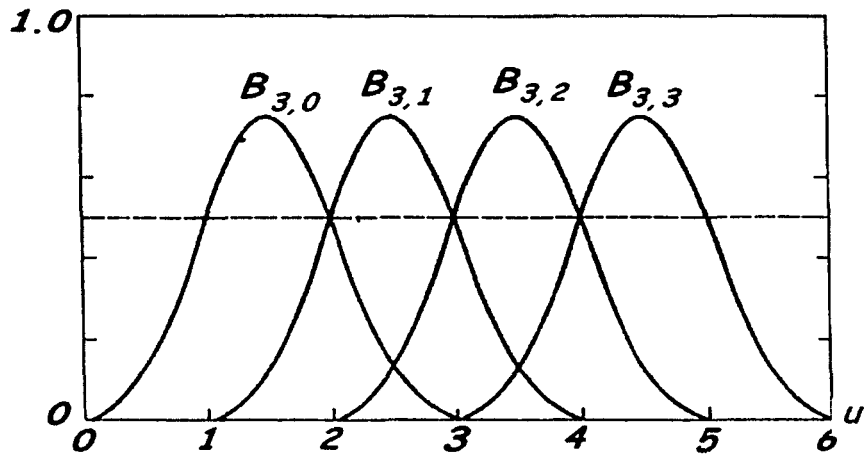


Figura 8-9
Comportamiento de las funciones de combinación para una curva B-spline periódica uniforme [X]=[0,1,,2,3,4,5,6], n+1=4,d=3. David F. Rogers, J. A. Adams, "*Mathematical Elements for Computer graphics*", McGraw Hill, 2° edición, pg. 310.

Vector de nudos uniforme y periódico

Los valores de nudo en este vector cumplen la siguiente relación

$$u_{i+1} - u_i = u_i - u_{i-1} \tag{8-18}$$

Esto es, que los valores de nudo son igualmente espaciados. Dos ejemplos son:

$$[0 \quad 2 \quad 3 \quad 4]$$
$$[-0.2 \quad -0.1 \quad 0 \quad 0.1 \quad 0.2]$$

En la práctica, los valores de nudo generalmente empiezan con cero y se incrementan en *1*. Algunos vectores de nudo son normalizados en el rango entre *0* y *1*, por ejemplo:

$$[0 \quad 0.25 \quad 0.50 \quad 0.75 \quad 1.00]$$

La figura 8-9 muestra el comportamiento de las funciones de combinación para cuatro puntos de control. Note que cada función es de tercer orden (grado 2), además en la misma figura notará el comportamiento periódico que tienen las mencionadas funciones, esto es:

$$B_{k,d}(t) = B_{k-1,d}(t-1) = B_{k+1,d}(t+1) \tag{8-19}$$

Debido a este comportamiento a las funciones de combinación se les califica de periódicas uniformes. Finalmente, note entre que rangos, la suma de las funciones de

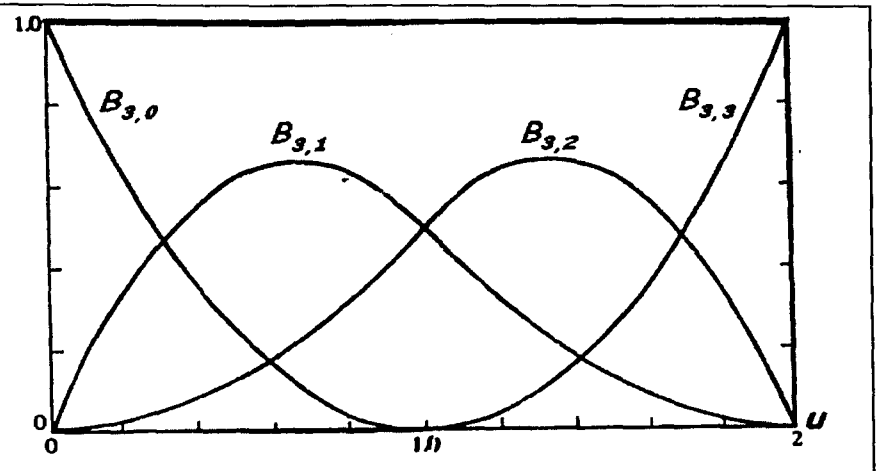


Figura 8-10
Funciones base para una curva B-spline uniforme y abierta. [X]=[0,0,0,1,2,2,2], n+1=4, d=3. David F. Rogers, J. A. Adams, "*Mathematical Elements for Computer Graphics*", McGraw Hill, 2° edición, 1990, pg. 311.

combinación vale *1*. Esto es $2 \le u \le 4$. Visto de otra forma, desde $d-1$ hasta $n+1$. ¿Qué quiere decir esto? Pues que debajo de *2* y encima de *4*, la función no está definida ¿y entonces? Entonces las posiciones inicial y final de la curva no coinciden con los puntos de control en los extremos de la curva.

Vector de nudos uniforme abierto

Este vector se caracteriza en que los valores de nudo en los extremos del vector tienen multiplicidad, es decir, los valores extremos se repiten *d* veces. Algunos ejemplos de vectores de nudo son los siguientes

$$\begin{aligned} k=2 \quad & [0 \;\; 0 \;\; 1 \;\; 2 \;\; 3 \;\; 4 \;\; 4] \\ k=3 \quad & [0 \;\; 0 \;\; 0 \;\; 1 \;\; 2 \;\; 3 \;\; 3 \;\; 3] \\ k=4 \quad & [0 \;\; 0 \;\; 0 \;\; 0 \;\; 1 \;\; 2 \;\; 2 \;\; 2 \;\; 2] \end{aligned}$$

Formalmente, un vector de nudo uniforme y abierto está dado por.

$$\begin{aligned} u_i &= 0 && 1 \le i \le k \\ u_i &= i-k && k+1 \le i \le n+1 \\ u_i &= n-k+2 && n+2 \le i \le n+k+1 \end{aligned} \qquad (8\text{-}20)$$

Cuando el vector de nudos tiene *d* ceros seguidos de *d* unos, la curva B-spline es una curva Bezier. Por ejemplo, para una curva con cuatro puntos de control, el polinomio que la representa es de cuarto orden ($d=4$ y grado $d-1=3$) y el vector uniforme es como sigue

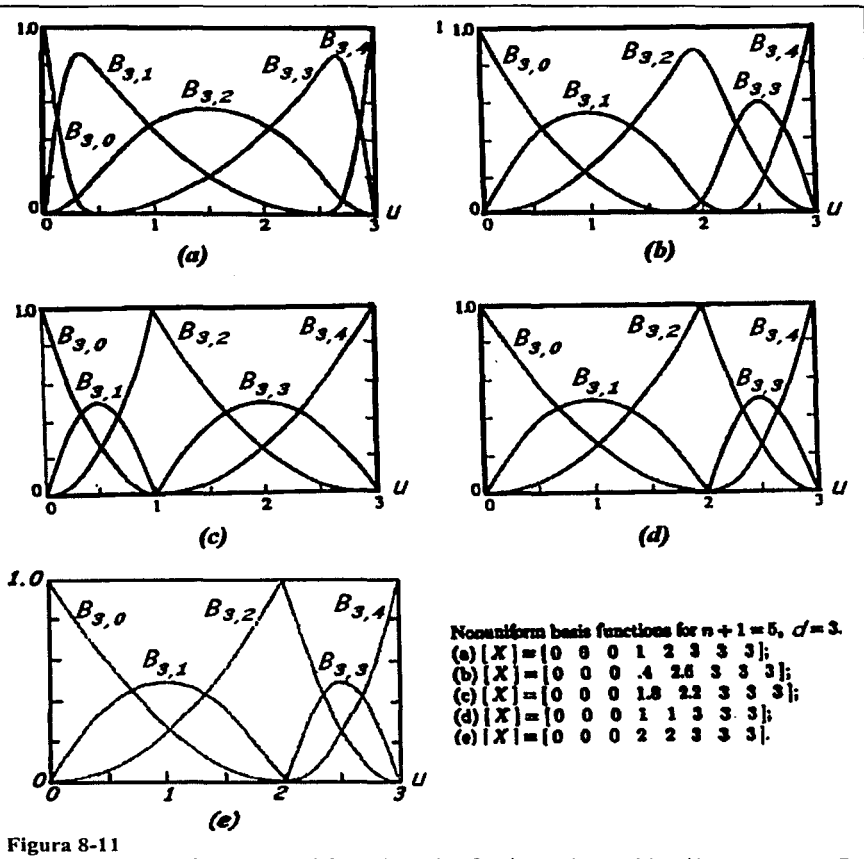


Figura 8-11
La figura muestra el comportamie3nto de varias funciones de combinación para curvas B-spline no uniformes. David F. Rogers, J. A. Adams, "*Mathematical Elements for Computer Graphics*", McGraw Hill, 2° edición, 1990, pg. 313.

$$[0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1]$$

La figura 8-10 (página anterior) muestra el comportamiento de cuatro funciones de combinación para una curva con 4 puntos de control y orden de $d=3$. Note que a diferencia de la curva B-spline uniforme periódica, la curva uniforme abierta si pasa por los puntos de control en los extremos de la curva.

Vector de nudos no uniforme

Los valores de nudo de este vector tienen sus valores desigualmente espaciados o bien, algunos valores tienen multiplicidad. Formalmente se cumple con la siguiente relación

$$u_{i+1} - u_i \neq u_i - u_{i-1} \tag{8-21}$$

La figura 8-11 muestra algunos ejemplos de funciones de combinación para diferentes vectores de nudo.

8-4 La implementación en OpenGL

NURBS es el acrónimo de *Non-Uniform Rational B-Spline*, o bien, traducido al buen español es B-spline racional no uniforme y es nuestro objetivo revisar los comandos que OpenGL nos proporciona.

Para especificar una curva NURBS, requerimos de un arreglo de puntos de control y de un vector de nudos. Ahora bien, antes de especificar una curva NURBS debemos indicarle a OpenGL que vamos a crear una curva NURBS. Para hacer esto simplemente invocamos la función

```
gluNewNurbsRender ();
```

Cada llamada a esta función retorna un puntero a un GLUnurbsObj, que es una clase (revise el apéndice A sobre programación) que se encarga de calcular los puntos en la curva. Si necesitamos otra curva NURBS, creamos otra clase GLUnurbsObj. De esta forma, nosotros podemos crear cualquier número de curvas y fijar las propiedades de cada una en forma independiente.

La forma de crear una clase del tipo GLUnurbsObj es:

```
nurbSurface = gluNewNurbsRender ();
```

Ahora bien, para fijar las propiedades de una curva NURBS, llamamos a la función

```
gluNurbsProperty ( GLUnurbsObj *nurb, GLenum property, GLfloat value);
```

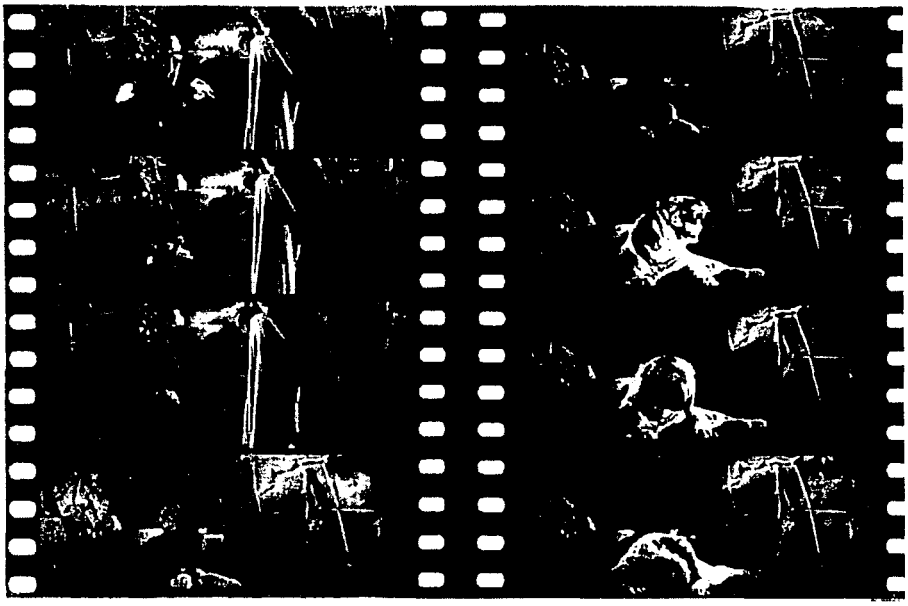
En donde:

- Nurbs representa una clase GLUnurbsObj.
- Property puede tener uno cualquiera de los siguientes significados.
    - GLU_SAMPLING_TOLERANCE: este valor especifica la longitud máxima en píxeles de los segmentos rectos que se emplean para representar la curva. Generalmente, OpenGL no traza cada punto de la curva calculándolo mediante funciones de combinación. En vez de ello calcula algunos puntos y luego realiza una interpolación lineal que requiere menos tiempo. El valor de longitud se indica en el parámetro *value*.
    - GLU_AUTOLOAD_MATRIZ: es un valor que puede ser verdadero (GL_TRUE) o falso (GL_FALSE). Un valor verdadero significa que la NURBS se trazará empleando las matrices de proyección y puerto de vista que viene por defecto en OpenGL. Un valor falso indica que el usuario tendrá que proporcionar estas matrices mediante la función gluLoadSamplingMatrices(). Los valores GL_TRUE y GL_FALSE se especifican en el parámetro *value*.

Ahora ya tenemos una curva NURBS bien definida, es decir, creamos la clase con la función gluNewNurbsRenderer() y establecimos sus propiedades con la función gluNurbsProperty(), lo que falta es implementar el código que la despliega en la pantalla.

```
GluNurbsCurve(  nobj      :   PGLUnurbs;
                Nknots :  TGLint;
                Knot      :   PGLfloat;
                Stride    :   TGLint;
                Ctlarray: PGLfloat;
                Order     :   TGLint;
                Atype     :   TGLEnum);
```

Todas las funciones que dibujan una NURBS deben se rodeadas por una pareja de funciones *gluBegin()* y *gluEndCurve()*. El único parámetro de estas funciones es el puntero a la NURBS que retornó la función *gluNewNurbsRenderer()*.



Capítulo 9 Deformación digital de imágenes y texturización

"...Tu tienes el poder, sólo necesitas confianza, vamos, concéntrate, dijo la hechicera. Entonces el pequeño mago apuntó la rama mágica hacia aquella que ahora tenía el aspecto de un avestruz. El aprendiz empeñó toda su energía y habilidad, la rama resplandeció con un brillo azul y la hechicera se transmutó en una tortuga, luego en tigre y finalmente en ella misma..."

El párrafo anterior narra un segmento del largometraje *Willow* (Estados Unidos) y de la cual algunas escenas están representadas en la figura que sirve de portada al presente capítulo. También representa un problema para los productores ¿cómo hacer que una cabra cambie su forma a un avestruz, a un tigre y a una mujer? Estará de acuerdo en que disolver una imagen sobre otra ya nadie se lo cree. Tampoco queremos abusar de la imaginación del espectador presentándole escenas de cada animal y esperando que entienda lo que ha sucedido.

Una solución la presenta la síntesis de imágenes. Veamos, primero necesitamos imágenes tridimensionales de cada animal; esto se logra modelando cada uno en arcilla y luego, con un escáner, se genera una descripción o modelo tridimensional de éstos. El siguiente paso implica a un programa que realice una transición suave entre las descripciones de los animales. Ahora bien, queda un detalle por resolver, el productor quiso que entre las transformaciones, los animales tuvieran movimientos naturales aunque también quería reducir costos.

Otra solución implica al procesamiento digital de imágenes, específicamente, técnicas de deformación geométrica, las cuales veremos a continuación. Ahora bien, dado que esta solución ha representado la mejor alternativa, este segmento tratará sobre transformaciones geométricas, mapeo de texturas, deformación y finalmente, veremos una variante de cómo se realiza el efecto deseado para *Willow* y que también servirá para dar una textura a nuestra bandera.

Como nota final a esta introducción, casi todo el trabajo se realiza considerando un sistema de coordenadas bidimensionales... alto, nosotros tratamos con objetos definidos en tres dimensiones, si nuestras texturas son de dos dimensiones ¿cómo podemos entonces aplicarlas a nuestros objetos definidos en la escena? La respuesta es simple, el usuario proporciona datos de tres dimensiones de los cuerpos en la escena, luego los mismos deben proyectarse a un plano, por lo que resulta una figura bidimensional. Es entonces el momento de aplicar las texturas.

9-1 Coordenadas homogéneas

Todos estamos acostumbrados a utilizar coordenadas cartesianas para representar los vértices que definen nuestra geometría. Es decir, un punto es algo así:

$$P = (x, y) \tag{9-1}$$

Y representa una determinada localización en un espacio 2D, pero cuando programamos gráficos hablamos de puntos y de vectores, los cuales pueden confundirse en cuanto a representación. Veamos un ejemplo, un vector se puede ver como una resta entre dos puntos.

$$Vector\ v \quad = \quad Punto\ 1 - Punto\ 2 \quad = \quad (x1, y1) - (x2, y2) = (a, b)$$

Y acaso (a,b) ¿no parece también un punto? Veamos como se resuelve esta situación mediante el empleo de coordenadas homogéneas.

Es muy sencillo convertir un vector o un punto cartesiano a su representación homogénea. De hecho, lo que se hace es añadir una nueva coordenada a las típicas (x,y). Añadimos entonces la componente (w) de esta forma:

- Un punto en coordenadas cartesianas es $p_1 = (x_1, y_1)$ y en coordenadas homogéneas es como $p_1 = (x_1, y_1, w_1)$.
- Un vector en coordenadas cartesianas es $\vec{v} = (a,b)$ y en coordenadas homogéneas es como $\vec{v} = (a,b,w)$.

Los valores típicos para la nueva componente son entonces:

- $w = 1$, cuando tratemos con puntos.
- $w = 0$, cuando sean vectores.

Por tanto el caso anterior queda modificado de la siguiente manera:

- Punto $p_1 = (x_1, y_1, 1)$ en homogéneas.

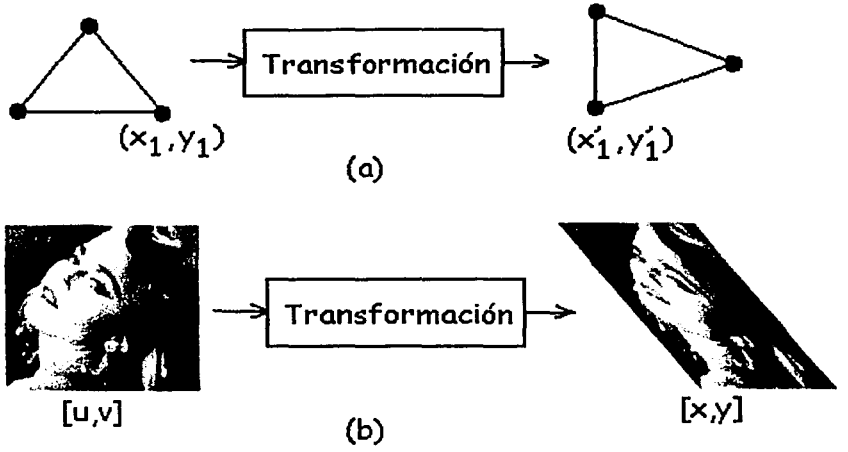


Figura 9-1
En (a) tenemos una transformación geométrica entre los vértices de entrada y salida. En (b) se muestra una transformación geométrica entre las imágenes de entrada y salida. Las imágenes se obtuvieron de George Wolber, *"Digital Image Warping"*, IEEE Computer Society Press Monograph, 1990, cap 7.

- Vector $\vec{v} = (a,b,0)$ en homogéneas.

Este convenio nos permite operar transformando un punto en otro punto y un vector en otro vector sin cruces extraños entre uno y otro.

Pero, y *¿si w es diferente de lo que he mencionado?* En ese caso tendremos que efectuar una sencilla operación para transformar un punto homogéneo en uno cartesiano. Veamos, si tenemos un punto en coordenadas homogéneas como:

$$p_1 = (x_1, y_1, w_1) \tag{9-2}$$

Entonces en coordenadas cartesianas el punto es:

$$p_1 = \left(\frac{x_1}{w_1}, \frac{y_1}{w_1}\right) \tag{9-3}$$

Es decir, normalizamos cada una de las componentes (x,y) del punto por su componente w. Claro que en el caso de $w = 1$ no hay que hacer nada pues la división es obvia pero puede pasar que nos interese variar w y entonces no podremos usar las (x,y) hasta haberlas normalizado según les acabo de explicar.

Según lo que ya hemos leído, debía quedar claro que:

$$p = (1,2,1) = (2,4,2) = (5,10,5)$$

9-2 Transformaciones espaciales, clasificación

Las transformaciones espaciales tienen un doble uso, sirven lo mismo para cuerpos geométricos definidos por vértices que para imágenes. Así es que antes de continuar debemos ponernos de acuerdo en algunas definiciones. Primero los polígonos.

Un vértice de cualquier polígono lo definimos como un punto mediante la pareja coordenada (x, y), lo que implica un espacio 2D. Luego de aplicar una operación de transformación a la pareja coordenada, el resultado es otra pareja coordenada (x', y'). La figura 9-1.a ilustra este concepto. Por cierto, al polígono que vamos a transformar lo llamamos **polígono de entrada** y al polígono que resulta lo llamamos **polígono de salida**.

Ahora es el turno de las imágenes. Para nuestro caso, en el que una imagen se almacena en la computadora, la misma se compone de decenas o cientos o incluso miles de muestras de la imagen original. Cuando una imagen se va a emplear como textura, a cada muestra la llamamos **elemento de textura** o simplemente **téxel** (téxel es un acrónimo adaptado al español de la frase anglosajona *texture element*). Cualquier téxel de una imagen, antes de su transformación (**imagen de entrada**) se puede referenciar con la pareja coordenada (u, v). Luego de la transformación obtenemos la **imagen de salida**, en la que cada muestra de la misma corresponde a un píxel. A cada píxel en la pantalla, como ya sabemos, se le hace referencia como (x, y). La figura 9-1.b ilustra este concepto.

Una transformación espacial define una relación geométrica ya sea entre los vértices de los polígonos de entrada y salida o entre los téxeles y píxeles de las imágenes de entrada y de salida. Dado que en este segmento nos interesa la transformación de imágenes, vamos a continuar con los conceptos relacionados a imágenes.

Las funciones que mapean los téxeles de la imagen de entrada a los píxeles de la imagen de salida las definimos como:

$$[x, y] = [X(u, v), Y(u, v)] \tag{9-4}$$

A su vez, las funciones inversas que mapean los píxeles de la imagen de salida a los téxeles de la imagen de entrada son:

$$[u, v] = [U(x, y), V(x, y)] \tag{9-5}$$

Donde X y Y son funciones que en conjunto mapean la imagen de entrada en la imagen de salida. Estas funciones representan un proceso que se conoce como **mapeo hacia delante**. U y V son funciones que realizan el proceso conocido como **mapeo inverso**, ya que mapean la imagen de salida en la imagen de entrada. El uso de paréntesis cuadrados es para facilitar la interpretación y no es parte de la nomenclatura propiamente.

Las transformaciones espaciales tienen su representación matemática y de acuerdo a ésta se clasifican como se muestra en la figura 9-2. Las transformaciones planares y las no planares generalmente se requieren para dar algunos efectos visuales, en tanto que las transformaciones polinomiales se requieren tanto para crear efectos visuales como para corregir distorsiones en la imagen. En resumen, las áreas de aplicación de las transformaciones son: el censado remoto, imágenes médicas, visión por computadora y síntesis de imágenes.

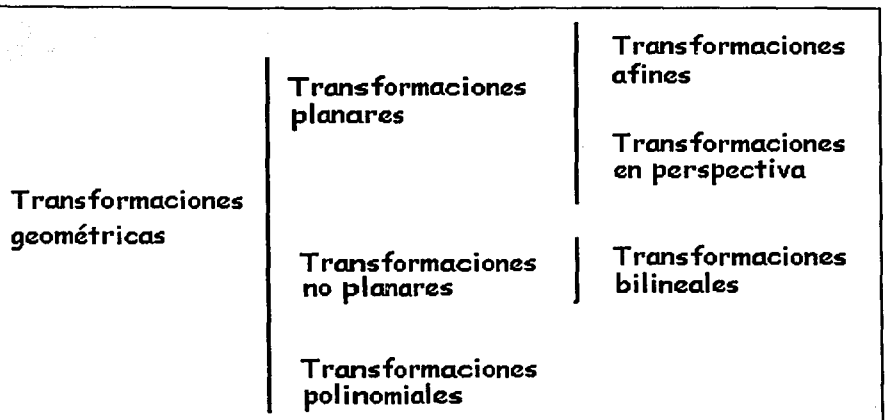


Figura 9-2
Clasificación de las transformaciones geométricas.

La matriz general de transformación planar

Algunas transformaciones espaciales simples pueden ser expresadas en términos de una matriz de transformación general de orden 3x3 y que para el caso 2D:

$$T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad (9\text{-}6)$$

y si deseamos transformar un punto, la operación a realizar es:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = T \times \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad (9\text{-}7)$$

9-3 Transformaciones afines

La representación general de una transformación afín para los vértices de un polígono es como sigue:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (9\text{-}8)$$

y para los elementos de una imagen:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad (9\text{-}9)$$

Note una primera propiedad, *el parámetro homogéneo se define como* $w = 1$. Ahora bien, ya que las transformaciones afines pueden cambiar la posición u orientación de un punto respecto de su sistema coordenado, tales transformaciones sirven igual para cuerpos geométricos definidos por vértices que para mapas de píxeles. A continuación veamos algunos ejemplos de las transformaciones que se pueden lograr con polígonos; recuerde que estas transformaciones también son válidas para imágenes:

Traslación: Es el movimiento en línea recta de un objeto, de una posición a otra. Se traslada un punto de una posición coordenada (x, y) a una posición (x', y') agregando distancias de traslación (t_x, t_y) a las coordenadas originales.

$$T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \qquad (9\text{-}10)$$

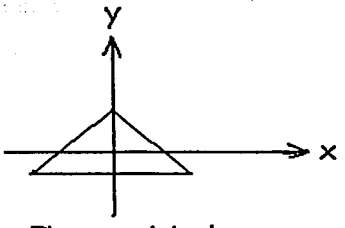


Figura original

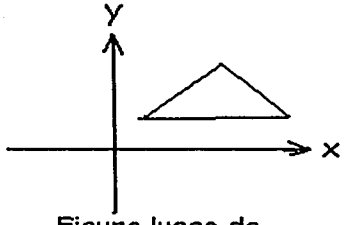


Figura luego de la traslación

Escalación. Es una transformación para alterar el tamaño de un objeto. Esta operación puede efectuarse con polígonos multiplicando los valores coordenados (x, y) de cada vértice en la frontera por los factores de escalación (s_x, s_y) para producir las coordenadas (x', y').

$$T(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (9\text{-}11)$$

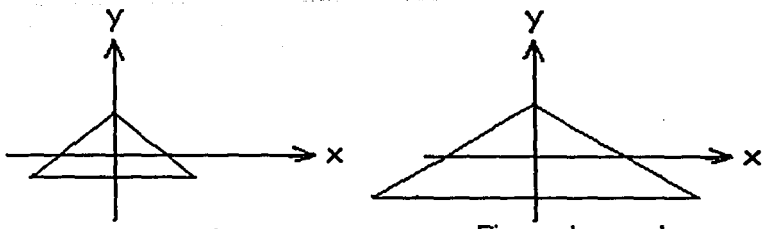


Figura original

Figura luego de la escalación

Rotación. La transformación de puntos de un objeto situados en trayectorias circulares se llama rotación. Este tipo de transformación se especifica con un ángulo de rotación, el cual determina la cantidad de rotación de cada vértice del polígono.

$$R(\theta)=\begin{bmatrix}\cos\theta & -sen\theta & 0\\ sen\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \qquad (9\text{-}12)$$

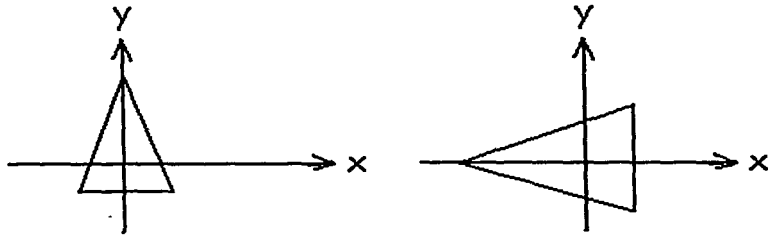


Figura original

Figura luego de la rotación

El ángulo de rotación se define como positivo si se supone un giro en la dirección contraria a las manecillas del reloj.

Corte. Una transformación que distorsiona la forma de un objeto de manera que parece como si el objeto estuviera compuesto por capas internas que se deslizaron una sobre otra. Otra forma de describir el corte es considerando la pareja coordenada (x, y), en donde una de las coordenadas varía proporcionalmente en función de la otra. Una de las posibles matrices de corte es la siguiente:

$$T\left((s_x, s_y\right)=\begin{bmatrix}1 & sh_x & 0\\ sh_y & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \qquad (9\text{-}13)$$

(a)

(b)

(c)

(d)

Figura 9-3
La figura exhibe los efectos de diversas transformaciones afines en la imagen de una reconocida artista. En (a) se exhibe la imagen de entrada en tanto que en (b) se muestra una transformación de corte. En (c) tenemos una rotación de 45 grados y en (d) se ha aplicado una escalación. Las imágenes se obtuvieron de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

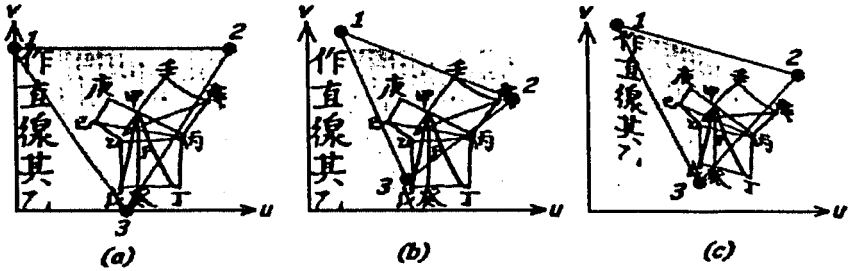
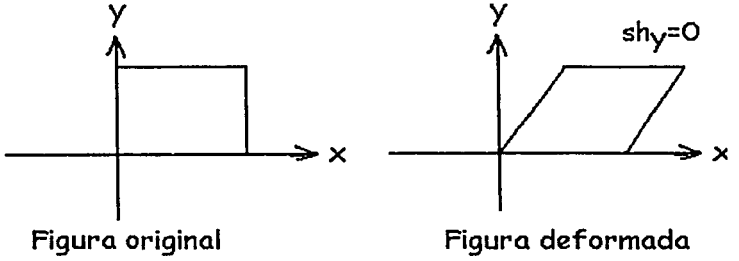


Figura 9-4
La figura ilustra los pasos para aplicar una transformación afín a una imagen mediante el uso de tres puntos de control. La figura se obtuvo de David *Bergamini*, "*Matemáticas*", Time-Life, 1981, pg. 78.



En *Hearn & Baker, Gráficas por computadora, 1990, Cap. 5* es posible encontrar mayor información sobre las transformaciones geométricas en los vértices de polígonos.

Ya hemos visto los efectos en figuras geométricas, pero que tal en mapas de píxeles. La figura 9-3 muestra los efectos de cada una de las transformaciones anteriores en la imagen de una artista. En 9-3.a tenemos la imagen original en tanto que las imágenes subsecuentes representan transformaciones de corte, rotación y escalación.

A continuación, presento un ejemplo de cómo se calculan los elementos de la matriz para una transformación afín cualquiera

Ejemplo

La figura 9-4.a nos muestra una imagen (una traducción al chino del Teorema de Pitágoras) a la que aplicaremos una transformación afín.

- Como primer paso a la transformación ubicamos tres puntos de control en la imagen de entrada: En la figura 9-4.a los puntos de control se indican en rojo.
- El segundo paso implica reubicar los puntos de control en posiciones diferentes para indicar que la imagen será transformada: en la figura 9-4.b las nuevas posiciones de los puntos de control se indican en azul.
- El último paso consiste en calcular los coeficientes de la matriz y aplicar ésta a cada píxel de la imagen: el resultado de la transformación puede verse en la figura 9-4.c.

Veamos ahora que ecuaciones resuelven nuestra transformación afín. Con base en la información que proporcionan tres puntos de control de entrada y tres puntos de control de salida, nosotros podemos formar dos sistemas de ecuaciones.

El primer sistema requiere de las abcisas de cada pareja coordenada correspondientes a los puntos de control, tanto de entrada como de salida y nos permite conocer los valores de las tres primeras constantes: *a, b* y *c*.

$$\begin{bmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad (9\text{-}14)$$

El segundo sistema requiere de las ordenadas de cada pareja coordenada correspondientes a los puntos de control tanto de entrada como de salida y nos permite conocer los valores de las últimas tres constantes: *d, e* y *f*.

$$\begin{bmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad (9\text{-}15)$$

Luego de calcular los coeficientes y aplicar la matriz de transformación afín a cada elemento de la imagen de entrada, debemos pasar de coordenadas homogéneas a coordenadas cartesianas. Por supuesto este paso es directo ya que $w = 1$.

Concluyendo

Una propiedad que caracteriza a las transformaciones afines implica esto: las transformaciones afines conservan paralelas las líneas y esto puede verse con claridad en la figura 9-5. En la figura 9-5.a se muestra un tablero de ajedrez y en las imágenes subsecuentes de la misma figura, que son transformaciones afines, puede verse el concepto de mantener las líneas paralelas.

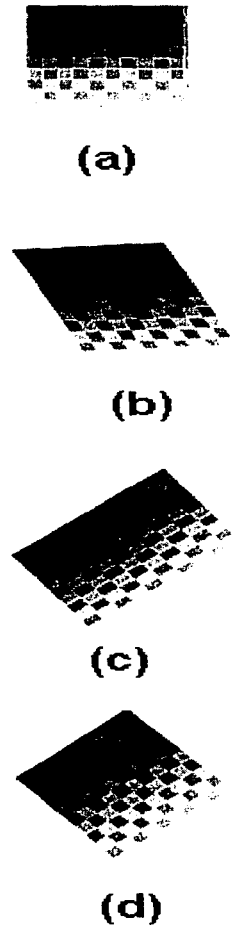


Figura 9-5
Patrones de tablero de ajedrez sometidos a diversas transformaciones afines. Puede notar que las transformaciones afines conservan las líneas rectas. Las imágenes se obtuvieron de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

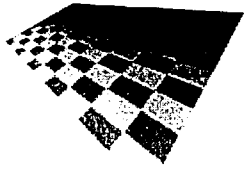
9-4 Transformación de perspectiva

La representación general de una transformación en perspectiva para los elementos de una imagen es como sigue:

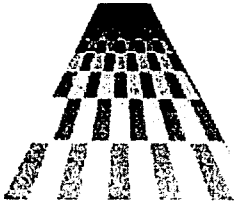
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad (9\text{-}16)$$

Ahora, desnormalizamos las coordenadas (x', y') de la ecuación (9-16) de tal forma que:

(a)



(b)



(c)

Figura 9-6
En (a) se muestra un patrón de tablero de ajedrez al que se le aplicará una transformación en perspectiva. Dos transformaciones en perspectivas para distintas posiciones del punto de fuga se muestran en (b) y (c). Las imágenes se obtuvieron de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

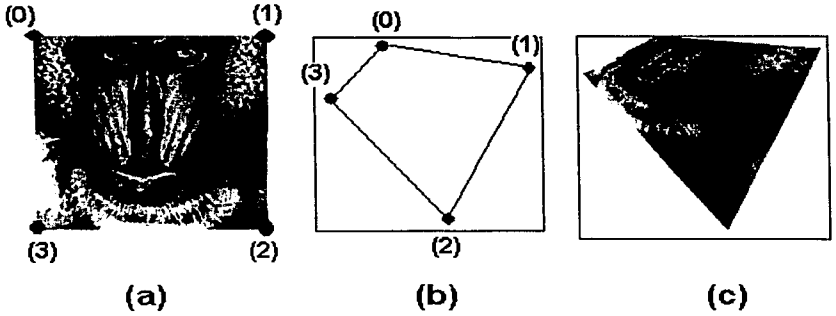


Figura 9-7
Pasos para aplicar una transformación en perspectiva. (a) Se seleccionan cuatro puntos de control. (b) Se modifica la posición de posición de los puntos de control en la imagen de salida y (c) Se aplica la transformación. Las imágenes se obtuvieron de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

$$x = \frac{x'}{w'} \qquad y \qquad y = \frac{y'}{w'} \tag{9-17}$$

Esta transformación corresponde a una perspectiva de un punto de fuga. A diferencia de las transformaciones afines puede notar que $w \neq 1$ y que la matriz de transformación tiene definidos todos sus elementos a_{ij}. La figura 9-6 muestra dos transformaciones en perspectiva de un patrón de tablero de ajedrez.

Revisemos entonces los pasos necesarios para realizar una transformación en perspectiva:

- El primer paso es seleccionar cuatro puntos en la imagen de entrada, según ejemplifica la figura 9-7.a, pero cuidado, existe la condición de que los puntos formen un polígono de cuatro lados que no se intercepte así mismo.
- El segundo paso es modificar las posiciones de estos puntos, lo que implica que la imagen de salida se verá distorsionada. Esto puede verlo en la figura 9-7.b.
- Finalmente, el último paso es calcular los coeficientes y aplicar el algoritmo. El resultado puede verlo en la figura 9-7.c.

Parece increíble que esto pueda lograrse con una matriz de 3x3, sin embargo así es. Ahora bien ¿cómo podemos determinar los coeficientes para una matriz de perspectiva? Una transformación en perspectiva es expresada en términos de nueve coeficientes en la matriz general de transformación planar. Ahora bien, no podemos perder generalidad si se normaliza esta matriz haciendo $a_{33} = 1$, lo cual deja ocho coeficientes que pueden ser determinados estableciendo la correspondencia entre los cuatro puntos de las imágenes de entrada y de salida, respectivamente.

Resolviendo la operación matricial de la ecuación (9-16) para cualquier (x, y) (note que ya desnormalizmos) y considerando que $a_{33} = 1$, obtenemos:

$$x = a_{11}u + a_{12}v + a_{13} - a_{31}ux - a_{32}vx$$
$$y = a_{21}u + a_{22}v + a_{23} - a_{31}uy - a_{32}vy \qquad (9\text{-}18)$$

En donde x y y representan las posiciones en la imagen de salida de los puntos de control. Lo que sigue ahora es sustituir los valores coordenados en las dos ecuaciones anteriores, lo que produce un sistema de 8 ecuaciones simultáneas con 8 incógnitas.

$$\begin{bmatrix} u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -v_0x_0 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -v_2x_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -v_3x_3 \\ 0 & 0 & 0 & u_0 & v_0 & 1 & -u_0y_0 & -v_0y_0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1y_1 & -v_1y_1 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2y_2 & -v_2y_2 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad (9\text{-}19)$$

Los coeficientes de la matriz de transformación son determinados resolviendo el sistema de ecuaciones. Es posible encontrar mayor información en *George Wolber (1990), "Digital Image Warping", Cap3*. Esta bibliografía proporciona un método para acelerar el cálculo de los coeficientes, lo que es recomendable.

9-5 Transformaciones bilineales

La representación general de las transformaciones bilineales es como sigue:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 \end{bmatrix} \begin{bmatrix} uv \\ u \\ v \\ 1 \end{bmatrix} \qquad (9\text{-}20)$$

La transformación bilineal, como en la transformación en perspectiva, también realiza un mapeo de un cuadrilateral a otro cuadrilateral. Las líneas que son horizontales o verticales, según la transformación, siguen siendo rectas y además tienden a ser equiespaciadas. Las líneas que son diagonales en la imagen de entrada tienden a deformarse en curvas cuadráticas. La figura 9-8 muestra el resultado de dos transformaciones bilineales aplicadas a un tablero de ajedrez.

La forma de determinar los coeficientes de la ecuación 9-20 es mediante conjuntos de ecuaciones simultaneas: con un conjunto determinamos los coeficientes a_k ; para $K = [0,1,2,3]$ y con el otro conjunto determinamos los coeficientes b_k ; para $K = [0,1,2,3]$.

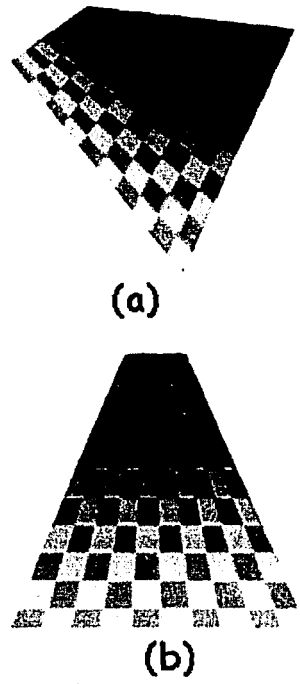


Figura 9-8
Recuerda el tablero de ajedrez de la figura 10-6, ahora los presente luego de habérseles aplicado una transformación bilineal. Note que las figura presentan ligeras distorsiones en el patrón cuadriculado. Las imágenes se obtuvieron de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

La forma de determinar los coeficientes a_k es como sigue: dados cuatro vértices (u_0,v_0), (u_1,v_1), (u_2,v_2), (u_3,v_3) y los respectivos valores de x_0, x_1, x_2 y x_3, cualquier coordenada $X(u,v)$ puede ser calculada por la expresión:

$$X(u,v) = a_0 + a_1 u + a_2 v + a_3 uv \qquad (9\text{-}21)$$

entonces los coeficientes a_k se obtienen resolviendo

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & v_0 & u_0 & u_0 v_0 \\ 1 & v_1 & u_1 & u_1 v_1 \\ 1 & v_2 & u_2 & u_2 v_2 \\ 1 & v_3 & u_3 & u_3 v_3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad (9\text{-}22)$$

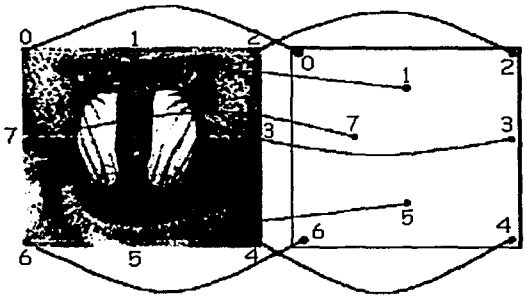
Los valores b_k se resuelven de forma semejante:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & v_0 & u_0 & u_0 v_0 \\ 1 & v_1 & u_1 & u_1 v_1 \\ 1 & v_2 & u_2 & u_2 v_2 \\ 1 & v_3 & u_3 & u_3 v_3 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad (9\text{-}23)$$

9-6 Transformaciones polinomiales

Las transformaciones polinomiales son muy requeridas en la corrección geométrica de imágenes que han sido afectadas por una función de distorsión desconocida. En *George Wolber (1990), "Digital Image Warping"*, Cap3, se pueden encontrar ecuaciones generales sobre las transformaciones polinomiales, nosotros emplearemos casos particulares que tienden a ser sencillos.

Los ejemplos más simples sobre transformaciones polinomiales ya los hemos visto, son las transformaciones en perspectiva y bilineales que nos permiten trabajar con cuatro puntos de control. La siguiente ecuación nos permite trabajar con seis puntos de control, pero ojo, las transformaciones polinomiales generalmente se trabajan con la técnica del mapeo inverso, esto es, una coordenada (x,y) de la imagen de salida se mapea a la imagen de entrada como (u,v), de la cual se toma un píxel de muestra que se coloca en la posición (x,y).



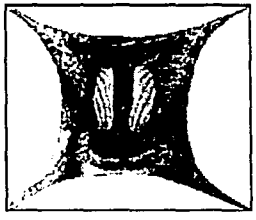
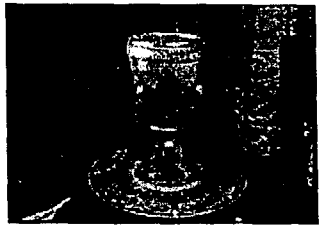


Figura 9-9
Transformación polinomial considerando 8 puntos de control. La figura se obtuvo de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{bmatrix} \qquad (9\text{-}24)$$

Nuevamente, los coeficientes a_k y b_k se resuelven de la misma forma que una transformación bilineal. La figura 9-9 muestra una transformación realizada considerando 8 puntos de control.



(a)

(b)

Figura 9-10
Mapeo de textura a una copa y a una jarra.

9-7 Mapeo de texturas

El mapeo de texturas es una técnica poderosa empleada para agregar detalles visuales a imágenes generadas en computadora, esto es, síntesis de imágenes. La figura 9-10 exhibe una muestra de lo que implica el mapeo de texturas.

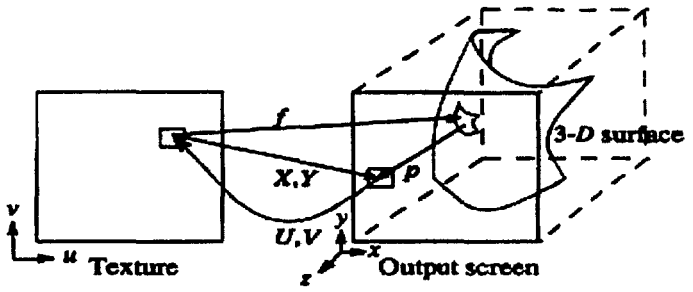


Figura 9-11
Funciones de transformación para mapeo de texturas. La figura se obtuvo de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

La actividad de mapear una textura consiste de una serie de transformaciones espaciales. Una textura plana en la que a un téxel genérico se le hace referencia como $[u,v]$, es transformada para adaptarla a una superficie 3D, en la que a cada muestra de la textura se le hace referencia como $[x,y,z]$, y entonces proyectada en una pantalla plana, en la que a los píxeles se les hace referencia como $[x,y]$. Esta secuencia es mostrada en la figura 10-11, donde f es la transformación desde $[u,v]$ a $[x,y,z]$ y p es la proyección de $[x,y,z]$ sobre $[x,y]$. Las funciones de mapeo hacia delante X y Y representan la función compuesta

$$p(f(u,v)) = [X(u,v), Y(u,v)] \qquad (9\text{-}25)$$

En la misma figura 9-11 se exhiben dos funciones U y V que representan un mapeo del plano *xy* hacia el plano *uv*. Estas dos funciones representan una técnica llamada mapeo hacia atrás. Así que tenemos dos técnicas para realizar el mapeo de una textura: directa y hacia atrás.

¿Cuál es la más técnica de mapeo más conveniente? Ahorrándonos explicaciones, la técnica del mapeo inverso es mucho más simple, rápida y además ha dado mejores resultados. En *Geroge Wolber (1990), "Digital Image Warping", cap3*, se proporciona una amplia información sobre diversas técnicas para el mapeo hacia delante.

Mapeo inverso

El mapeo inverso opera de la siguiente manera, una coordenada (x,y) de la imagen de salida se mapea a la imagen de entrada como (u,v), de la cual se toma un téxel que se coloca en la posición (x,y). La ecuación que realiza el mapeo de coordenadas es la siguiente:

$$p(u,v) = [U(x,y), V(x,y)] \qquad (9\text{-}26)$$

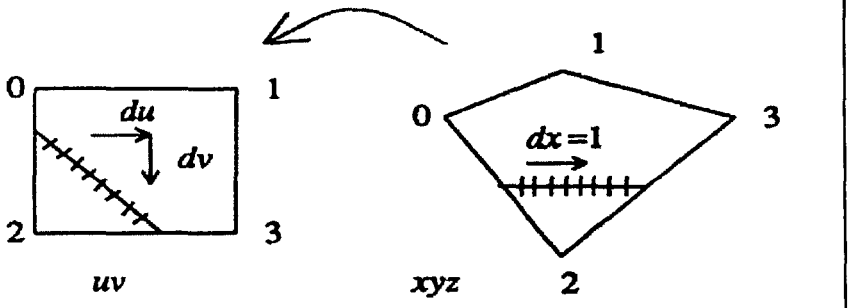


Figura 9-12
Mapeo inverso.

Fíjese bien, luego de mapear las coordenadas, un téxel es tomado de la imagen de entrada y su valor es copiado a la imagen de salida.

Ahora bien, cada posición de la imagen de salida se obtiene a partir de una línea de rastreo, típicamente horizontal. La figura 9-12 nos indica un mapeo entre la línea de rastreo horizontal en la imagen de salida y su equivalente en la imagen de entrada.

Las consecuencias del mapeo inverso son las siguientes:

- Una consecuencia inmediata del mapeo inverso implica que siempre habrá uno o varios valores de téxel para cada píxel de la imagen de salida, o sea, no habrá agujeros sin cubrir pero si podría suceder que se vean píxeles gigantes en la imagen de salida.
- Es posible que no todos los téxeles de entrada sean considerados para formar la imagen de salida, lo que resulta en pérdida de detalles.

Para solucionar las desventajas del mapeo inverso se recurre a técnicas de muestreo y filtrado que arrojan resultados muy aceptables y que veremos a continuación

Magnificación y minificación

Normalmente ocurre que al mapear una imagen de textura sobre una superficie, los téxeles no corresponden exactamente con los píxeles en la imagen de salida.

- Si la superficie es mayor que la textura, cada píxel se corresponderá con un trozo pequeño de téxel: magnificación (figura 9-13.b).
- Si la superficie es menor que textura, cada píxel se corresponderá con un conjunto de téxeles contiguos: minificación (figura 9-13.a).

Mipmaps

Mipmaps es un conjunto ordenado de arreglos representando la misma imagen de textura con resoluciones progresivamente menores. El primer mipmap representa a la imagen de textura original y tiene dimensiones de $2^n \times 2^m$, Cada mipmap subsiguiente tendrá

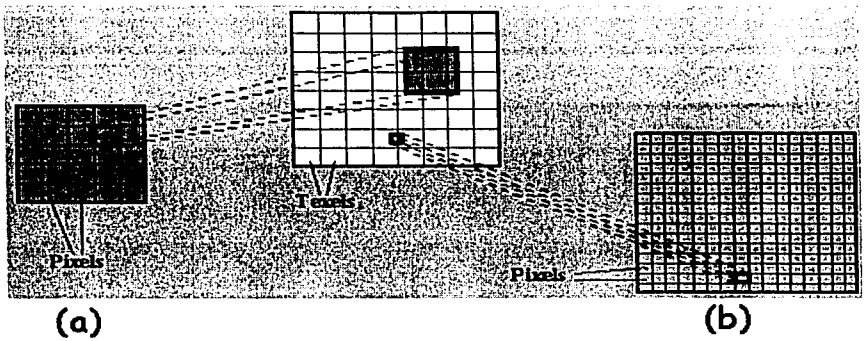


Figura 9-13
Minificación y magnificación. Figura obtenida de
http://serdis.dis.ulpgc.es/~atrujill/iga/Iluminacion/sld012.htm

entonces dimensiones de $2^{n-1} \times 2^{m-1}$. El mipmap más pequeño que puede existir es el que tiene las dimensiones de *1x1*. La figura 9-14 ilustra mejor este concepto.

Los mipmaps se emplean normalmente en la minificación y es para ajustar las dimensiones de la textura con la superficie. Esta técnica puede extenderse a la magnificación si se sobre-muestrea la imagen de textura original.

Seguramente se preguntará ¿cómo se obtienen los mipmaps? La respuesta, para este capítulo será simple, cada mipmap subsiguiente se genera obteniendo muestras de la imagen anterior, pero veamos como se hace: suponga la siguiente matriz:

$$\begin{bmatrix} 1635723454 \\ 3453556458 \\ 4537586734 \\ 3454865663 \\ 6354665456 \\ 7784574526 \end{bmatrix}$$

Si ahora submuestreamos la matriz, quedará entonces como

$$\begin{bmatrix} 13735 \\ 43563 \\ 65655 \end{bmatrix}$$

Si volvemos a submuestrear, la matriz quedará finalmente como

$$\begin{bmatrix} 175 \\ 665 \end{bmatrix}$$

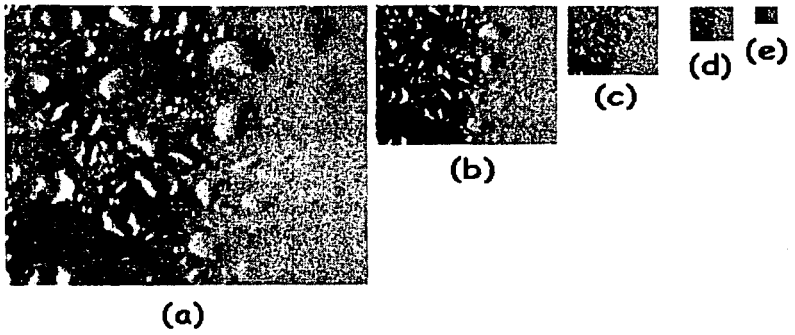


Figura 9-14
Conjunto de imágenes mipmap que se obtiene de la figura (a). Cada uno tiene dimensiones de la mitad del anterior.

Espectros de imágenes submuestreadas

Sistemas de graficación como OpenGL y DirectX obtienen sus mipmaps mediante la técnica de submuestreo, no obstante, aplican un paso adicional a la obtención de cada mipmap. Este paso adicional es un filtrado paso-bajas. Veamos entonces un esquema general del espectro espacial de la imagen de textura original en la figura 9-15.a. Si ahora obtenemos muestras de la imagen como ya sabemos, el espectro de la nueva imagen se verá como ilustra la figura 9-15.b. Observe, de esta última figura, que ocurre un solapamiento de los lóbulos espectrales por lo que se originan nuevas frecuencias. Una forma de evitar esta situación es aplicar un filtro paso-bajas, cuyo espectro puede verse en 9-15.c, en la imagen original. La figura 9-15.d muestra el espectro de la imagen original a la cual se le ha aplicado este filtrado. Si ahora submuestreamos, el espectro se verá como en la figura 9-15.e.

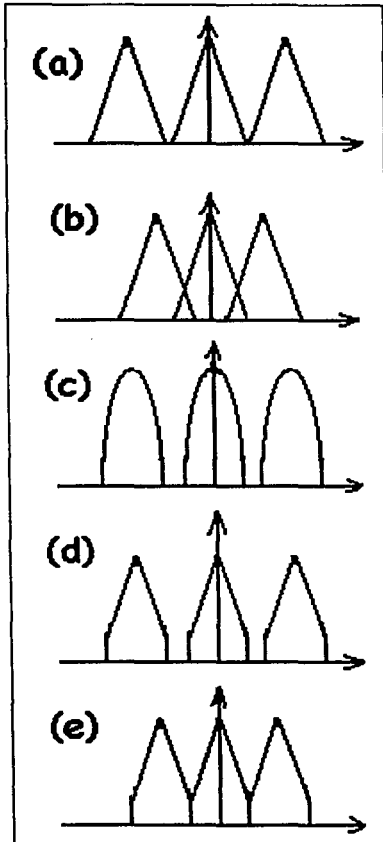


Figura 9-15

El filtrado

Veamos en que consiste la técnica del filtrado. Primero se obtiene el elemento de textura que está más cercano al píxel a texturizar. Luego, tomando ese elemento de textura y cuatro de sus vecinos, se promedian y el resultado corresponde al color que tendrá el píxel. La siguiente matriz ilustra como se consideran al elemento de textura y a sus vecinos.

$$\begin{bmatrix} 1 & 6 & 3 & 5 & 7 & 2 & 3 & 4 & 5 & 4 \\ 3 & 4 & 5 & 3 & 5 & 5 & 6 & 4 & 5 & 8 \\ 4 & 5 & 3 & 7 & 5 & 8 & 6 & 7 & 3 & 4 \\ 3 & 4 & 5 & 4 & 8 & 6 & 5 & 5 & 6 & 3 \\ 6 & 3 & 5 & 4 & 6 & 6 & 5 & 4 & 5 & 6 \\ 7 & 7 & 8 & 4 & 5 & 7 & 4 & 5 & 2 & 6 \end{bmatrix}$$

(a)

(b)

(c)

(d)

Figura 9-16
Secuencia de transformaciones para el largometraje *Willow*. La secuencia muestra la transformación avestruz a tortuga. La secuencia de imágenes se obtuvo de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

En esta matriz, el cuatro representa el téxel más cercano y sus cuatro vecinos quedan enmarcados en una cruz.

9-8 Malla de deformación en dos pasadas

Recuerda el problema que se mencionó al principio de este capítulo, la hechicera que fue transformada en cabra, avestruz, tortuga, tigre y mujer. Este problema nos abrió un mundo dentro del procesamiento de imágenes, el mundo de las deformaciones geométricas.

El algoritmo de la malla de deformación en dos pasadas fue desarrollado por Douglas Smythe, que en esos días trabajaba para *Industrial Light and Magic* (ILM). El algoritmo fue concebido originalmente para crear la secuencia de transformaciones cabra, avestruz, tortuga, tigre y finalmente en mujer. La figura 9-16 muestra la secuela de transformación avestruz a tortuga. En este contexto, **una transformación se refiere a una metamorfosis geométrica de una forma en otra.**

El algoritmo completo incluye dos técnicas, una es la deformación de imágenes, lo que implica dividir la imagen de entrada y la de salida en cuadrilaterales y alinearlos geométricamente. La otra técnica es llamada *cross-disolve*, que es una operación que transforma una imagen en otra mediante la interpolación píxel a píxel. El resultado de aplicar esos dos algoritmos es una secuencia de imágenes que dan la impresión de que una forma cualquiera se transforma en otra.

Descripción el algoritmo

En lo que respecta al presente trabajo escrito, nos dedicaremos al concepto de la malla de deformación, en cuanto a la técnica de *cross-disolve*, la dejaremos de lado aunque bien no puede ser difícil de imaginar como trabaja o también, es posible consultarla en algún libro sobre procesamiento digital de imágenes.

El algoritmo que nos ocupa tiene la característica de actuar en dos pasadas para transformar una imagen de entrada en la imagen de salida. Ambas pasadas son idénticas exceptuando por el hecho de que la primer pasada se aplica sobre los renglones de la imagen de entrada generando así una imagen intermedia. La segunda pasada actúa sobre las columnas de la imagen intermedia.

Vamos a requerir una nomenclatura para designar a las imágenes, veamos: a nuestra imagen de entrada la designaremos como S, la imagen intermedia como I y la de salida como D.

La figura 9-17.a muestra la imagen que deseamos afectar: es un tablero de ajedrez coloreado. La figura 9-17.b muestra un bosquejo de cómo debe quedar la imagen de salida. Se requieren entonces **dos arreglos bidimensionales de puntos de control**, uno de ellos es marcado en la imagen de entrada y el otro en la imagen de salida. Como ayuda, la figura 9-18 muestra la ubicación de los puntos de control y la tendencia de su movimiento entre las imágenes S y D.

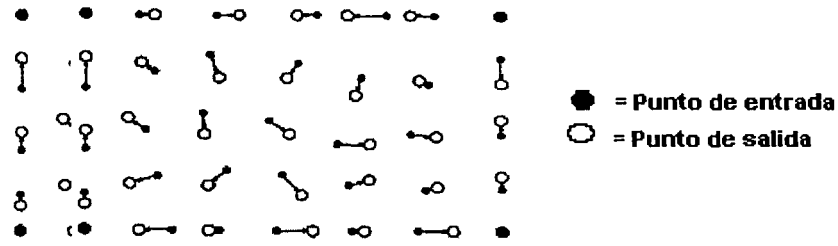


Figura 9-18
Ubicación de los puntos de control para las imágenes de entrada y de salida. La figura muestra el movimiento que tendrán que realizar los puntos de control de entrada para deformar la imagen. La figura se obtuvo de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

La primer pasada

Los puntos de control de las imágenes S y D son ajustados con splines verticales que interpolan posiciones entre ellos. La figura 9-19.a muestra la situación. Estará de acuerdo en que las líneas de rastreo horizontales se han dividido en segmentos. La primer pasada ajustará cada segmento de línea de rastreo a su nueva longitud, ya sea que debe alargarse (magnificación) o acortarse (minificación). La figura 9-19.b muestra el resultado intermedio.

La segunda pasada

Esta es idéntica a la primer pasada: Los puntos de control e las imágenes *I* y *D* deben ajustarse con splines interpolantes horizontales. Esto deja a las líneas de rastreo verticales en segmentos. La segunda pasada ajustará cada segmento de línea de rastreo a una nueva longitud, ya sea que debe alargarse (magnificación) o acortarse (minificación). La figura 9-20 muestra el resultado final.

9-9 Concluyendo

Todos los conceptos que han sido mencionados en este capítulo: deformación geométrica y mapeo de texturas, son parte integral de cualquier software que se emplee para la síntesis de imágenes. En este capítulo he introducido las generalidades que podrían interesar a cualquier entusiasta que se esté iniciando, no obstante, gente más avanzada quedaría decepcionada ya que hay conceptos que no se han incluido:

- Otras técnicas de texturizado como son la esférica, la cilíndrica, etc.
- Algoritmos de dos pasadas.
- Nociones de filtrado empleados en la magnificación y minificación de imágenes.

Abarcar estos conceptos requeriría de un libro completo o bien de otra Tesis.

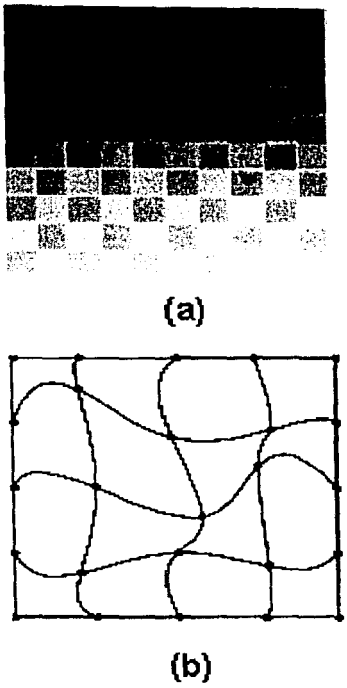


Figura 9-17
En (a) se muestra la imagen de entrada: un tablero de ajedrez coloreado. En (b) se muestra un bosquejo de cómo debiera verse la imagen final. Las líneas se han creado con splines de interpolación. Las figuras se obtuvieron de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

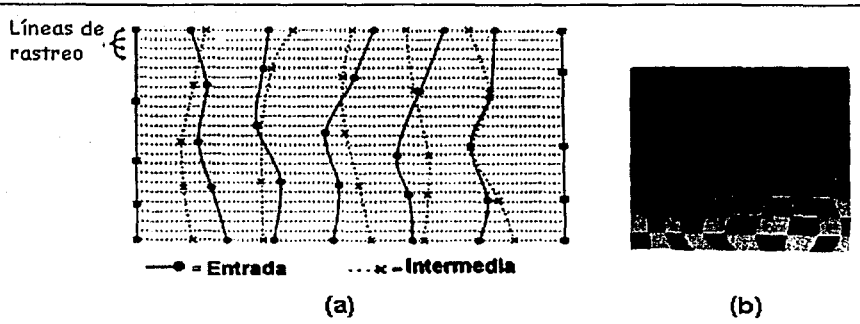


Figura 9-19
En (a) se muestran las splines que ajustan los puntos de control verticalmente. Las splines sirven para segmentar las líneas de rastreo horizontales. La primer pasada ajusta entonces todos los segmentos de líneas de rastreo a sus nuevas longitudes mediante la magnificación o la minificación de los mismos: esto es un problema del procesamiento de imágenes. En (b) se muestra la imagen intermedia. Las figuras se obtuvieron de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

El algoritmo de deformación de dos pasadas es importante ya que tiende a trabajar con elementos unidimensionales en cada pasada. Esta situación toma ventaja de la implementación de filtros digitales implementados en los procesadores, ya sea el procesador principal del sistema o el que se encuentra en el subsistema de video o bien, otro instalado en la placa de la computadora como es el caso de Silicon Graphics (esta computadora es toda tarjeta de video).

Figura 9-20
La figura muestra el resultado de aplicar las dos pasadas a la imagen del tablero de ajedrez. Puede entonces notar el éxito y la eficiencia del algoritmo. La figura se obtuvo de George Wolber, "*Digital Image Warping*", IEEE Computer Society Press Monography, 1990, Cap. 7.

Cortesía de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*"

Capítulo 10 Modelos de color

Un modelo de color es un método para explicar las propiedades o el comportamiento del color en algún contexto en particular. Ningún modelo puede explicar todos los aspectos, de modo que utilizamos diferentes métodos cómo ayuda para describir las diversas características del color que se percibe.

No es necesario desarrollar un tema completo sobre óptica, sin embargo, la curiosidad y la ambigüedad de términos entre distintas fuentes bibliográficas nos conducen a establecer una convención de términos propia.

El Sol es una fuente de cantidades gigantescas de radiación. La radiación también la emiten lámparas incandescentes, las lámparas fluorescentes y las flamas. Algo de esta radiación puede estimular los receptores sobre la retina en el interior de nuestros ojos. A esta radiación la llamamos **luz**. Mucha de la radiación que proviene de estas fuentes, el ojo no la percibe y además tiene otros nombres, como ondas infrarrojas, ondas ultravioleta y ondas de radio. Estas ondas pueden percibirse por otros medios.

La figura 10-1 es una carta del espectro electromagnético y nos permite ubicar nuestra capacidad de visión. El rango de frecuencias que captamos lo llamamos la **banda de frecuencias visible**.

Ya que la luz es una onda electromagnética, podemos describir los diversos colores ya sea en términos de la frecuencia f o de la longitud de onda λ. Así, el color rojo se describe como una onda de 630 nm o 4.7×10^{14} hertz.

Cuando un rayo de luz existe solo a una frecuencia, a este mismo lo podemos llamar **rayo de luz monocromática**. A este respecto, la longitud de onda y la frecuencia de la onda

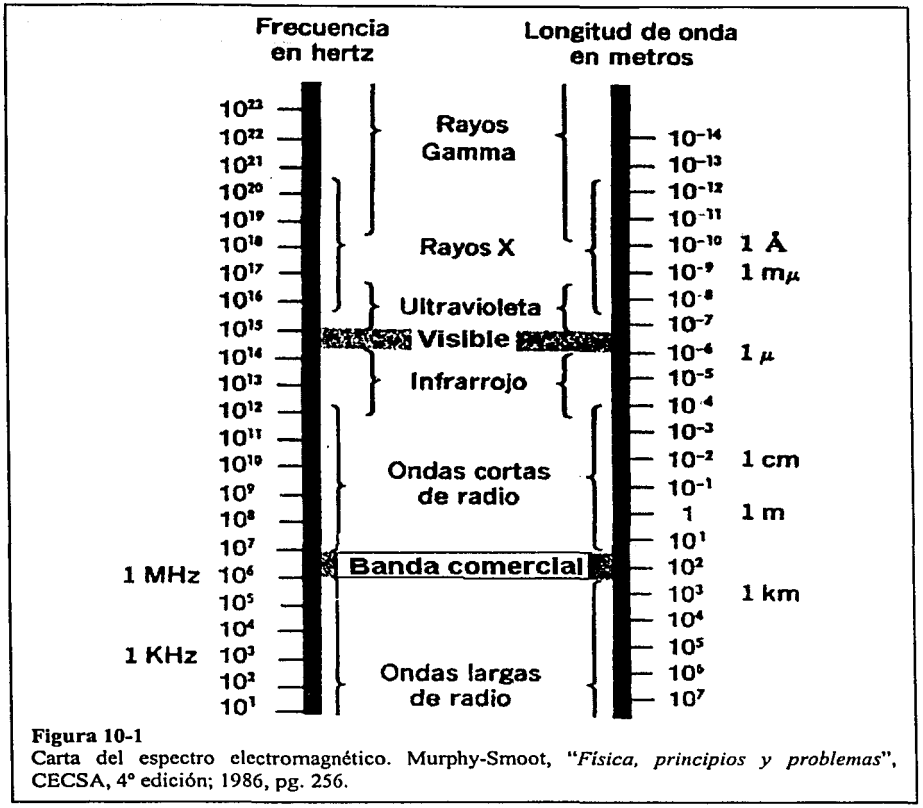


Figura 10-1
Carta del espectro electromagnético. Murphy-Smoot, "*Física, principios y problemas*", CECSA, 4° edición; 1986, pg. 256.

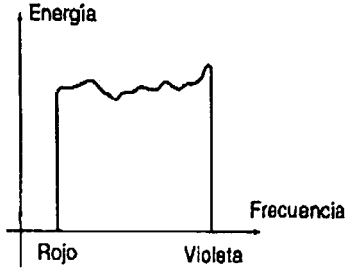


Figura 10-2
Distribución de energía para una fuente de luz blanca. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 597.

monocromática son inversamente proporcionales. La siguiente fórmula relaciona la longitud de onda, la velocidad de la luz y la frecuencia de la misma:

$$\lambda = c\frac{1}{f} \qquad (10\text{-}1)$$

Una fuente de luz como el Sol o un foco emite todas las frecuencias en el rango visible para producir luz blanca. La figura 10-2 exhibe la distribución de energía de la luz blanca emitida por una de estas fuentes.

Cuando la luz blanca llega a un objeto, éste refleja algunas frecuencias y absorbe otras. La combinación de frecuencias propias de la luz reflejada determina lo que percibimos como el color de un objeto. Si en la luz que se refleja predominan las frecuencias bajas, se describe el objeto cómo rojo. En este caso se dice que la luz percibida tiene una **frecuencia dominante** o **longitud de onda dominante** en el extremo rojo del espectro. La figura 10-3 muestra una distribución de energía de una fuente de luz con una frecuencia dominante cercana al extremo rojo de la banda visible de frecuencias. También se conoce a la frecuencia dominante como **matiz**, o sólo como el **color** de la luz.

En un tratamiento electromagnético moderno, una onda electromagnética transporta energía y este transporte se considera en la unidad de tiempo, lo que corresponde a la definición de potencia. A su vez, a esta definición de potencia, a frecuencias ópticas no es posible medirla, por lo que se considera una **potencia promedio**.

Dentro del mismo tratamiento electromagnético, la potencia promedio no es muy empleada para estudios de reflexión. En su lugar, el planteamiento de los cálculos se realiza considerando la **densidad de potencia** o **irradiancia**, esto es, el transporte promedio de energía en la unidad de tiempo a través de un área unitaria normal al flujo de energía. Este parámetro también es conocido como la **iluminación**.

Una característica que podemos percibir es la **pureza** o **saturación** de la luz. La pureza describe cuan deslavado, pálido o puro es el color de la luz a la vista. Los colores pastel y pálidos se describen como menos puros.

A la densidad de potencia se le suele llamar en forma empírica como brillantez. A la frecuencia dominante y a la pureza se les conoce como valores de cromaticidad. Los conceptos de frecuencia dominante, brillantez y pureza son conceptos que empleamos para describir algunas propiedades que percibimos en una fuente de luz.

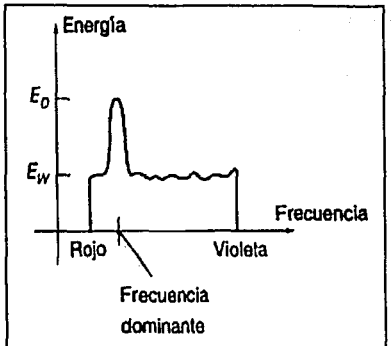


Figura 10-3
Distribución de energía de una fuete de luz con una frecuencia dominante cercana al extremo rojo del rango de frecuencia. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 597.

10-1 Conceptos intuitivos de color

Un artista crea una imagen al mezclar pigmentos de color con pigmentos blancos y negros. Con estas combinaciones forma las distintas sombras, tintes y tonos en la escena. Al iniciar con un pigmento para un color puro, el artista agrega un pigmento negro para producir distintas sombras de ese color. Cuanto mayor es la cantidad de pigmento negro que se agrega, más oscura es la sombra. De modo similar, se obtienen diversos tintes del color al agregar pigmentos blancos al color original, haciéndolo más claro conforme se agrega más blanco. Se producen tonos de color al agregar pigmentos tanto negros como blancos.

10-2 El modelo de color XYZ

En 1931, la Comisión Internacional sobre Iluminación, CIE[1], definió tres colores primarios estándar, los cuales tienen el carácter de imaginarios y se definen de manera matemática con funciones que nos permite definir cualquier color únicamente sumando estas funciones.

Por lo general, al conjunto de colores primarios de la CIE se le conoce como **modelo de color XYZ**. Este modelo de color se define en un sistema rectangular. Así que cualquier color C_λ puede expresar como:

$$C_\lambda = X\bar{x} + Y\bar{y} + Z\bar{z} \qquad (10\text{-}2)$$

Donde X, Y, Z definen las cantidades de colores primarios estándar que se requieren para igualar C_λ. Al analizar las propiedades del color es conveniente normalizar las cantidades de la ecuación 10-2 contra la suma de las tres cantidades (a la suma

[1] CIE son las siglas de Comision Internationale de l'Eclairage

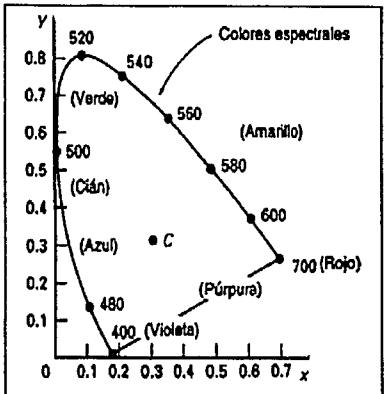


Figura 10-4
Diagrama de cromaticidad de la CIE. Las posiciones de color espectral a lo largo de la curva se definen en unidades de longitud de onda (nm). Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 600.

$X+Y+X$ se le suele llamar la **luminancia**). De esta manera se calculan las cantidades normalizadas como:

$$x = \frac{X}{X+Y+Z}$$
$$y = \frac{Y}{X+Y+Z} \qquad (10\text{-}3)$$
$$z = \frac{Z}{X+Y+Z}$$

Observe que $x+y+z=1$ dado que se trata de cantidades normalizadas.

Según la CIE, es posible representar cualquier color con sólo los valores de x y y. Entonces a estos valores los conoceremos mejor como los **valores de cromaticidad**. Ahora bien, cuando se trazan las cantidades normalizadas de x y y para colores en el espectro visible obtenemos la curva de la figura 10-4. Esta curva junto con los valores interiores se conoce como el **diagrama de cromaticidad de la CIE**. Los puntos a lo largo de la curva de cromaticidad son los colores puros en el espectro electromagnético, los cuales se designan junto con la longitud de onda en nanómetros. La línea que une los puntos espectrales rojo y violeta, la cual se denomina **línea púrpura**, no forma parte del espectro. Los puntos interiores representan todas las combinaciones de color visible posibles. El punto C en el diagrama corresponde a la posición de la luz blanca. La figura 10-5 (al final del presente capítulo) es una representación a color del diagrama de cromaticidad, por supuesto, es una impresión en papel por lo que no puedo garantizar la fidelidad de la imagen.

Seguramente notará también que este diagrama muestra la composición de cualquier color como una función de x (rojo) y y (verde). Para cada valor de x y y, el correspondiente valor de z (azul) se obtiene como:

$$z = 1-(x+y) \qquad (10\text{-}4)$$

Por ejemplo, el punto marcado verde en la figura 10-5, tiene aproximadamente un 62 por 100 de verde y un 25 por 100 de contenido rojo. Entonces la composición de azul es de aproximadamente el 13 por 100.

Una de las utilidades del diagrama de cromaticidad implica calcular la longitud de onda y pureza dominantes de un color específico.

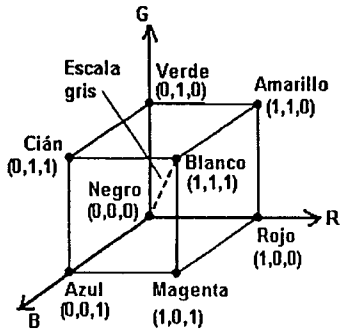


Figura 10-6
Modelo de color RGB. Se trata de un cubo unitario en el cual los colores se definen en un proceso aditivo. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 600.

10-3 Modelo de color RGB

Con base en la teoría de los tres estímulos de la visión, nuestros ojos perciben los colores mediante el estímulo de tres pigmentos visuales en los conos de la retina. Estos pigmentos visuales tienen una sensibilidad pico en longitudes de onda de alrededor de 630 nm (rojo), 250 nm (verde) y 450 nm (azul). Esta teoría de la visión es la base para desplegar la salida a color en un monitor de video al utilizar los tres colores primarios, rojo, verde y azul. A la teoría se le conoce entonces como el modelo de color RGB[2].

[2] RGB es un acrónimo para referirse a los términos anglosajones Red, Green, Blue.

Podemos presentar este modelo en un sistema de referencia rectangular en tres dimensiones y a partir de un cubo unitario según se muestra en la figura 10-6. Los tres ejes de referencia corresponden a los tres colores primarios.

- El origen representa al negro.
- El vértice con las coordenadas $(1,1,1)$ representa el blanco.
- Los vértices del cubo en los ejes coordenados representan los colores complementarios para cada pareja de colores primarios. Un color complementario se genera al sumar dos de los tres colores primarios en los vértices.
- Al diagonal que va del origen negro al vértice blanco se le conoce como diagonal de la escala de gris de negro a blanco.

El esquema de color RGB es un modelo aditivo, es decir, se suman las intensidades de los colores primarios para producir otros colores. Cada punto de color dentro de las fronteras del cubo se puede representar como el conjunto de tres coordenadas (R,G,B), cada dimensión puede tomar valores entre 0 y 1. Así un color C_λ se expresa en componentes RGB como:

$$C_\lambda = R\bar{r} + G\bar{g} + B\bar{b} \qquad (10\text{-}5)$$

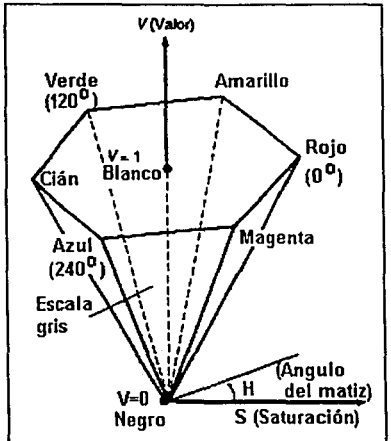


Figura 10-7
Hexacono HSV. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 607.

10-4 Modelo de color HSV

En vez de un conjunto de colores primarios, el modelo HSV utiliza descripciones de color que tienen una aplicación más intuitiva por parte del usuario. Así, para dar una especificación de color, el usuario selecciona un color espectral puro o matiz, la pureza y el valor del mismo.

La figura 10-7 nos muestra el modelo HSV. Tal vez no se note, sin embargo, conceda que se trata de un hexacono definido en un sistema de coordenadas cilíndricas. El **matiz (*H*)** corresponde a un color puro y se especifica como un ángulo alrededor del eje del hexacono. En este modelo H=0° corresponde al rojo. Los colores restantes se especifican alrededor del perímetro. Por ejemplo, el verde se encuentra a 120° y el azul está a 240°. Los colores complementarios amarillo, cián y magenta tienen una separación de 120°.

La **saturación (*S*)** varía de 0 a 1 y se representa en este modelo como la razón de pureza del matiz. Un matiz puro implica $S = 1.0$ o sea, saturación al 100%. Una pureza de un cuarto se tiene con el valor de $S = 0.25$.

El eje vertical en este modelo se llama **valor (*V*)** y representa la escala de grises de negro a blanco. Tenemos entonces negro cuando $V = 0$ y tenemos blanco cuando $V = 1$ y $S = 0$. Cuando $S = 0$ y $0 \le V \le 1$ tenemos la escala de grises de negro a blanco.

Los conceptos de color que se asocian con los términos de tonos, sombras y tintes, se representan en un plano transversal del hexacono HSV, figura 10-8. Al agregar negro en un matiz puro se reduce el valor (*V*). Por tanto se representan diversas sombras con los valores $S = 1.0$ y $0 \le V \le 1$. Al agregar blanco en un tono puro se producen diferentes tintes a lo largo del plano superior del hexacono, así es que se representan diversos tintes con los valores $0 \le S \le 1.0$ y $V = 1$. Se especifican diversos tonos al agregar tanto

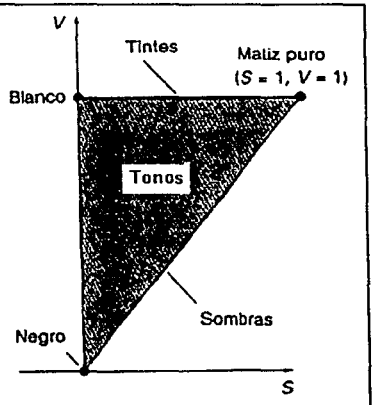


Figura 10-8
Sección transversal de hexacono HSV, que representa las regiones para sombras, tintes y tonos. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 607.

negro como blanco, produciendo puntos de color en el área transversal triangular del hexacono.

El ojo humano puede distinguir alrededor de 128 matices distintos y más o menos 130 tintes diferentes (niveles de saturación). Para cada matiz se puede detectar un cierto número de sombras. Se pueden detectar cerca de 23 sombras con colores amarillos y más o menos 16 sombras diferentes en el extremo azul del espectro. Esto implica que podemos distinguir 128x130x23 = 82,720 colores diferentes. Para la mayor parte de las aplicaciones son suficientes 128 matices, 8 niveles de saturación y 15 especificaciones de sombras, lo cual provee 16,384 colores y 14 bits de almacenamiento para cada color.

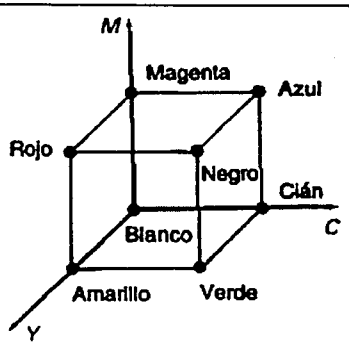


Figura 10-9
Modelo de color CMY. Este modelo define los colores con un proceso sustractivo en un cubo unitario. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 605.

10-5 El modelo de color CMY

Se trata de un modelo de color en el que se definen como colores primarios al cián, al magenta y al amarillo[3]. El modelo CMY se emplea en los dispositivos de salida impresa para generar imágenes de color. Bueno, si se pregunta cuales son los dispositivos de salida impresa, pues bien, son los impresores a color, los trazadores, etc.

El sistema CMY también es un modelo que se define en un sistema rectangular de coordenadas. La figura 10-9 nos ilustra la situación. Ahora bien, si recuerda al modelo RGB, podemos formar el cián combinando luces verde y azul. Por tanto, cuando se refleja la luz blanca en una tinta color cián, la luz reflejada no debe tener ningún componente rojo, es decir, la tinta absorbe o sustrae la luz roja. De modo similar la tinta magenta sustrae el componente verde de la luz incidente y el amarillo sustrae el componente azul.

Ahora bien, veamos si hemos entendido esto de la sustracción. Supongamos que iluminamos con luz blanca una mezcla de tinta magenta y cián ¿cuál será el color resultante de la tinta? Pues azul. Veamos porqué, el magenta absorbe el componente verde y el cián absorbe el componente rojo, así que solo queda el componente azul de la luz incidente.

Ahora vamos a especificar un poco el modelo de color CMY que se ilustra en la figura 10-9. En este modelo el punto (1,1,1) representa el negro porque se sustraen todos los componentes de la luz incidente. Entones el origen representa la luz blanca. Finalmente, las cantidades iguales de cada uno de los colores producen grises a lo largo del diagonal principal del cubo.

Los sistemas de salida impresa emplean más de los tres colores primarios, la razón implica que la sustracción no es perfecta, por ejemplo, el combinar las tres tintas, cián, magenta y amarillo, lo más que se puede lograr es un color gris, por ello, las impresoras de color emplean un depósito de tinta negra. La figura 10-10, ubicada al final del capítulo nos muestra un ejemplo de cómo los tres colores primarios se combinan para formar otros tres colores complementarios y el negro.

Finalmente podemos expresar la conversión de una representación RGB a CMY mediante la siguiente operación

[3] De hecho, CMY con las iniciales de los vocablos anglosajones *cyan*, *magent* y *yellow*.

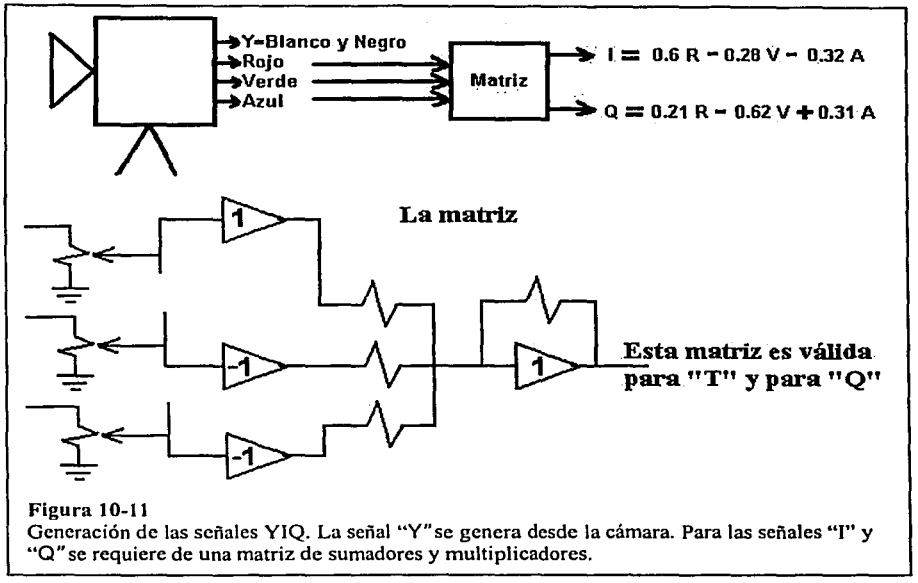


Figura 10-11
Generación de las señales YIQ. La señal "Y" se genera desde la cámara. Para las señales "I" y "Q" se requiere de una matriz de sumadores y multiplicadores.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (10\text{-}6)$$

La operación inversa se deja al lector.

10-6 Modelo de color YIQ

Un monitor RGB requiere de tres señales separadas para generar una imagen en color. Un televisor utiliza una sola señal en la cual se encuentra toda la información necesaria para formar una imagen en color. Este modelo fue estandarizado por la *National Television System Comite* (NTSC).

La señal YIQ nace de tres señales cuyo significado visual ahora voy a comunicarle:

- "*Y*" es la componente que implica una escala de grises de negro a blanco. Normalmente se le conoce como la información de luminancia o brillantez.
- "*I*" es la componente que implica otra escala de grises, pero que va del naranja al cián. Esta componente es lo que empíricamente ofrece un sombreado parecido al tono de la piel.
- "*Q*" es la componente que implica una escala de grises que va del verde al magenta.

La figura 10-11 nos ofrece un diagrama que indica el cómo se generan las tres señales YIQ. La figura 10-12 es una continuación de la 10-11 y nos muestra el cómo se mezclan

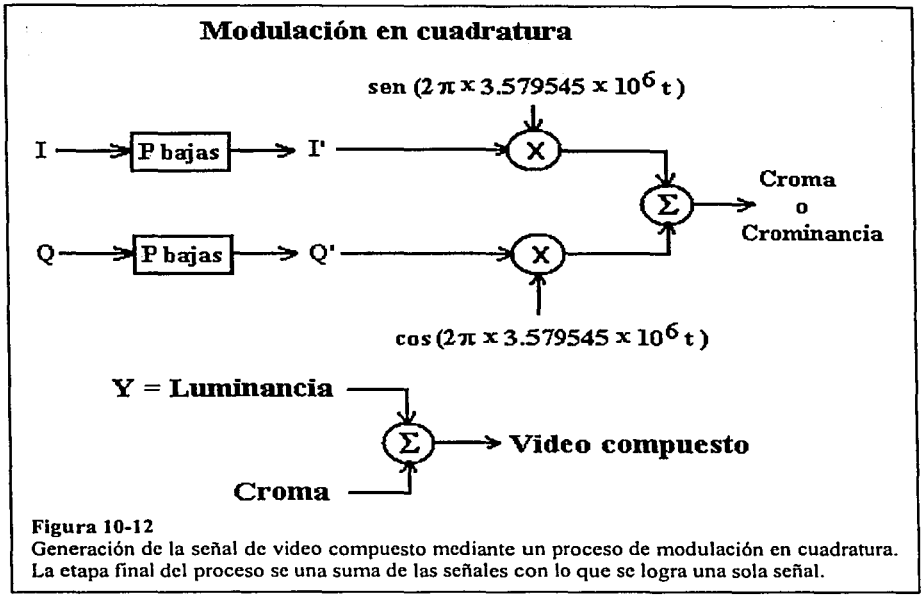


Figura 10-12
Generación de la señal de video compuesto mediante un proceso de modulación en cuadratura. La etapa final del proceso se una suma de las señales con lo que se logra una sola señal.

las señales IQ para formar la **señal de croma**, o bien, de **crominancia**, como suele llamársele normalmente. Finalmente, la figura 10-12 nos muestra la etapa final en la que las señales de luminancia y de crominancia se combinan para formar la señal de NTSC que normalmente se califica como **señal de video compuesto**.

Tal vez estos datos le sirvan: la señal de video compuesto ocupa un ancho de banda de 4MHz. Este mismo ancho de banda se asigna a la señal de luminancia *Y*. 1.5 MHz es el ancho de banda de la señal *I* y 0.6 MHz corresponden a la señal *Q*. La señal de croma requiere de 1.5 MHz.

10-7 El modelo de color HLS

Otro modelo que se basa en parámetros de color intuitivos es el sistema HLS. Este modelo tiene una representación de cono doble que se ilustra en la figura 10-13. Los parámetros de color en este modelo se llaman matiz (*H*), brillantez (*L*) y saturación (*S*).

El matiz corresponde a un color puro y se especifica como un ángulo alrededor del eje del cono doble. En este modelo $H=0^0$ corresponde al azul. Los colores restantes se especifican alrededor del perímetro del cono. Por ejemplo, el magenta se encuentra a 60^0, el rojo a 120^0 y el cián se localiza a 180^0. Los colores complementarios tienen una separación de 120^0.

El eje vertical de este modelo se llama brillantez, L. En $L=0$, tenemos el negro y el blanco está en $L=1$. La escala de gris está a lo largo del eje L y los matices puros se encuentran en el plano $L=0.5$.

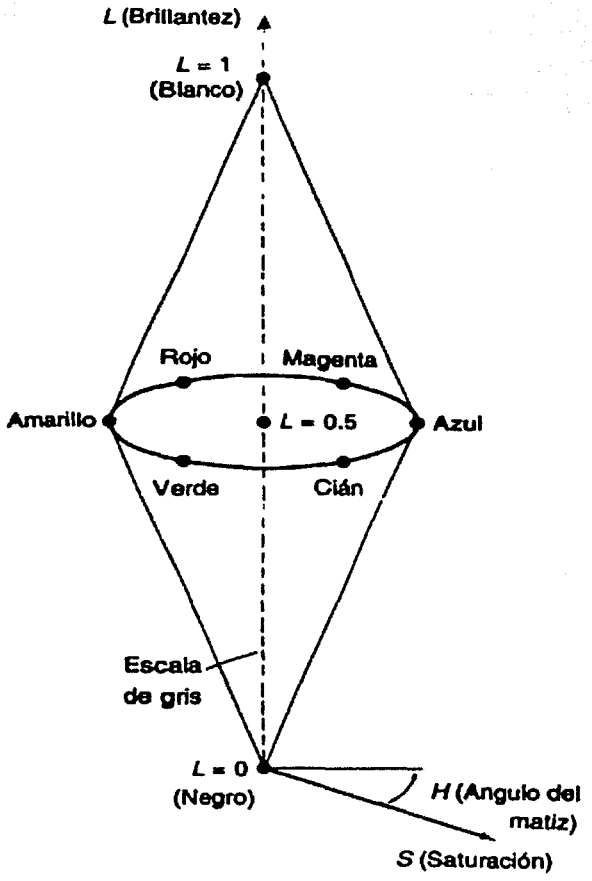


Figura 10-13
Cono doble HLS. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall Hispanoamericana, S.A., 2° ed., 1995, pg. 610.

El parámetro saturación (*S*) especifica la pureza relativa de un color. Este parámetro varía de 0 a 1 y los matices puros son aquellos para los cuales *S=1* y *L=0.5*. Conforme *S* se reduce se agregan tintes a los matices.

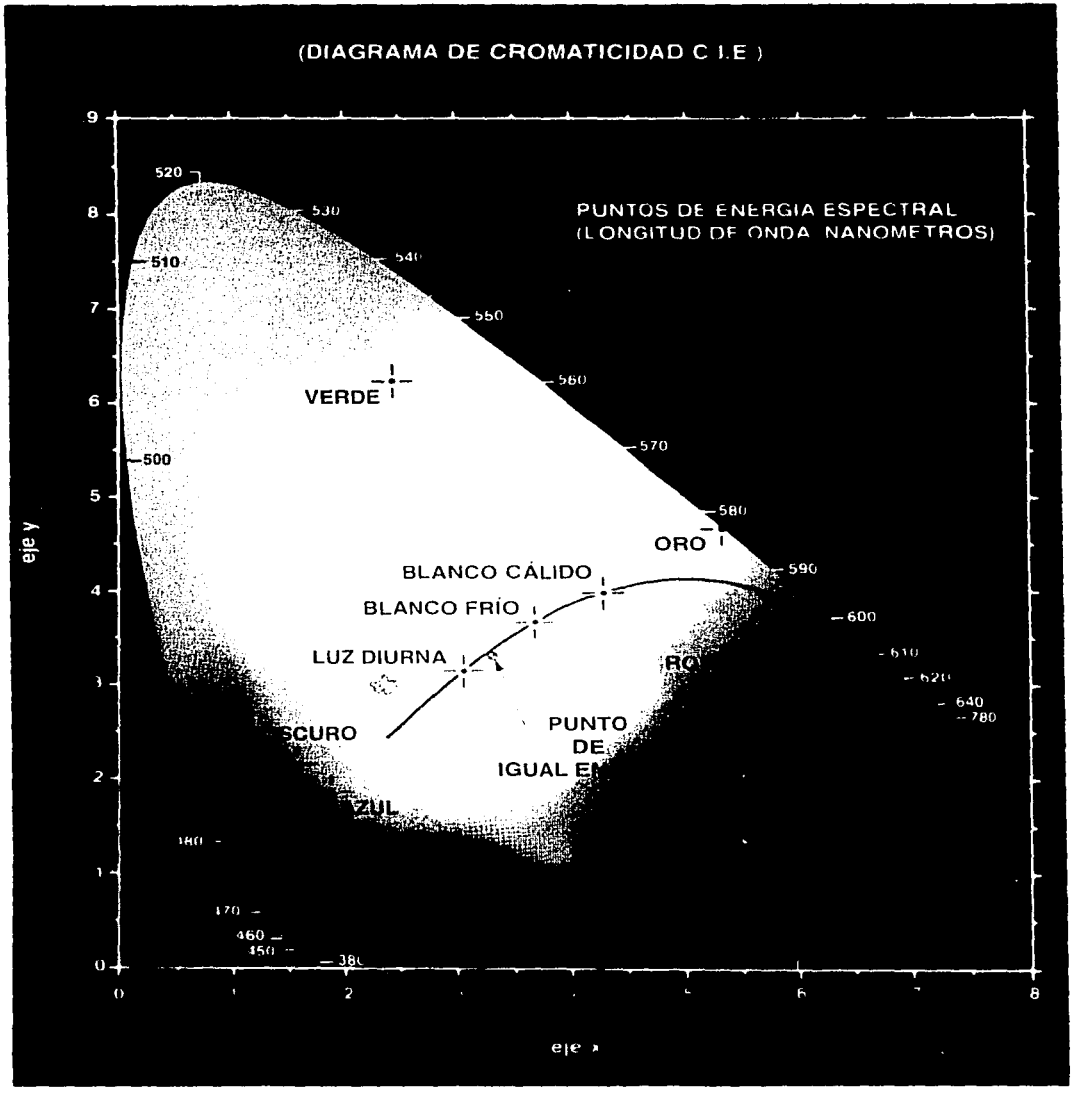


Figura 10-5
Diagrama de cromaticidad de la CIE. Rafael C. González, Richard E. Woods, "*Tratamiento Digital de Imágenes*", Addison-Wesley, 1992, Lámina IV.

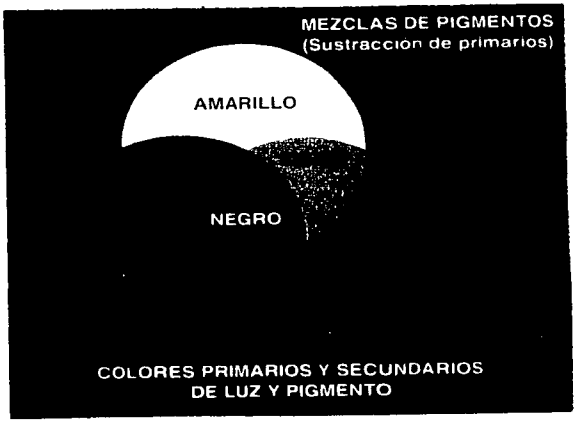
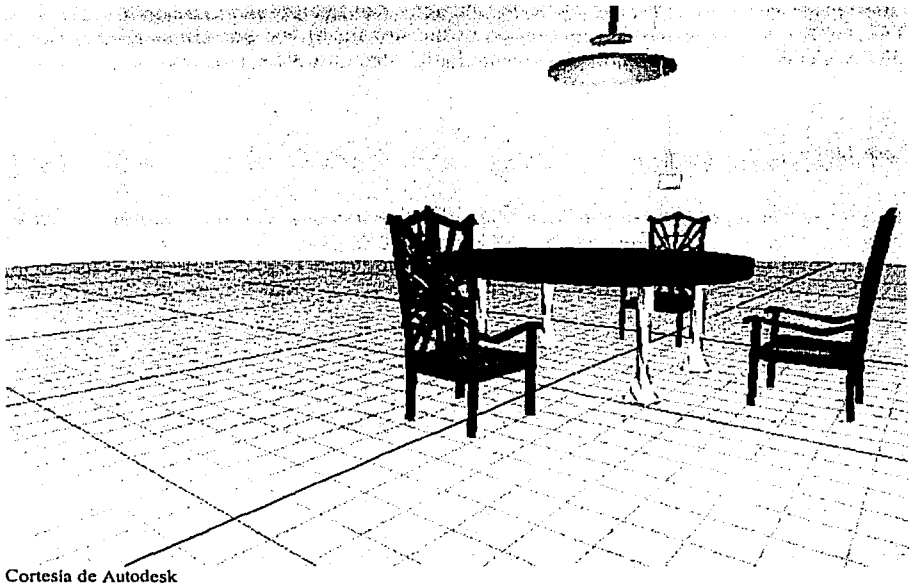


Figura 10-10
Mezclas de pigmentos cián, magenta y amarillo. Rafael C. González, Richard E. Woods, "*Tratamiento Digital de Imágenes*", Addison-Wesley, 1992, Lámina III.



Cortesía de Autodesk

Capítulo 11 Modelos de iluminación

Una vez que la luz llega hasta nuestros ojos, activa un proceso de percepción que determina lo que en realidad vemos en una escena. Por supuesto, cada individuo tiene su propia percepción de la realidad y casi siempre está en función de que el observador preste atención a los detalles. Veamos ahora el caso de la síntesis de imágenes. En esta actividad encontramos el concepto de foto realismo e implica lo siguiente:

- Representaciones gráficas exactas de las superficies de los objetos en la escena.
- Síntesis de los fenómenos de iluminación basándose más en principios físicos y menos en principios psicológicos.

El primer punto se refiere a la cantidad de detalles con los que se define la superficie de un cuerpo. El segundo se refiere a las propiedades ópticas de la luz y de las superficies en la escena y es este tópico el que nos interesa en el presente capítulo.

Veamos entonces, el modelado de los colores y efectos de iluminación que vemos en un objeto es un proceso complicado e implica principios tanto físicos como psicológicos. Algunos parámetros implicados en este proceso son:

- Propiedades ópticas de las superficies y de la luz que incide en ellas.
- Posiciones y orientaciones relativas de las fuentes de luz, de las superficies y del observador.

Así que estos parámetros nos dan a entender que es necesario conocer el cómo interacciona la energía electromagnética de la luz con las superficies... ¡Alto! Modelar la

interacción de la luz con una superficie es una actividad que requiere nuestra pericia en Óptica. Bueno, seguramente esta ciencia no es nuestro fuerte, así que empezaremos con algunos conceptos básicos que pomposamente llamaremos **modelos empíricos**.

11-1 Iluminación

El primer concepto que requerimos es el de **iluminación**. Cuando apuntamos nuestra lámpara de mano a la superficie de una escena, decimos que la estamos iluminando. De manera más formal, iluminación se refiere a la cantidad de energía, por unidad de tiempo, que incide normalmente en una superficie unitaria. Así que las unidades de la iluminación pueden ser watt por metro cuadrado.

11-2 Modelos de iluminación y de representación de superficies

Un **modelo de iluminación** es una técnica empleada para calcular la cantidad de iluminación, reflejada o transmitida, de un punto particular de una superficie en una dirección de vista específica. A los modelos de iluminación se les puede llamar con los siguientes nombres: modelo de alumbrado o bien, modelo de sombreado.

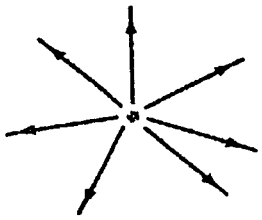


Figura 11-1
Fuente de luz de punto. Tal como lo lee, un punto es una posición y no tiene dimensiones de longitud. Los rayos de luz emanan radialmente desde la fuente. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 195, pg 522.

Un **método de presentación de superficies** es una técnica que indica como emplear el modelo de iluminación para una superficie a diferencia de un punto de la misma. En otras palabras, esta técnica se emplea para calcular las cantidades de iluminación para cada punto de una superficie en una dirección de vista específica. Algunos métodos de presentación interpolan los parámetros entre dos puntos cuyas cantidades de iluminación ya fueron calculadas. A estos métodos se les conoce como algoritmos de **imagen-espacio de línea de rastreo**. Otros métodos invocan un modelo de iluminación para cada punto de una superficie.

Otro tipo de métodos, conocidos como **métodos de rastreo de rayos de luz**, no se avocan a calcular solo reflexiones, también calculan la refracción de estos rayos.

Los métodos de imagen-espacio por línea de rastreo requieren de menos cálculos, por lo que son muy requeridos en representaciones de tiempo real.

Los métodos de presentación de superficies los he reservado para un capítulo posterior ya que no son tan simples como he dado a entender en el presente capítulo.

11-3 Fuentes de luz

Nominalmente podemos considerar las siguientes fuentes de luz:

- Fuentes de emisión de luz.
- Fuentes de luz por reflexión de la luz incidente.
- Fuentes de luz por transmisión de la luz incidente.

Fuentes de emisión de luz.

Las fuentes de emisión de luz las tratamos todos los días y las vemos como el Sol, las estrellas, las lámparas incandescentes y fluorescentes, etc. Las fuentes de emisión de luz pueden clasificarse de acuerdo a las necesidades de nuestros modelos de iluminación en:

- Fuente de luz puntual.
- Fuente de luz distribuida.
- Fuente de luz omnidireccional.
- Fuente de luz direccional.

El modelo más sencillo para una fuente de luz, es una fuente de luz de punto. Como puede observar en la figura 11-1, si consideramos el concepto de rayos de luz, éstos se emiten desde la fuente y siguen trayectorias que divergen en forma radial. Observe que hablamos de una fuente de luz de punto, es decir, no tiene dimensiones espaciales como longitud o ancho, por lo que este modelo es adecuado cuando las dimensiones de la fuente de luz son pequeñas en comparación con los objetos cercanos.

Una fuente de luz distribuida como la mostrada en la figura 11-2, puede ser una lámpara fluorescente y la podemos aproximar con múltiples fuentes de luz puntual. Ahora bien, observe que esta fuete de luz se distingue de la puntual en que si tiene dimensiones espaciales: al menos tienen una dimensión de longitud.

Una fuente de luz omnidireccional es una fuente de luz que irradia en todas direcciones, tal como puede verse en la figura 11-1.

Una fuente de luz direccional se caracteriza porque la mayor parte de la energía radiada se concentra alrededor de una dirección específica. Un ejemplo de este tipo de fuentes de luz son las lámparas de mano.

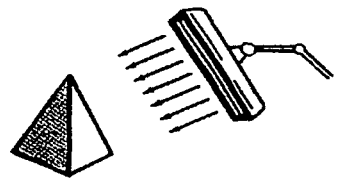


Figura 11-2
Objeto iluminado por una fuente de luz distribuida. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 195, pg 522.

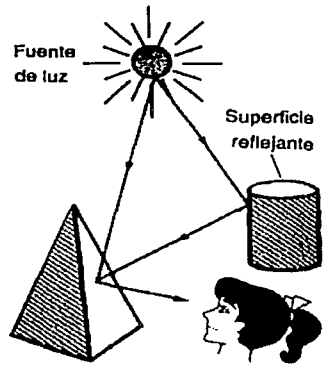


Figura 11-3
La luz que se ve en una superficie opaca no luminosa, es en general, una combinación de la luz emitida por una fuente y de las reflexiones de luz de otras superficies. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 195, pg 522.

Fuentes de luz por reflexión

Las fuentes de luz por reflexión pueden ser, por ejemplo, la Luna, un espejo y en general cualquier cuerpo opaco y hasta los cuerpos transparentes. Considere, por ejemplo, un objeto opaco no luminoso en una sala con ventanas a medio día. Seguro sabrá que el objeto es visible, o sea, podemos ver la luz que se refleja en las superficies del objeto. Para resumir, la luz reflejada que vemos es la suma de las contribuciones de las fuentes de luz y otras superficies reflejantes en la escena; vea la figura 11-3. De esta forma, una superficie que no se expone directamente a una fuente de luz, puede ser visible si los objetos cercanos están iluminados. Las fuentes de reflexión de luz que vamos a considerar pueden ser una de las siguientes:

- Fuentes de luz por reflexión difusa de la luz incidente.
- Fuentes de luz por reflexión especular de la luz incidente.

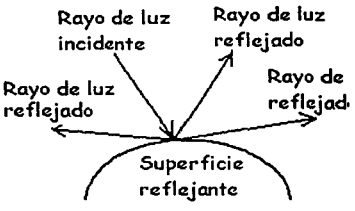


Figura 11-4
Un rayo de luz que incide en una superficie mate es dispersado en varias direcciones a la vez.

Reflexión difusa de la luz

Ahora bien, las superficies mate muy rugosas o ásperas, tienden a dispersar la luz reflejada en todas direcciones por lo que, visualmente, presentan la misma brillantez desde cualquier posición de vista, esto es, irradian la misma cantidad de energía luminosa en todas direcciones. La figura 11-4 ilustra este concepto genérico. A este fenómeno en el que la cantidad de iluminación se refleja por igual en todas direcciones se le conoce como

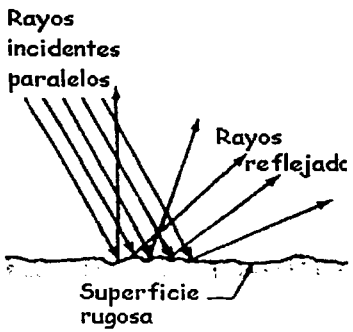


Figura 11-5
La superficie rugosa en la cual incide un manojo de rayos, dispersa éstos en todas direcciones. Murphy-Smoot. "*Física, principios y problemas*", CECSA, 4° ed., 1986, pg 274.

reflexión difusa de la luz. Este concepto es una buena aproximación a la realidad, la figura 11-5 muestra como podría ser posible este fenómeno.

Lo que llamamos color de un objeto es el color de la reflexión difusa de la luz incidente. Un objeto azul iluminado por una luz blanca, por ejemplo, refleja el componente azul de la luz blanca y absorbe por completo el resto de los componentes. Si se ve un objeto azul con una luz roja, éste parecerá negro dado que se absorbe toda la luz incidente.

Reflexión especular de la luz

Seguramente, habrá notado en las superficies brillantes algunos pequeños toques de luz o manchitas brillantes, las cuales reciben el nombre de fuentes de luz por reflexión especular. La cantidad de iluminación reflejada de esta forma está en función de la posición de vista y es mayor en superficies brillantes que en superficies opacas. En un segmento posterior veremos como se puede modelar este tipo de reflexión.

Fuentes de luz por transmisión

En una fuente de luz por transmisión, los rayos de luz incidentes[1] se desvían al pasar por el material y esto es lo que se dice transmisión de la luz o bien, refracción. Un ejemplo de fuentes de transmisión es el vidrio, algunos plásticos, cristales minerales, etc. La figura 11-6 nos ilustra el fenómeno de la transmisión de un rayo de luz que incide en la superficie de un prisma semicircular. El prisma está hecho con vidrio crown.

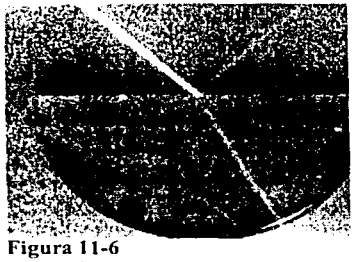


Figura 11-6
Rayos de luz refractado al ingresar en un prisma semicircular. Murphy-Smoot, "*Física, principios y problemas*", CECSA, 4° ed., 1986, pg.274.

11-4 Modelos básicos de iluminación

Los modelos empíricos que se presentan a continuación ofrecen métodos sencillos y rápidos para calcular las cantidades de iluminación en un punto en una superficie. Debe tomar las siguientes consideraciones:

- Como parte de estos métodos, se acuerda que el tipo de superficie a considerar es del tipo mate, por lo que se presentarán reflexiones difusas de la luz.
- Estos modelos, aunque no son aplicables a superficies transparentes, se pueden extender a éstas.
- Estos modelos trabajan con fuentes de luz monocromáticas, es decir, fuentes de luz de un sólo color. No obstante se puede emplear un modelo de color aditivo como el RGB para crear cualquier tipo de iluminación.

Los modelos básicos de iluminación son:

- Luz ambiente y su reflexión difusa.
- Fuente de luz puntual y su reflexión difusa.
- Fuente de luz puntual y el Modelo Phong para su reflexión especular.

[1] El concepto de rayo de luz es muy socorrido en la Óptica geométrica.

Concepto general de reflexión

Los modelos de iluminación considerados arriba fueron diseñados considerando una fuente de luz monocromática y polarizada en todas direcciones. Así, antes de revisar estos modelos de iluminación, revisaremos algunos conceptos básicos de reflexión.

Visualmente, una superficie cuya orientación es perpendicular a la dirección de la luz incidente, tiene una apariencia más brillante que si la superficie estuviera inclinada en un ángulo oblicuo a la dirección de la luz que llega: realice un experimento con una hoja de papel. Ahora bien, observe la figura 11-7. En ella se muestran dos manojos de rayos paralelos e incidentes en dos parches de superficie plana con la misma área y orientaciones espaciales distintas, en relación con la dirección de la luz incidente. Note que la superficie que está inclinada recibe menos rayos de luz y por tanto una menor cantidad de iluminación incidente.

Ahora vamos a considerar algo de matemáticas. Si queremos saber cuantos rayos de luz inciden de manera normal (normal significa perpendicular) en el parche de superficie inclinado, sólo hacemos.

$$\#\ de\ \ rayos\ \times\ \cos\theta_i \qquad (11\text{-}1)$$

De manera más formal, la iluminación radiada por una fuente de luz cercana la representamos como I_i. En consecuencia, la iluminación que incide normalmente en un área unitaria de una superficie, por unidad de tiempo, es:

$$I_i \cos\theta_i \qquad (11\text{-}2)$$

En donde θ_i es el ángulo de incidencia considerado a partir de la normal a la superficie. La figura 11-8 ilustra este concepto.

Similarmente, la iluminación reflejada por esa área es I_r, sin embargo nos interesa la energía que sale normalmente (normalmente significa perpendicularmente) de un área unitaria en la unidad de tiempo y la expresamos así:

$$I_r \cos\theta_r \qquad (11\text{-}3)$$

La figura 11-8 resume la situación que hemos descrito con ecuaciones. Ahora bien, lo que sigue es definir un parámetro que depende del tipo de superficie y que llamaremos **reflectancia o coeficiente de reflexión**. Este parámetro se representa como K_r y está definido como:

$$K_r = \frac{I_r \cos\theta_r}{I_i \cos\theta_i} \qquad (11\text{-}4)$$

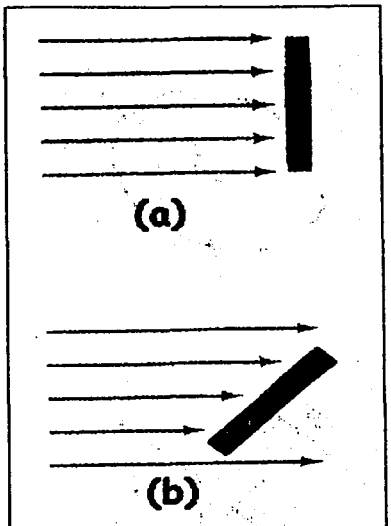


Figura 11-7
Una superficie perpendicular a la dirección de la luz incidente (a) se ilumina más que una superficie del mismo tamaño en ángulo oblicuo (b) a la dirección de la luz incidente. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 195, pg 525.

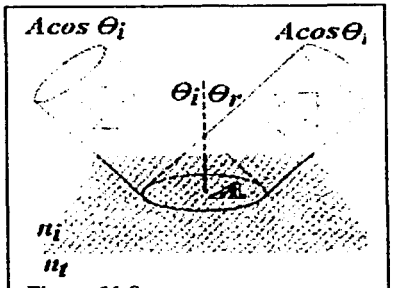
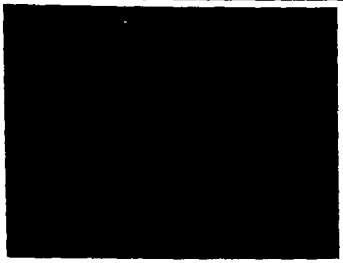


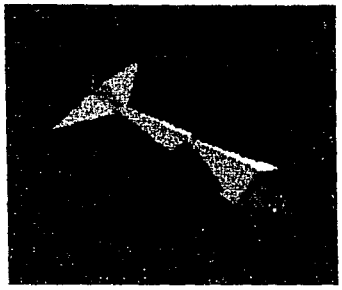
Figura 11-8
Concepto general de reflexión. Cada cilindro representa la iluminación incidente en una superficie plana. Note pues los ángulos de incidencia y de reflexión.

Luz ambiente y su reflexión difusa

Empíricamente, una superficie que no está expuesta de manera directa a una fuente de luz será aún visible... ¡Alto! ¿Cómo es esto posible? Pues bien, la luz que incide en el cuerpo que nos interesa es reflejada por otros cuerpos a su alrededor. Ahora bien, puesto que esta luz existe debido a que se refleja en distintos cuerpos, podemos pensar que no tiene



(a)



(b)

Figura 11-9
Figuras iluminadas con luz ambiente. Note que no hay sensación de espacio 3D en estas figuras. Figura obtenida del sitio http://serdis.dis.ulpgc.es/~atrujill/iga/

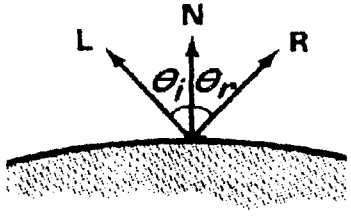


Figura 11-10
El ángulo de reflexión de un rayo es igual al ángulo de incidencia. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 195, pg 528.

características de espacio, o sea, no tiene origen ni dirección. Así que modelaremos esta luz considerando el siguiente principio: *la cantidad de luz ambiente que incide en cada objeto es una cantidad constante para todas las superficies, sin importar su posición y orientación espacial.*

Considere ahora lo siguiente: la iluminación ambiente incidente en una superficie es constante e independiente de la orientación de la misma, por lo que:

$$\theta_i = 0 \quad \therefore \quad \cos\theta_i = 1 \tag{11-5}$$

y dado que la reflexión difusa implica radiar energía en todas direcciones con la misma densidad de potencia, entonces:

$$\theta_r = 0 \quad \therefore \quad \cos\theta_r = 1 \tag{11-6}$$

Así, la cantidad de iluminación reflejada desde cualquier punto de una superficie es:

$$I_{r,amb,dif} = K_{r,amb,dif} I_{i,amb} \tag{11-7}$$

La figura 11-9 nos muestra dos ejemplos de iluminación ambiente. En 11-9.a tenemos la representación de una dona (en realidad es un toroide, pero por ahora prefiero llamarlo dona) en tres dimensiones, sin embargo, podrá ver que por su color, en realidad parece una curva azul hecha con plumón. En 11-9.b tenemos la vista en perspectiva de un avión hecho con polígonos planos. Debido a que solo trabajamos con iluminación ambiente, la vista en perspectiva se pierde. En sí, cuando sólo empleamos iluminación ambiente, podemos perder esa sensación de espacio que caracteriza a las escenas 3D.

Fuente de luz de punto y su reflexión difusa.

Este método nos permite modelar las reflexiones de la iluminación a partir de una fuente de luz de punto que irradia luz monocromática.

En este caso, tenemos una fuente de luz con una posición dada y una superficie con una posición y orientación específicas. La conclusión implica que debemos considerar un ángulo de incidencia.

$$\theta_i \geq 0 \quad \therefore \quad \cos\theta_i \leq 1 \tag{11-8}$$

Ya sabemos que la reflexión difusa de la iluminación desde una superficie implica dispersar la energía radiante en todas direcciones con la misma densidad de potencia:

$$\theta_r = 0 \quad \therefore \quad \cos\theta_r = 1 \tag{11-9}$$

Por lo tanto, la cantidad de iluminación reflejada desde cualquier punto de una superficie es:

$$I_{r,dif} = K_{r,dif} I_i \cos\theta_i \tag{11-10}$$

A este modelo se le prefiere dar un tratamiento vectorial, así que considere la figura 11-10. Si $\vec{N}$ es un vector normal unitario para una superficie y $\vec{L}$ es un vector unitario que

apunta en la dirección de la fuente de luz de punto desde cualquier posición en la superficie, entonces.

$$\cos\theta_l = \vec{N} \cdot \vec{L} \qquad (11\text{-}11)$$

Así que la cantidad de iluminación reflejada desde cualquier punto de una superficie también se puede expresar como:

$$I_{r,dif} = K_{r,dif} I_l \left(\vec{N} \cdot \vec{L}\right) \qquad (11\text{-}12)$$

La figura 11-11 nos muestra la misma dona de la figura 11-9, note ahora la diferencia en cuanto al aspecto. Ahora si ya tenemos la sensación de que la dona ocupa un espacio en tres dimensiones.

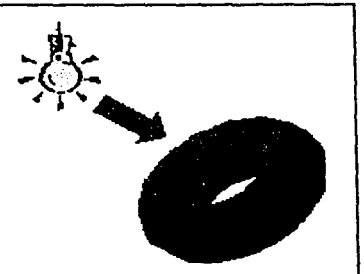


Figura 11-11
Una dona es iluminada por luz ambiente y una fuente de luz puntual. Note ahora que si tenemos sensación de espacio. Figura obtenida del sitio http://serdis.dis.ulpgc.es/~atrujill/iga/

Fuente de luz de punto y el modelo Pong para su reflexión especular.

Ahora bien, cuando vemos una superficie iluminada y brillante, como metal pulido, una manzana o la frente de una persona, vemos un toque de luz o mancha brillante en algunas direcciones de vista. A este fenómeno se le conoce como reflexión especular. La figura 11-12 nos muestra una tetera esmaltada a modo de espejo. Note las reflexiones especulares en la oreja, al centro y en el cuello de la tetera.

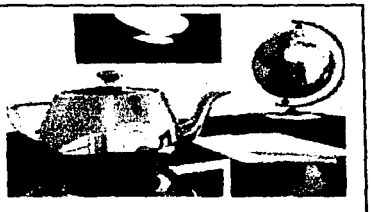


Figura 11-12
Reflexiones especulares en una tetera esmaltada a modo de espejo. Las reflexiones especulares son las manchitas brillantes que puede notar en la oreja, el cuerpo y en el cuello de la tetera. Figura obtenida del sitio http://serdis.dis.ulpgc.es/~atrujill/iga/

Para el desarrollo de este modelo requerimos de una fuente de luz de punto que emita luz monocromática.

De la Óptica Geométrica sabemos que un rayo de luz que incide en un espejo perfecto (reflector ideal) se refleja en un ángulo, respecto de la normal a la superficie, numéricamente igual al ángulo de incidencia: vea la figura 11-13. Visualmente, un observador vería el rayo reflejado solamente si se coloca en la dirección de éste, así que al mencionado ángulo le llamaremos **ángulo de reflexión especular**. De otra forma cuando un manojo de rayos de luz paralelos, inciden en superficies que no son reflectores ideales, se reflejan dispersándose radialmente alrededor del ángulo de reflexión especular: mientras más alejados estemos del ángulo de reflexión, menor será la energía radiada.

La figura 11-13 muestra una dirección de reflexión especular en un punto de una superficie iluminada. El ángulo de reflexión especular θ equivale al ángulo de incidencia. Tanto el ángulo de incidencia, como el ángulo de reflexión se miden en lados opuestos del vector unitario $\vec{N}$ a la superficie. En esta figura utilizamos $\vec{R}$ para representar el vector unitario en la dirección de la reflexión especular ideal; $\vec{L}$ para expresar el vector unitario dirigido hacia la fuente de luz de punto y $\vec{V}$ como el vector unitario que apunta hacia el observador desde la posición de incidencia en la superficie. El ángulo ϕ es el ángulo de vista con respecto a al dirección de reflexión especular $\vec{R}$.

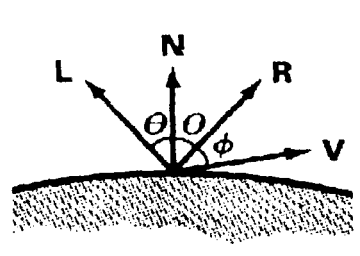


Figura 11-13
Figura obtenida de Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 195, pg. 528.

Para un reflector ideal, la luz incidente se refleja solo en la dirección de reflexión especular, o sea, solo vemos luz reflejada cuando coinciden los vectores $\vec{V}$ y $\vec{R}$, esto es, $\phi = 0$. Los objetos que no son reflectores ideales presentan reflexiones en un intervalo finito de posiciones de vista en relación con el vector $\vec{R}$.

Un modelo empírico para calcular la reflexión especular fue desarrollado por Phong Bui Tuong. Este modelo recibe el nombre de **modelo de reflexión especular de Phong** o sólo

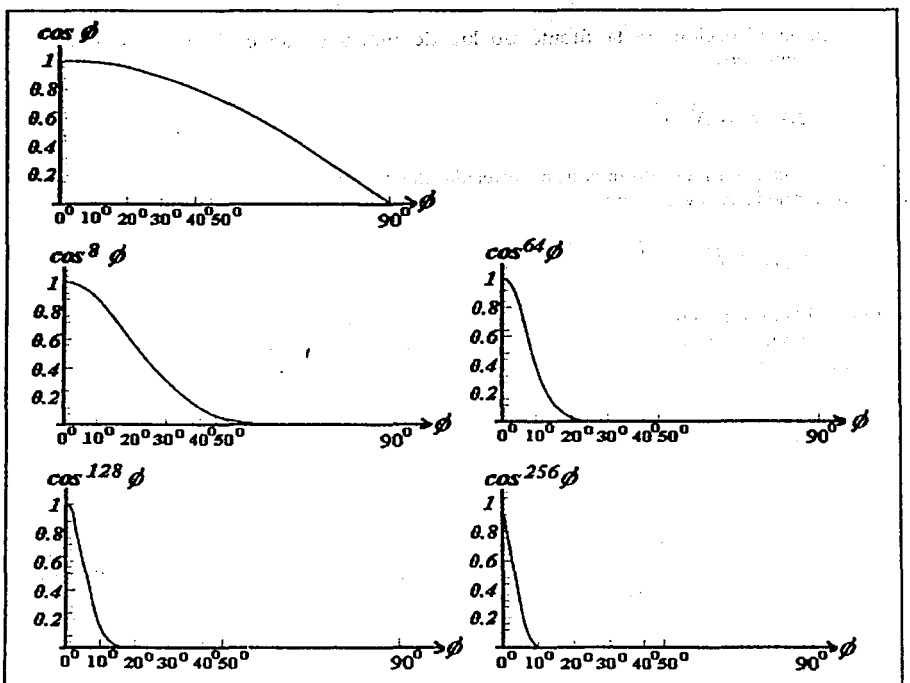


Figura 11-14
Trazos de $\cos^{ns}\phi$ para varios valores del parámetro especular n_s. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2° ed., 195, pg 528.

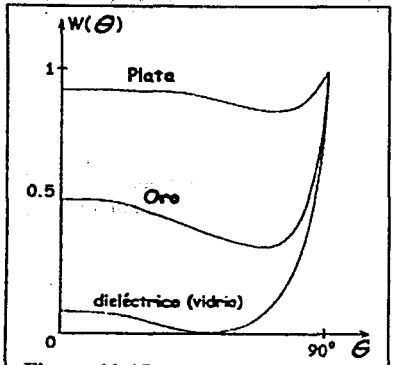


Figura 11-15
Variación aproximada del coeficiente de reflexión especular como una función del ángulo de incidencia para diferentes materiales. Donald Hearn, M. Pauline Baker, "*Gráficas por computadora*", Prentice Hall, 2ª ed., 195, pg. 528.

modelo de Phong. Éste establece que la cantidad de iluminación reflejada se puede calcular como:

$$I_{r,spc} = W(\theta) I_{i,spc} \cos^{n_s}\phi \qquad (11\text{-}13)$$

En esta ecuación, al término n_s se le llama **parámetro de reflexión especular** y su rango de valores varía de cero a infinito. La figura 11-14 muestra el comportamiento de $\cos^{n_s}\phi$. Los valores de n_s se asignan según el tipo de superficie que deseamos representar. Una superficie muy brillante se modela con un valor alto para n_s, digamos 100 o más. En cambio, los valores más bajos, hasta 1, se emplean para superficies más opacas. Para un reflector perfecto, n_s es infinito. Para una superficie rugosa, como un gis o un ladrillo de cenizas, se asignan a n_s valores cercanos a 1.

Pero ¿qué pasa con $W(\theta)$? Bueno, se llama **coeficiente de reflexión especular** y ahora veremos como se usa. Seguramente podemos acordar que distintos materiales como la plata, el oro o el vidrio producen distintas reflexiones especulares. Con el parámetro $W(\theta)$ podemos especificar esa reflexión para un material dado. La figura 11-15 muestra

las variaciones de este coeficiente para distintos ángulos de incidencia. Note que hablamos del ángulo de incidencia.

En muchos materiales opacos, la reflexión es más o menos constante para todos los ángulos de incidencia. Así que podemos modelar $W(\theta)$ con un coeficiente de reflexión especular constante, K_{spc}, el cual varía en el rango de 0 a 1 y su uso depende la experiencia del programador.

Entonces ¿cómo queda la expresión para calcular la reflexión especular en un punto de una superficie? Veamos, debemos considerar que $W(\theta)$ es constante y que además preferimos emplear álgebra vectorial, así que la expresión final sería como:

$$I_{r,spc} = K_{spc} I_{i,spc} \left(\vec{V} \cdot \vec{R}\right)^{n_s} \qquad (11\text{-}14)$$

La figura 11-16 nos muestra a la ya conocida dona, la cual ahora sufre los efectos de la iluminación especular para diferentes valores de n_s. Note la apariencia que adquiere la dona.

Por lo que observamos en la figura 11-16 ¿cómo podemos distinguir una superficie plástica de una superficie metálica? La respuesta es un tanto psicológica y se encuentra en la reflexión especular. Una superficie plástica presenta reflexiones especulares del mismo color de la luz incidente. Para un material no plástico, el color de la reflexión es una función de las propiedades de la superficie y puede ser distinto tanto del color de la luz incidente como del color de las reflexiones difusas.

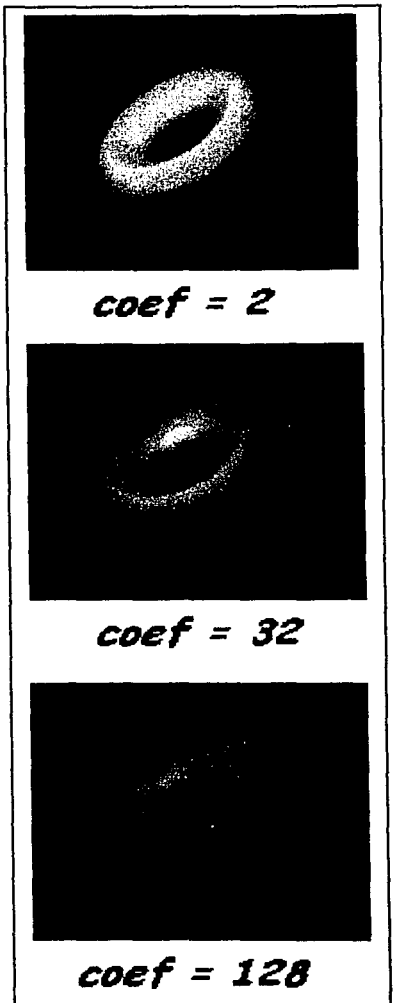


Figura 11-16
La figura muestra las variaciones en la textura de la dona debido a la manipulación del coeficiente de reflexión especular. Figura obtenida del sitio http://serdis.dis.ulpgc.es/~atrujill/iga/

11-5 Modelo de Warn

En los libros de óptica básica podemos encontrar que el flujo luminoso que emana de una fuente disminuye en un factor de $1/d^2$ donde d es la distancia que la luz ha recorrido. La ecuación 11-15 representa la atenuación que sufre la iluminación debido a la distancia.

$$I_{Atenuación} = \frac{1}{r^2} I_{Fuente} \qquad (11\text{-}15)$$

La conclusión de esto es obvia, una superficie cercana se ilumina más que una superficie lejana. Ahora bien ¿cuál es la necesidad de aplicar este modelo? Pues bien, si iluminamos dos superficies paralelas, con los mismos parámetros ópticos, a distancias diferentes y sin considerar la atenuación con la distancia, las dos superficies se sobreponen, con lo que se pierde realismo en la escena.

La experiencia indica que al emplear la ecuación 12-15, no se producen los efectos de iluminación adecuados y esto, porque cuando iluminamos una escena real no empleamos fuentes de luz de punto. En consecuencia se ha planteado la ecuación siguiente que se conoce como la **función de atenuación cuadrática inversa**.

$$f(d) = \frac{1}{a_0 + a_1 d + a_2 d^2} \qquad (11\text{-}16)$$

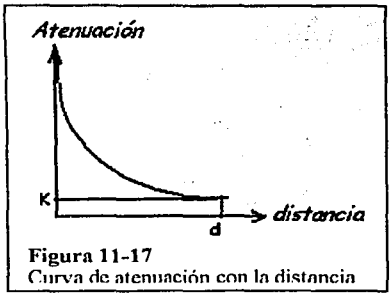


Figura 11-17
Curva de atenuación con la distancia

Aplicación de la función de atenuación cuadrática inversa

Ahora, voy a recomendarle una forma de uso, sin embargo, el lector está en libertad de experimentar con la ecuación. La ecuación 11-17 representa una aplicación de la ecuación 11-16. En esta nueva formulación Ud. controla la atenuación a una cierta distancia. Adicionalmente, se tiene la ventaja de que a distancia cero, la atenuación es nula.

$$K = \frac{1}{1 + ad^2} \qquad (11\text{-}17)$$

La figura 11-17 nos muestra la forma de la curva de atenuación con la distancia. Ahora bien, seguramente observa que aún debe calcular la constante a, no hay problema, recuerde que a una distancia d fijamos la atenuación K, así que el cálculo de la constante es entonces como sigue:

$$a = \left(\frac{1}{k} - 1\right)\frac{1}{d^2} \qquad (11\text{-}18)$$

11-6 La implementación en OpenGL

La implementación en OpenGL implica considerar los siguientes aspectos

- Característica ópticas del material.
- Manejo de las fuentes de luz y activación-desactivación de las mismas.
- Características de las fuentes de luz.

Características ópticas del material

Antes de empezar a activar luces como unos cosacos *tenemos que definir nuestros materiales*. Y eso ¿qué es? Para cada polígono de la escena hay que definir un material de forma que su respuesta a la incidencia de la luz varíe según sea el caso. Está claro que no se refleja igual la luz en un pedazo de oro que una manzana. Por tanto tenemos que decirle a OpenGL de que forma tendrá que tratar cada pedacito de la geometría: ese pedacito de polígono ¿es de oro? o por el contrario forma parte de mi manzana. Se pueden especificar diferentes parámetros en cuanto al material para cada polígono. Es una tarea ardua pero lógicamente, a más variedad de comportamientos más real se verá la escena. Cada vez que se llama a la correspondiente función se activan esos valores que no cambiarán hasta llamarla de nuevo con otros.

La función siguiente nos permite especificar las características de los materiales:

```
GLvoid glMaterialfv ( GLenum face, GLenum pnam, const GLfloat *params );
```

GLenum face	GLenum pname	const GLfloat *params
GL_FRONT	GL_DIFFUSE	(R, G, B, 1.0)
GL_BACK	GL_AMBIENT	(R, G, B, 1.0)
GL_FRONT_AND_BACK	GL_AMBIENT_AND_DIFUSE	(R, G, B, 1.0)
	GL_SPECULAR	(R, G, B, 1.0)
	GL_SHININESS	[0, 127]

En esta tabla se observan los valores que pueden adoptar los parámetros de la función. En el caso de la columna *face* tenemos tres posibilidades dependiendo de sí la característica en cuestión debe aplicarse al lado visible (FRONT) , al no visible (BACK) o a ambos (FRONT_AND_BACK). En cuanto a la columna *pname* se decide aquí cual es la característica que vamos a definir en concreto; las posibilidades se muestran en la tabla anterior y corresponden al comportamiento del material para distintos tipos de iluminación. Por último **params*, donde damos los valores concretos de cada comportamiento especificado por el parámetro *pname*: son tres valores, de hecho tres números en punto flotante definidos en el rango de 0 a 1 y que en conjunto especifican un color RGB. Ese color define exactamente como debe verse el objeto que se va a representar en cuanto a color ambiente, difuso y especular.

Hay una excepción en el caso de GL_SHININESS. Este parámetro es el coeficiente de reflexión especular n_s. Si usamos esta constante como segundo parámetro, el tercero tendrá que se un número entre 0 y 127.

Algunos valores típicos son: 0.8 para las tres componentes en GL_DIFUSSE, de 0.2 para GL_AMBIENT y GL_SPECULAR. Por supuesto, tendremos que retocar estos valores hasta conseguir el efecto deseado. Si los dejamos como están, no nos gustará lo que veremos.

Luces

OpenGL soporta en principio hasta 8 luces simultáneas en un escenario. De hecho también está en función de la máquina que poseamos y de la RAM que le dejemos usar. Las luces cuentan con nombre propio del estilo: GL_LIGHT0, GL_LIGHT1, GL_LIGHT2 y así sucesivamente. Para activar o desactivar una de ellas, por ejemplo la cuarta luz, es decir la número 3, aplicamos:

```
glEnable(GL_LIGHT3);
glDisable(GL_LIGHT3);
```

También podemos activar o desactivar todo el cálculo de iluminación con:

```
glEnable(GL_LIGHTING);
glDisable(GL_LIGHTING);
```

Características de las fuentes de luz

Tenemos que decirle a OpenGL cuales son las propiedades de cada una de nuestras luces. Para ello utilizaremos la función:

```
glvoid glLightfv ( GLenum light, GLenum pnam, const GLfloat *params );
```

GLenum pname	const GLfloat *params
GL_DIFFUSE	(R, G, B, 1.0)
GL_AMBIENT	(R, G, B, 1.0)
GL_AMBIENT_AND_DIFUSE	(R, G, B, 1.0)
GL_SPECULAR	(R, G, B, 1.0)
GL_POSITION	(X, Y, Z, 1.0)

El valor *light* será siempre el número de la luz a la que nos estemos refiriendo: GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, etc. El parámetro *pname* se refiere al modelo de iluminación que se va a aplicar. En cuanto a **params*, le pasamos un arreglo de valores RGBA. ¿RGBA? Así es, los primeros tres ya los conocemos, se refieren al modelo aditivo

RGB, sin embargo el cuarto parámetro nos es desconocido. *A* es la abreviación de alfa y se interpreta como el nivel de opacidad de una superficie: *0* significa transparencia total y de aquí podemos usar cualquier valor hasta *1*, objeto totalmente opaco. Y esto ¿para qué servirá? Pues para poder ver a través del objeto o sea lo que hay detrás.

La característica ambiental

Para definir una luz ambiental para una escena empleamos la función como sigue:

```
glvoid  gllightfv (glenum light, GL_AMBIENT, const glfloat *params)
```

Este comando define la contribución de alguna de las fuentes de luz a la luz ambiental de la escena. Por defecto la contribución es nula.

La característica difusa

Ahora deseamos definir una fuente de luz puntual de cierto color:

```
glvoid  gllightfv (glenum light, GL_DIFUSSE, const glfloat *params)
```

La componente difusa de la fuente es lo que entendemos como el color que tiene la luz. Para GL_LIGHT0 los valores RGBA por defecto valen 1.0. Para el resto de las luces los valores por defecto son 0.0.

La característica especular

Ahora deseamos controlar el color de una fuente de luz que genere una machita brillante en alguna superficie:

```
glvoid  gllightfv (glenum light, GL_SPECULAR, const glfloat *params)
```

Como ya mencioné, la característica especular es la responsable de las zonas más brillantes en los objetos.

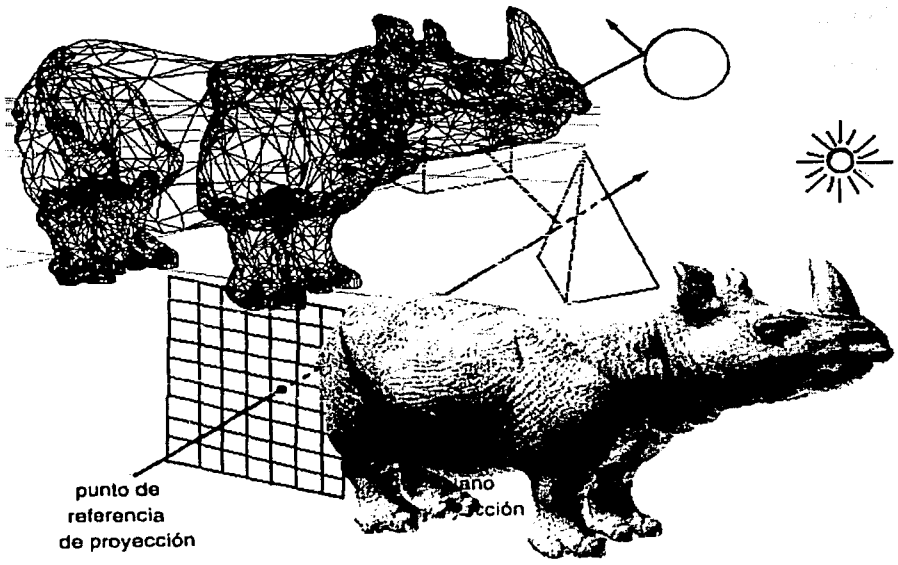
Para conseguir un efecto suficientemente realista deberíamos dar a este parámetro el mismo valor que la componente difusa. Al igual que en el caso anterior, en GL_LIGHT0, los valores RGBA valen 1.0 mientras que el resto valen 0.0.

Colocando las luces

Ha observado que definimos las fuentes de luz únicamente por su color, pero no hemos establecido la posición de las mismas en la escena. Así que debemos especificar donde queremos colocar cada una de las fuentes de luz. Para ellos utilizamos la función de siempre.

```
glvoid  gllightfv (glenum light, GL_POSITION, const glfloat *params)
```

Note el uso de la constante GL_POSITION. En este caso **params* se corresponde con el valor de la coordenada homogénea $[X,Y,Z,W]$ que especifica donde se coloca la fuente de luz. Ahora bien, el parámetro W, que conocemos como el valor homogéneo, tiene un significado especial. Si $W = 0.0$ se considerará que la luz se encuentra



Cortesía de Auodesk

Capítulo 12 Métodos de presentación de superficies

Las superficies cuadráticas y de polígonos proporcionan descripciones exactas para objetos euclidianos como poliedros y elipsoides; las superficies de spline son útiles para diseñar alas de aeronaves, carrocerías y cualquier otra estructura de superficie curva. Los métodos fractales y los sistemas de partículas nos permiten dar representaciones exactas para las nubes, montones de hierba y otros objetos naturales. Los despliegues de isosuperficies son muy útiles en la ingeniería civil para la generación de mapas topográficos.

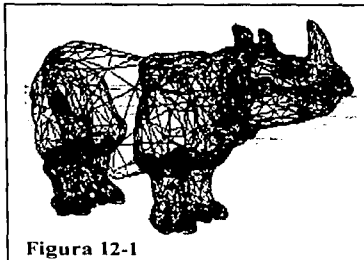


Figura 12-1
Poliedro en forma de rinoceronte. Figura cortesía de Autodesk.

12-1 Aproximación de superficies con facetas poligonales

La representación de frontera que más se utiliza para un objeto gráfico tridimensional es un conjunto facetas poligonales planas que encierran el interior del objeto. La figura 12-1 nos muestra el ejemplo de un poliedro construido con facetas triangulares. El hecho de generar superficies con facetas, en particular triangulares, facilita y acelera la representación de cualquier objeto, ya que todas las superficies se describen con ecuaciones lineales.

Tablas de polígono

Normalmente especificamos una faceta poligonal con un conjunto de coordenadas de vértices y parámetros de atributos asociados. Al dar entrada a la información de cada polígono, los datos se colocan en tablas que se van a utilizar en el subsiguiente procesamiento, despliegue y manipulación de objetos en una escena. Las tablas de datos de polígonos se pueden clasificar en dos grupos:

- tablas geométricas.
- tablas de atributos.

Las tablas geométricas contienen las coordenadas de vértices y parámetros para identificar la orientación espacial de las superficies de polígono. Las tablas de atributos contienen parámetros que especifican el grado de transparencia del objeto y las características de reflectividad y textura de su superficie.

Una organización conveniente para almacenar los datos geométricos es crear tres tablas: una tabla de facetas, una tabla de vértices y una tabla de vectores normales a las facetas. La figura 12-2 nos ilustra el concepto de las tablas.

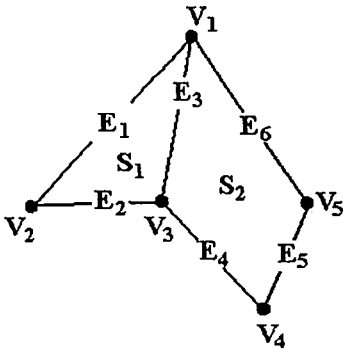


TABLA DE PRIMITIVAS		
Índice	Tipo	Vértices
S1	Triangle	V1,V2,V3
S2	Quad	V3,V4,V5,V1

TABLA DE VÉRTICES	
V1	x1,y1,z1
V2	x2,y2,z2
V3	x3,y3,z3
V4	x4,y4,z4
V5	x5,y5,z5

TABLA DE VECTORES NORMALES		
Índice	Vértice prueba	Superficie
N1	V3	S1
N2	V1	S2

Figura 12-2
Tablas de polígono.

Ecuación del plano

Dado que trabajamos con facetas poligonales para construir superficies, particularmente triángulos, requerimos cierta información como la posición de los vértices que definen cada faceta, el color de la misma y grado de transparencia. Ahora bien, si deseamos agregar efectos de iluminación, requerimos conocer la orientación espacial de la superficie del polígono.

Recordemos entonces la ecuación del plano:

$$Ax + By + Cz + D = 0 \qquad (12\text{-}1)$$

en donde:

(x,y,z)	Representan un punto cualquiera del plano.
<A,B,C>	Representan un vector normal a la superficie y que es el elemento que nos interesa.
D	Es un valor característico que surge de las posiciones de los vértices.

Veamos ahora como se obtienen los coeficientes de la ecuación 12-1 a partir de tres vértices. Para este propósito seleccionamos tres vértices sucesivos de un polígono triangular (x_1, y_1, z_1), (x_2, y_2, z_2) y (x_3, y_3, z_3). Sustituimos ahora sus coordenadas en la ecuación 12-2 a forma de obtener un sistema de 3 ecuaciones simultáneas:

$$\left(\frac{A}{D}\right)x + \left(\frac{B}{D}\right)y + \left(\frac{C}{D}\right)z = -1 \qquad (12\text{-}2)$$

La solución para el conjunto de ecuaciones se puede obtener en forma de determinante mediante la regla de Cramer, como:

$$A=\begin{vmatrix}1 & y_1 & z_1\\ 1 & y_2 & z_2\\ 1 & y_3 & z_3\end{vmatrix} \qquad B=\begin{vmatrix}x_1 & 1 & z_1\\ x_2 & 1 & z_2\\ x_3 & 1 & z_3\end{vmatrix}$$
$$C=\begin{vmatrix}x_1 & y_1 & 1\\ x_2 & y_2 & 1\\ x_3 & y_3 & 1\end{vmatrix} \qquad D=-\begin{vmatrix}x_1 & y_1 & z_1\\ x_2 & y_2 & z_2\\ x_3 & y_3 & z_3\end{vmatrix} \qquad (12\text{-}3)$$

Ya que por lo general manejamos superficies de polígonos que encierran un objeto, necesitamos distinguir entre los dos lados de una superficie. El lado de la faceta que da hacia el interior del objeto se llama **cara interna** y el lado visible o de afuera es la **cara externa**. Así que, por convención, si los vértices del polígono se especifican en una dirección opuesta a las manecillas del reloj cuando se ve el lado externo del plano, la dirección del vector normal será de adentro hacia fuera.

12- 2 Métodos de presentación de superficies

Existen diversas técnicas que nos permiten dar representación a una superficie formada por facetas poligonales. A continuación nombro algunas.

- Método del Búfer Z.
- Sombreado constante.
- Sombreado Gouraud.
- Sombreado Phong.
- Sombreado por rastreo de rayos.

El método del Búfer Z

El planteamiento general del método del **búfer z o búfer de profundidad** es comparar las profundidades de las superficies en cada posición de píxel.

Para aplicar el método del búfer Z requerimos de dos áreas de búfer. Se utiliza el búfer Z para almacenar los valores de profundidad de cada píxel en la posición (x,y) conforme se procesan las superficies. También se requiere del búfer de estructura para almacenar la intensidad de cada píxel en sus posiciones respectivas. Al principio, todas las posiciones en el búfer de profundidad se establecen al valor de máxima profundidad como pudiera ser 65,535 y se inicializa el búfer de estructura a la intensidad del fondo. Luego, se procesa cada superficie que se lista en las tablas de polígonos mediante el método de líneas de rastreo. Cada línea de rastreo arroja una posición de píxel (x,y) y una profundidad z. Entonces, para cada una de estas tríadas de valores, verificamos la siguiente situación: si la profundidad calculada es menor que el valor almacenado, se almacena un nuevo valor para z, se determina la intensidad de la superficie en esa posición y se almacena en el búfer de estructura.

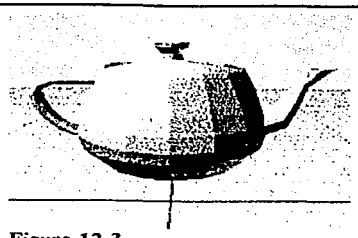


Figura 12-3
Poliedro cuyas facetas se representan con sombreado constante.

Sombreado constante

Un método sencillo y rápido para presentar un objeto con facetas poligonales es el sombreado de intensidad constante, que también se conoce como sombreado plano. En este método se calcula la intensidad de un rayo de luz reflejado en cualquier punto de la faceta poligonal. Todos los puntos sobre la faceta se despliegan entonces con el mismo valor de intensidad. La figura 12-3 nos muestra un poliedro representado con sombreado constante.

Concluyendo, el método de sombreado constante se emplea para desplegar con rapidez la apariencia general de una superficie curva.

Sombreado Gouraud.

Se presenta cada superficie con el sombreado Gouraud al efectuar los cálculos siguientes:

- Se determina el vector normal unitario promedio en cada vértice.
- Se aplican los modelos de iluminación ambiente y difuso en cada vértice para calcular la intensidad de un rayo de luz reflejado en el mismo vértice.
- Se aplica el polígono a la tubería de representación para obtener su proyección 2D.
- Se interpolan de manera lineal las reflexiones de la iluminación incidente a partir de los vértices que definen la superficie de polígono.

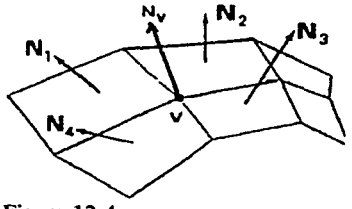


Figura 12-4
Se calcula el vector normal en el vértice V como el promedio de las normales de la superficie de cada polígono que tiene ese vértice.

En cada vértice del polígono, obtenemos un vector normal al promediar las normales de la superficie de todos los polígonos que tienen ese vértice en común, tal como ilustra la figura 12-4. Por tanto para cualquier posición de vértice, obtenemos el vector normal unitario con el cálculo.

$$N_v = \frac{\sum_{k=1}^{n} N_k}{\left|\sum_{k=1}^{n} N_k\right|} \qquad (12\text{-}4)$$

La figura 12-5 muestra el paso siguiente, la interpolación de las intensidades a lo largo de las aristas del polígono. Primero, para cada línea de rastreo calculamos sus intersecciones con las aristas de la faceta poligonal. En el ejemplo de la figura 12-5, los puntos 4 y 5 de la línea de rastreo se calculan como:

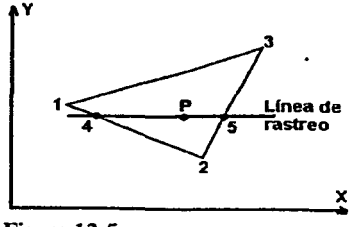


Figura 12-5

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$
$$I_5 = \frac{y_5 - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_5}{y_3 - y_2} I_2 \qquad (12\text{-}5)$$

Una vez que se establecen las intensidades en los extremos de las líneas de rastreo, un punto interior (como el punto p de la figura 12-5) se interpola de las intensidades de las fronteras en los puntos 4 y 5 como

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5 \qquad (15\text{-}6)$$

Concluyendo, la figura 12-6 nos muestra un ejemplo de un poliedro representado con sombreado Gouraud ¿nota la diferencia? Bueno, no todo lo que brilla es oro, el sombreado Gouraud elimina las discontinuidades de intensidad que se asocian con el modelo de sombreado constante, pero tiene otras deficiencias. Los toques de luz en las superficies a veces se despliegan con formas anormales y la interpolación de intensidad lineal puede provocar que en las superficies aparezcan bandas contiguas con cambios notorios de intensidad.

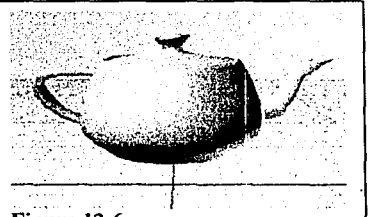


Figura 12-6
Tetera representada empleando sombreado Gouraud.

Sombreado Phong modificado

El método original fue desarrollado por Phong Bui Tuong por lo que se le llamó sombreado de Phong y consiste en interpolar los vectores normales en cada punto de la superficie del polígono y entonces aplicar modelos de iluminación ambiente, difuso y especular. Como consecuencia, se pueden desplegar toques de luz más realistas en las superficies. La figura 12-15 ilustra este concepto.

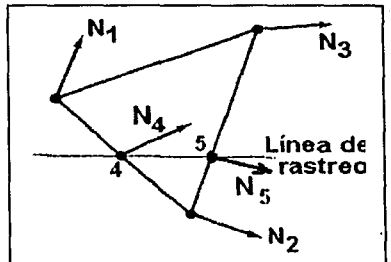


Figura 12-7

Debido a que en los procesadores gráficos se han desarrollado las capacidades de representar grandes cantidades de triángulos, el sombreado Phong se implementa como una variante del sombreado Gouraud. Veamos entonces los pasos para representar una imagen con sombreado Phong:

- Cada polígono se divide en polígonos más pequeños.
- Se determina el vector normal unitario promedio en cada vértice.
- Se aplican los modelos de iluminación ambiente, difuso y especular en cada vértice para calcular la intensidad de un rayo de luz reflejado en el mismo vértice.
- Se aplica el polígono a la tubería de representación para obtener su proyección 2D.
- Se interpolan de manera lineal las reflexiones de la iluminación incidente en cada punto de la superficie 2D a partir de los vértices que definen la superficie de polígono.

Si compara estos pasos con los del sombreado Gouraud, notará dos diferencias fundamentales. La primera implica que cada polígono se divide en polígonos más pequeños con lo que se eliminan las bandas contiguas con cambios notorios de intensidad. La otra diferencia radica en la aplicación del modelo de iluminación especular por lo que despliegan toques de luz más realistas en las superficies. La figura 12-8 nos muestra un ejemplo de lo que se puede lograr con el sombreado de Phong.

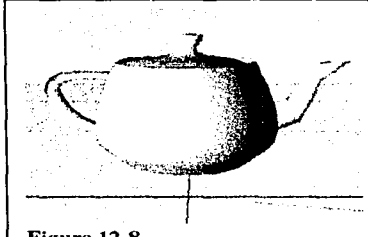


Figura 12-8
Tetera representada empleando el sombreado Phong.

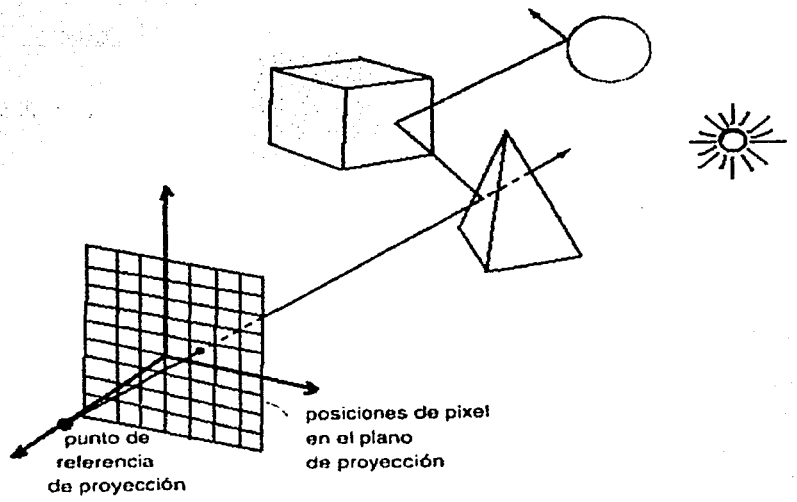


Figura 12-9
Rastreo de un rayo desde el punto de referencia de proyección a través de una posición de píxel con múltiples reflexiones y transmisiones.

12-3 Sombreado por rastreo de rayos

Primero se establece un sistema de coordenadas de píxel designadas en el plano xy. La descripción de la escena se da en este marco de referencia (figura 12-9). A partir del centro de la proyección, se determina entonces la trayectoria de un rayo que pasa a través del centro de cada posición de píxel en la pantalla.

Para cada rayo de píxel, probamos cada superficie en la escena para determinar si se interceptan con el rayo. Si una superficie hace intersección, calculamos la distancia desde el píxel hasta el punto de intersección con la superficie. La distancia de intersección más pequeña que se calcula identifica la superficie visible para ese píxel. Entonces reflejamos el rayo de la superficie a lo largo de una trayectoria especular (el ángulo de reflexión equivale al ángulo de incidencia). Si la superficie es transparente, también enviamos un rayo a través de la superficie en la dirección de refracción. Los rayos de reflexión y refracción se conocen como rayos secundarios.

Este procedimiento se repite para cada rayo secundario: se prueban los objetos con respecto de la intersección y se utiliza la superficie más cercana a lo largo de la trayectoria del rayo para producir de manera recursiva la siguiente generación de trayectorias de reflexión y refracción. Como se puede apreciar en la figura 12-10, conforme los rayos de un píxel rebotan a través de la escena, se agrega cada superficie que se intercepta en forma sucesiva a un árbol binario de rastreo de rayos. Utilizamos las ramificaciones del lado izquierdo del árbol para representar trayectorias de refracción y las del lado derecho para representar trayectorias de transmisión. La profundidad máxima de los árboles de rastreo de rayos se puede establecer como una opción para el usuario o determinarse por la cantidad de almacenamiento disponible. Así, una trayectoria en el

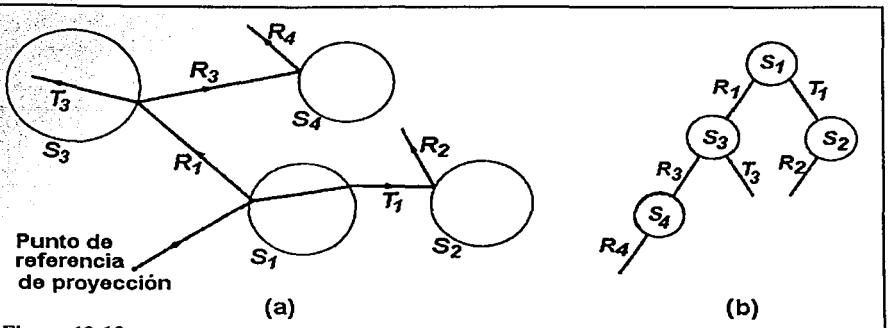


Figura 12-10
Trayectorias de rayos (a) reflexión y refracción a través de una escena para un píxel de la pantalla (b) Árbol de rastreo de rayos binario para las trayectorias que aparecen en la parte (a).

árbol se determina si alcanza el valor máximo preestablecido o si el rayo choca contra una fuente de luz.

La intensidad que se asigna a un píxel se determina entonces al sumar las contribuciones de intensidad, empezando en la parte inferior (nodos terminales) del árbol de rastreo de rayos. Finalmente, debe considerar que la intensidad de cada nodo se atenúa con la distancia entre las superficies.

Si ninguna superficie se intercepta con el rayo, se agrega un nodo con la intensidad asignada al fondo. Si un rayo intercepta una fuente de luz, se crea un nodo que contenga la intensidad de la fuente.

Cortesía de Rhino

Capítulo 13 Superficies de spline

Las superficies de spline son ampliamente empleadas en el modelado de superficies de cuerpos. Algunos ejemplos los podemos ver en el dibujo que sirve de portada al presente capítulo. En general, nosotros podemos modelar animales, rostros humanos, criaturas del espacio, etc.

Para simplificar un poco la idea de las superficies de spline, éstas son una extensión de las curvas de spline. En una curva de spline, aunque se define en un mundo tridimensional, debido al carácter paramétrico de las ecuaciones que la generan, podemos pensar que la misma curva es unidimensional. Una superficie pasa por una situación semejante; un conjunto de ecuaciones paramétricas la definen en R^3 , sin embargo, se trata de un espacio bidimensional ya que se requiere de dos parámetros para generar esta curva, lo que a su vez indica que no ocupa un volumen.

Finalmente, hay dos razones por las cuales, el tema de superficies de spline no se desarrolla a totalidad:

- No se emplearon las superficies de spline para generar la bandera.
- El tema de superficies de spline, sobre todo las superficies NURBS, implica escribir otra tesis.

En consecuencia, el tema de superficies spline será un somero, no obstante su importancia es tal que se les requiere para una vista del diseño de algún un objeto, esto es, un edificio, un zapato, una navaja, etc.

13-1 Superficies B-Spline

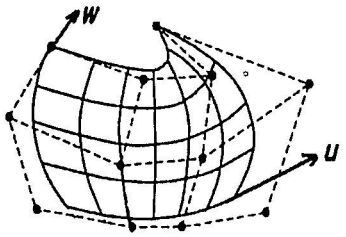


Figura 13-1
Superficie de spline para m=3 y n=4. Figura de Donald Hearn, M. Pauline Baker, "*Gráficas por Computadora*", Prentice Hall, 2° edición, 1995, pg. 350.

Las superficies B-spline son más fáciles de entender que las NURBS, no obstante, las ecuaciones que las representan guardan cierta semejanza. Una superficie B-spline, como la mostrada en la figura 13-1, se representa matemáticamente con la ecuación siguiente:

$$Q(u,w)=\sum_{i=0}^{n+1}\sum_{j=0}^{m+1}B_{i,j}\ N_{d,i}(u)\,M_{e,j}(w) \qquad d\in[2,n+1];\quad e\in[2,m+1] \qquad \text{... (13-1)}$$

Donde:

$B_{i,j}$	Es una matriz con los puntos de control que definen la forma de la superficie.
$N_{d,i}$	Es el conjunto de funciones de combinación en la dirección paramétrica de *u*.
$M_{e,j}$	Es el conjunto de funciones de combinación en la dirección paramétrica de *w*.
u	Siguiendo la figura 13-1, *u*, representa la longitud.
w	Siguiendo la figura 13-1, *w*, representa la latitud.

Ambas funciones de combinación *N* y *M* junto los vectores de nudo que les corresponden, se generan de la misma forma que una curva de spline, según ya vimos en el capítulo 8.

Figura 13-2
Curva de Hermite. La forma de la curva se controla mediante dos vectores tangentes a la misma en los puntos de control.

13-2 Superficie de Hermite

Una curva de Hermite es como la mostrada en la figura 13-2. Note que podemos controlar la forma de curva modificando las dos pendientes en los extremos de la misma. Una superficie de Hermite hereda esta característica. La figura 13-3 nos muestra un ejemplo de tales superficies. Note, en particular, que tenemos tres vectores por cada punto de control; dos vectores que representan las pendientes de la curva en las direcciones paramétricas *u* y *w*. El tercer vector que representa la pendiente de la superficie en las direcciones *u* y *w* simultáneamente.

Matemáticamente, una superficie de Hermite queda representada por medio de la siguiente ecuación:

$$Q(u,w)=[U][N][P][N]^T[W] \qquad (13\text{-}2)$$

donde:

$$[U]=\begin{bmatrix}u^3 & u^2 & u & 1\end{bmatrix}$$

$$[W]^T=\begin{bmatrix}w^3 & w^2 & w & 1\end{bmatrix}$$

$$[N]=\begin{bmatrix}2 & -2 & 1 & 1\\ -3 & 3 & -2 & -1\\ 0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\end{bmatrix};$$ Esta es la matriz de Hermite

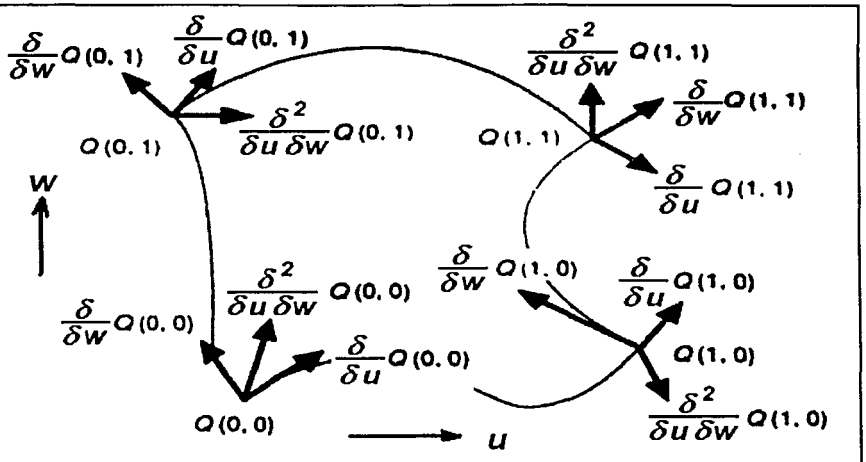


Figura 13-3
Componentes de la matriz geométrica [P] de Hermite. Foley, van Dam, "Computer Graphics, Principles and Practice", Addison Wesley, 2° edición, 1990, pg. 520.

Ahora falta la matriz $[P]$. Esta matriz contiene toda la información geométrica requerida para generar la superficie de Hermite. Esta matriz siempre se subdivide en cuatro matrices de 2x2 ubicadas en las esquinas.

$$[P]=\begin{bmatrix} Q(0,0) & Q(0,1) & \frac{\partial}{\partial w}Q(0,0) & \frac{\partial}{\partial w}Q(0,1) \\ Q(1,0) & Q(1,1) & \frac{\partial}{\partial w}Q(1,0) & \frac{\partial}{\partial w}Q(1,1) \\ \frac{\partial}{\partial u}Q(0,0) & \frac{\partial}{\partial u}Q(0,1) & \frac{\partial^2}{\partial u\partial w}Q(0,0) & \frac{\partial^2}{\partial u\partial w}Q(0,1) \\ \frac{\partial}{\partial u}Q(1,0) & \frac{\partial}{\partial u}Q(1,1) & \frac{\partial^2}{\partial u\partial w}Q(1,0) & \frac{\partial^2}{\partial u\partial w}Q(1,1) \end{bmatrix} \quad \text{... (13-3)}$$

La submatriz de 2x2 en la esquina superior izquierda representa las coordenadas de las cuatro esquinas que forman a una superficie cuadrilateral de Hermite. La submatriz de 2x2 en la esquina superior derecha representa vectores tangentes (derivadas parciales) en las cuatro esquinas en la dirección paramétrica de *w*. La submatriz de 2x2 en la esquina inferior izquierda representa vectores tangentes (también son derivadas parciales) en las cuatro esquinas en la dirección paramétrica de *u*. La submatriz en la esquina inferior derecha son las cuatro derivadas parciales de la superficie en las cuatro esquinas respecto a *u* y *w*.

13-3 Concluyendo

Hasta aquí el tema de superficies de spline, aunque debo admitir que aún falta por mencionar las superficies de Bezier, tan importantes en el dibujo mecánico, y las superficies Coons, que son facetas cuadrilaterales cuyas fronteras son curvas de spline y que son importantes en el modelado de objetos.

Cortesía de McGraw Hill.

Capítulo 14 Introducción al modelo de iluminación de radiosidad

En el capítulo 11 revisamos los tres modelos empíricos de iluminación: ambiente, difuso y especular. Adicionalmente agregamos algunos refinamientos como el modelo cuadrático de atenuación que es representado por la ecuación 14-1. De no incluir este refinamiento, superficies a distintas distancias se verían igualmente iluminadas, así que la sensación de profundidad se pierde. Estos modelos empíricos tienen una severa desventaja, las fuentes de luz solo pueden ser puntuales, sin embargo, mediante el modelo cuadrático de atenuación es posible disimular esa situación.

$$f(d) = \frac{1}{a_0 + a_1 d + a_2 d^2} \qquad (14\text{-}1)$$

A diferencia, *el modelo de iluminación de radiosidad permite modelar con precisión reflexiones difusas en las superficies*. Adicionalmente, este modelo si *permite modelar fuentes de luz que no son puntuales*, así que podemos modelar focos de neón y sus efectos de iluminación en una escena. Por supuesto, *sabemos que a mayor realismo, menor velocidad en la representación de una escena*, lo que significa una gran cantidad de cálculos y administración de información.

El modelo básico de radiosidad parte de las siguientes formulaciones:

- Se trabaja con una escena en la que las superficies de los cuerpos están formadas por parches o facetas, es decir, superficies poligonales.
- La energía radiante que fluye de una faceta se distribuye entre todas las demás facetas.
- Una faceta puede ser emisora de energía radiante y puede ser también una superficie reflejante.
- Por el punto anterior, la energía radiante que fluye de una faceta es la suma de la energía que emite como fuente de luz y de la energía que refleja.

Debido a que el modelo de radiosidad no se implementa en la API OpenGL, solo veremos las ecuaciones correspondientes al modelo. Por lo que respecta al algoritmo, queda al lector la tarea de su desarrollo.

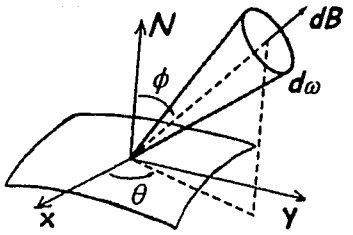


Figura 14-1
Ángulo sólido que representa energía radiante emanando por una superficie, ya sea emisora o reflectora de energía radiante. Figura obtenida de Donald Hearn, M. Pauline Baker, "*Gráficas por Computadora*", Prentice may, 2° edición, 1995, pg. 574.

14-1 Conceptos físicos

Como primer paso al planteamiento del modelo debemos repasar algunas ecuaciones y conceptos básicos de Óptica que requeriremos más adelante.

Densidad de potencia e intensidad luminosa de la fuente

Empezamos recordando el concepto de densidad de potencia. Como ya sabemos, la energía luminosa que emana de la fuente transporta energía, así que definimos *el transporte de energía, en la unidad de tiempo, que atraviesa normalmente una unidad de superficie*, como la **densidad de potencia** y la designamos con la letra *B*. Las unidades que definen la densidad de potencia son *watt/m*^{2}.

Observe ahora la figura 14-1 y permítame definir la **intensidad luminosa de la fuente** como la *energía radiante que fluye a través de la unidad de ángulo sólido, en la unidad de tiempo*. Como consecuencia de emplear la unidad de ángulo sólido en la definición anterior, la superficie que atraviesa esta energía radiante es también unitaria y normal a la dirección de propagación. Las unidades que le corresponden son *watt/(m*^{2}*·esterorradián)*.

Es posible conocer la intensidad de la fuente a partir de la densidad de potencia si realizamos la siguiente operación:

$$I = \frac{B}{\omega} \qquad (14\text{-}2)$$

donde I : es la densidad luminosa de la fuente
B : densidad de potencia
ω : ángulo sólido

Es mas conveniente expresar la ecuación 14-2 en forma diferencial como:

$$I = \frac{dB}{d\omega} \qquad (14\text{-}3)$$

Intensidad luminosa y densidad de potencia proyectada

Suponga que ahora tenemos una fuente de luz iluminando una superficie "A", según se muestra en la figura 14-2.a. Si ahora la fuente de luz ilumina una superficie "B" tal como se muestra en 14-2.b, observamos que, debido a la inclinación del ángulo sólido, la densidad de potencia incidente se reduce. Otra forma de ver la situación implica considerar el aumento en la superficie de incidencia.

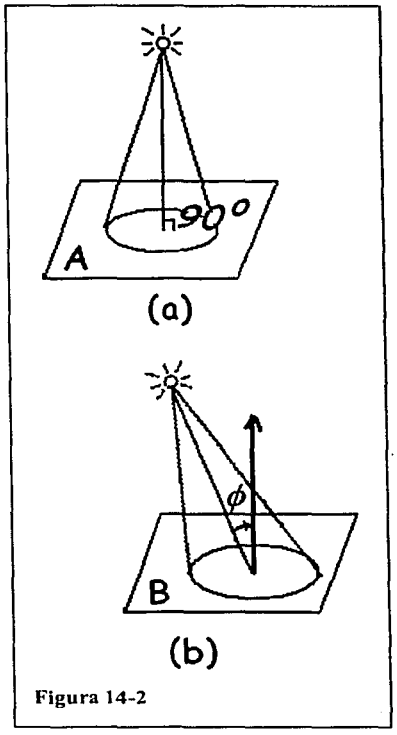


Figura 14-2

Para hallar un área normal a la trayectoria del flujo de energía que sea equivalente al área iluminada pensamos en la siguiente ecuación:

$$B_{equivalente} = B_{Iluminada} \cos\phi \qquad (14\text{-}3)$$

Si deseamos calcular la intensidad luminosa de la fuente a partir de la densidad de potencia proyectada en una superficie lo hacemos como:

$$I = \frac{dB}{d\omega} Cos\phi \qquad (14\text{-}4)$$

14-2 Ecuaciones del modelo de radiosidad

Previamente debemos considerar a una frontera cerrada que abarque a todas las superficies implicadas en la escena. Con esto evitamos que fuentes desconocidas proporcionen energía a la misma. Toda la energía radiante se distribuye en cada superficie encerrada por la frontera.

Ecuación de radiosidad

Consideremos ahora las siguientes situaciones:

- La energía radiante que fluye de una faceta se distribuye entre todas las demás facetas.
- Una faceta puede ser emisora de energía radiante o fuente de luz y puede ser también una superficie reflejante. Por lo tanto, la energía radiante que fluye de una faceta es la suma de la energía que emite como fuente de luz y de la energía que refleja.

Respecto del primer punto, designamos el parámetro de radiosidad B_j como la cantidad total de energía que emana de una superficie j por unidad de área por unidad de tiempo. Definimos también el parámetro H_k como la suma de las contribuciones de energía de todas las superficies comprendidas en la escena y que llegan a la superficie k por unidad de área por unidad de tiempo. Este párrafo se expresa en forma matemática como:

$$H_k = \sum_j B_j F_{j,k} \qquad (14\text{-}5)$$

Donde F_{jk} es el **factor de forma** para las superficies j y k. Acaso no amamos cuando alguien nos resuelve una incógnita con otra incógnita ¿qué significa el factor de forma? El factor de forma es la *cantidad fraccional de energía radiante de la superficie j que llega a la superficie k*. El factor de forma no es una variable definida por el usuario, de hecho,

está en función de la orientación espacial de las superficies j y k o bien, en función de la magnitud del área incidente. Así que el factor de forma debe calcularse. Más adelante veremos como calcular este parámetro.

Respecto del segundo punto, para una escena con n superficies dentro de ella, la energía radiante que diverge de la superficie k se describe en la ecuación siguiente 14-6 y que llamaremos ecuación de radiosidad.

$$B_k = E_k + \rho_k H_k \tag{14-6}$$

Veamos, E_k es la energía que se emite de la superficie en la unidad de tiempo por unidad de área. Fíjese bien, "emite", es decir, se trata de la superficie de una fuente de luz por emisión. Así que, si la superficie k no es una fuente de luz entonces $E_k = 0$. El parámetro ρ_k es el factor de reflectividad para la superficie, o sea, es el porcentaje de luz incidente que se refleja en todas direcciones. Este factor puede verse de forma análoga como el coeficiente de reflexión difusa que se utiliza en los modelos empíricos de iluminación.

Reacomodemos los términos de la ecuación 14-6 para calcular la densidad de potencia que se emite desde una fuente de luz.

$$B_k - \rho_k H_k = E_k \tag{14-7}$$

Sustituyendo la ecuación 14-15 en la ecuación 14-7, tenemos

$$B_k - \rho_k \sum_{j=1}^{n} B_j F_{j,k} = E_k \tag{14-8}$$

La ecuación 14-8 debe aplicarse a cada faceta dentro de la escena, por lo que tendremos un sistema de ecuaciones como en 14-9.

$$\begin{bmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{14-9}$$

Donde:

Los términos $[E_1 \ E_2 \ \ldots \ E_n]$ corresponden a la densidad de potencia que emite la fuente de luz, pero ¿qué no las fuentes de luz se especifican por su intensidad luminosa? Así es, para convertir el dato de intensidad luminosa a densidad de potencia recurrimos a la ecuación 14-10

$$E = \frac{I}{\pi} \tag{14-10}$$

El vector $[B_1 \ B_2 \ .. \ B_n]^T$ implica las densidades de potencia que se emiten desde las facetas y también es nuestro vector de incógnitas.

En cuanto a los factores de forma, ahora vemos como se calculan

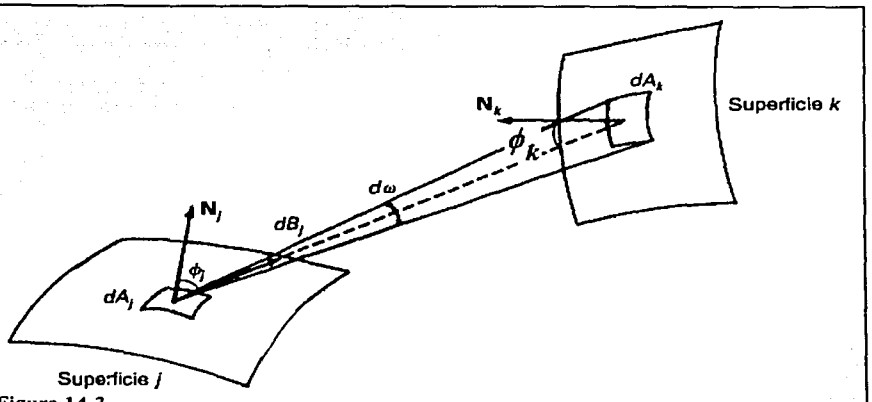


Figura 14-3
Índice de transferencia de energía dB_j de un elemento de superficie con área dA_j, al elemento de superficie dA_k. Figura obtenida de Donald Hearn, M. Pauline Baker, "*Gráficas por Computadora*", Prentice may, 2° edición, 1995, pg. 577.

Cálculo del factor de forma

El factor de forma entre dos facetas de superficies A_j y A_k, siendo j la superficie emisora de energía radiante y k la superficie de incidencia, se puede definir así.

$$F_{Aj,Ak} = \frac{Energía\ incidente\ sobre A_k}{Energía\ total\ que\ abandona\ Aj} \qquad (14\text{-}11)$$

La figura 14-3 nos ilustra mejor la situación: tenemos energía radiante que emana del segmento de superficie dA_j incidiendo en el segmento de superficie dA_k. Ya se imaginará que debemos recurrir al cálculo integral para resolver la ecuación (14-11).

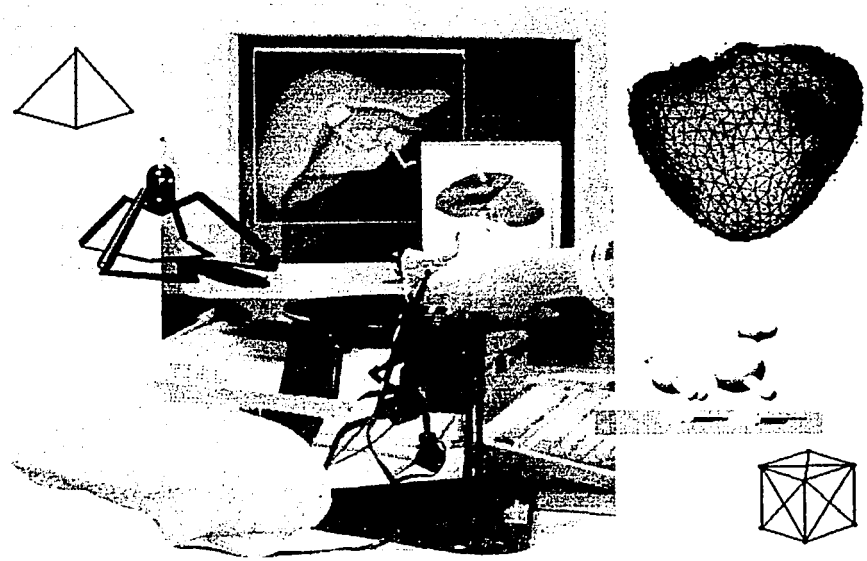
En resumen, el coeficiente de forma se calcula mejor mediante la ecuación 14-12

$$F_{jk} \frac{1}{A_j} \int_{Superficiej} \int_{Superficiek} \frac{\cos\phi_k}{\pi r^2} dA_k dA_j \qquad (14\text{-}12)$$

Estas dos integrales se evalúan al utilizar técnicas de integración numérica y establecer las condiciones siguientes:

- $\sum_{k=1}^{n} F_{j,k} = 1$ Para toda k (conservación de la energía).

Finalmente, es posible subdividir cada superficie en la escena en muchos polígonos pequeños y cuanto más pequeñas son las áreas de los polígonos, más realista es la apariencia del despliegue.



Cortesía de Inria

Capítulo 15 El modelo de movimiento

Vamos a partir del siguiente supuesto: toda estructura como edificios, sillas, gelatinas, pedazos de tela, láminas de metal y demás, pueden ser modelados con partículas de masa *m* que se enlazan con resortes y amortiguadores de tal forma que éstos forman una red. Para ilustrar, la figura 15-1.a nos muestra cuatro masas enlazadas por cuatro sistemas resorte-amortiguador. Las conexiones de cada enlace con la partícula son flexibles en este caso. La figura 15-1.b nos muestra una mesa y una silla cuyas deformaciones pueden modelarse mediante el concepto de partículas que acabo de describir.

Es necesario mostrar que esta forma de modelar cuerpos blandos no es una teoría aislada en mi mente, si no que tiene aceptación en empresas de investigación médica, ingeniería civil, industria cinematográfica y otras. La figura 15-2.a muestra varias escenas de un riñón presionado en varios puntos por un instrumento. El modelo se ejecuta en un equipo Origin 2000[1]. Tal modelo fue realizado conjuntamente por los equipos de investigación Imagis y Epidaure, mismos que pertenecen al grupo INRIA[2]. Según los investigadores de INRIA, un riñón puede simularse a partir de una red cúbica de masas y enlaces resorte-amortiguador. En particular, este diseño tiene la capacidad de modificar el volumen que ocupa cada partícula durante las interacciones con el fin de reducir los cálculos. El hecho de modificar el volumen no significa pérdida o ganancia de masa. La figura 15-2.b nos

[1] Origin 2000 es una marca registrada de Silicon Graphics Inc.

[2] INRIA se divide en grupos de investigación. Cada grupo cuenta con un presupuesto que los integrantes del mismo administran. Los grupos Epidaure e Imagis realizan la integración de la medicina con la síntesis de imágenes.

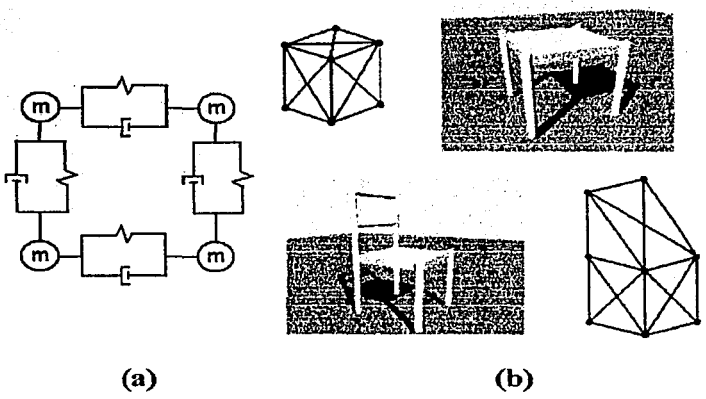


Figura 15-1
Existe la teoría de que cualquier objeto físico, cuya forma puede ser modificada, puede modelarse empleando un sistema masa-resorte-amortiguador. En (a) se muestran cuatro masas enlazadas por parejas resorte-amortiguador. En (b) se muestran ejemplos de aplicación de esta teoría. Las imágenes en (b) son cortesía del grupo INRIA. Harvard, "*Managing Desing Complexity: Using Stochastic in the Production of Computer Graphics*", John Chrsitensen, Junio 1995.

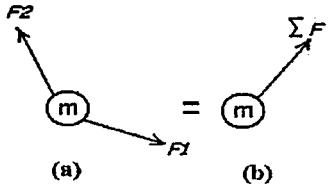


Figura 15-3
Dado un sistema de fuerzas actuando sobre una masa, el sistema puede ser remplazado por una sola fuerza equivalente. Ferdinand. P. Beer, E. Russell Johnston Jr. "Mecánica Vectorial para Ingenieros", McGraw Hill, 5° ed., 1990.

muestra un corte transversal del modelo en diversas fases de su construcción con estas partículas de volumen variable.

15-1 Ecuaciones básicas de movimiento: Segunda ley de Newton, el resorte y el amortiguador

Considere a una partícula de masa m sujeta a varias fuerzas, tal como se ve en la figura 15-3.a. Si aplicamos la segunda ley de Newton a este sistema masa-fuerzas, tenemos la ecuación 15-1, misma que se representa en la figura 15-3.b.

$$\sum F = ma \qquad (15\text{-}1)$$

Sin embargo, para resolver problemas en los que interviene el movimiento de la partícula, se encontrará más conveniente sustituir la ecuación 15-1 por ecuaciones equivalentes en términos de los componentes rectangulares los cuales, a su vez, vienen a ser cantidades escalares. Descomponiendo ahora cada fuerza y la aceleración en términos de las cantidades rectangulares.

$$\sum F_x\hat{i} + F_y\hat{j} + F_z\hat{k} = m\left(a_x\hat{i} + a_y\hat{j} + a_z\hat{k}\right) \qquad (15\text{-}2)$$

de donde se sigue que

$$\sum F_x = ma_x\ ;\quad \sum F_y = ma_y\ ;\quad \sum F_z = ma_z\ ; \qquad (15\text{-}3)$$

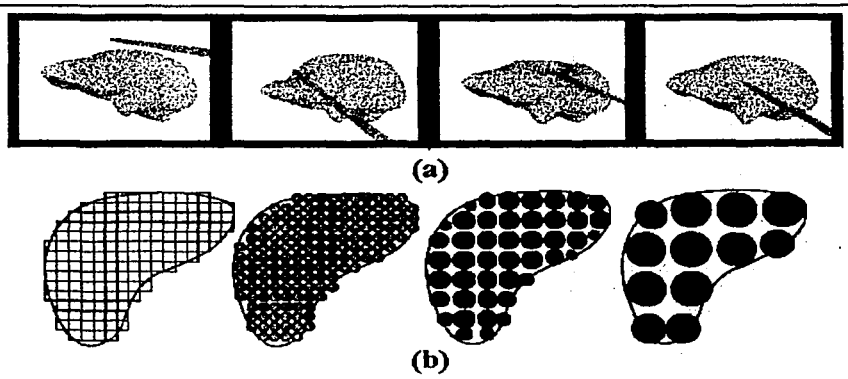


Figura 15-2
En (a) se muestra el resultado de la simulación de un riñón empleando la técnica masa-resorte-amortiguador. La técnica implica conceptos de deformación de cuerpos de masa constante y volumen variable. Esta última características se implementa para reducir los cálculos. En (b) se puede observar como se acomodan las masas en el interior del riñón. Las figuras fueron obtenidas de Gilles Rebunne, Mathieu Desbrun, Alan Barr, Marie-Paule Cani, "*Interactive Resolution Animation of Deformable Models*", iMagis, 1999.

De las ecuaciones anteriores, se puede notar que estamos empleando un sistema de referencia rectangular, el cual, aunque tal vez no sea el más propicio para resolver un sistema masa-resorte-amortiguador, nos proporciona directamente el tipo de datos adecuado para OpenGL.

El resorte

El resorte se representa con el símbolo mostrado en la figura 15-4.a. Las variables asociadas con este elemento son fuerza, desplazamiento y longitud de reposo. Su comportamiento físico está definido por:

$$f_k = -K(x - \lambda) \qquad (15\text{-}4)$$

donde

- f_k : Es la fuerza producida por el resorte, se opone al movimiento de éste y es igual en magnitud y opuesto a la fuerza externa aplicada (f) con base en la tercera ley de Newton. Las unidades con las que medimos la fuerza son newtons.
- λ : Es la longitud del resorte en su estado de reposo. La longitud la medimos en metros.
- x : Es la longitud actual del resorte.
- K : Es la constante del resorte y se mide en newton/metro.

El resorte tiene una curva característica fuerza-longitud que es lineal y que se muestra en la figura 15-4.b

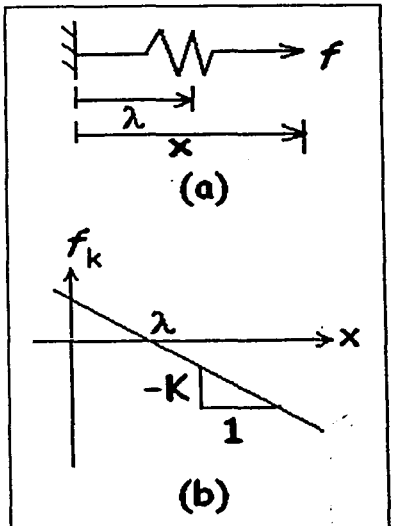
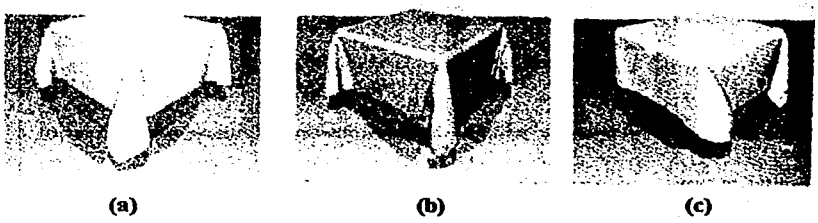


Figura 15-4
En (a) tenemos la representación de un resorte. Puede notar la indicación de una longitud de reposo (lambda) y una fuerza externa alargándolo. En (b) tenemos la curva característica del mismo. Note que empleamos un modelo lineal.



(a) (b) (c)

Figura 15-6
Modelado de las características de (a) algodón, (b) lana y (c) poliéster. Los comportamientos característicos de cada material se logran al modificar convenientemente las masas y constantes de resortes y amortiguadores. Figura de Donald Hearn, M. Pauline Baker, "*Gráficas por Computadora*", Prentice Hall, 2° ed., 1995, pg. 414.

El amortiguador

El amortiguador se representa con el símbolo mostrado en la figura 15-5.a. las variables asociadas con este elemento son fuerza y rapidez. Su comportamiento físico está definido por:

$$f_B = -Bv \qquad (15\text{-}5)$$

donde:

- f_B : Es la fuerza que producida por el amortiguador, se opone al movimiento de éste y es igual en magnitud y opuesto a la fuerza externa aplicada (f) con base en la tercera ley de newton. La fuerza se mide en newtons.
- v : Es la velocidad a la cual se mueve el émbolo dentro del medio viscoso. La velocidad se mide en metros por segundo.
- B : Es el valor de la constante del amortiguador. Se mide en newton-metro/segundo.

El amortiguador tiene una curva característica fuerza-velocidad que es lineal y que se muestra en la figura 15-5.b.

¿Cómo funciona el amortiguador? La respuesta es simple, se trata de un émbolo moviéndose dentro de un fluido. A consecuencia, el émbolo experimenta los efectos de una fuerza de fricción, misma que llamamos **fricción viscosa**.

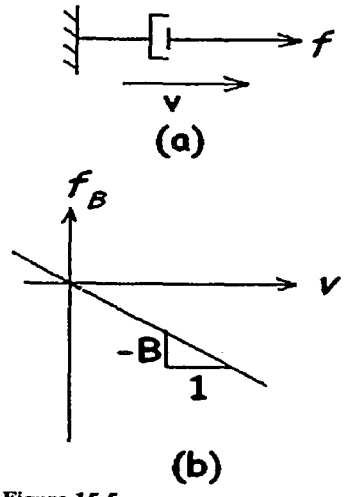


Figura 15-5
En (a) se tiene la representación de un amortiguador. Puede notar la indicación de la velocidad del émbolo en el medio viscoso debido a la fuerza externa que lo mueve. En (b) tenemos la curva característica del mismo. Note que empleamos un modelo lineal.

15-2 Partículas gigantes

Toda estructura como edificios, sillas, gelatinas, pedazos de tela, láminas de metal y demás, pueden ser modelados con partículas de masa m que se enlazan con resortes y amortiguadores, de tal forma que éstos formen una red

Siguiendo lo indicado en el párrafo anterior vamos a tratar de construir una bandera que en general es una lámina de tela. Así, diversos tipos de tela pueden ser simulados fijando las propiedades correctas de cada masa, resorte y amortiguador. La figura 15-6 nos muestra tres tipos de tela: algodón, lana y poliéster. Cada material se aproxima

minimizando la constante del resorte en cada enlace.

Partícula gigante

Habiendo considerado la teoría anterior, podemos definir entonces una **partícula gigante** *como una masa m de superficie esférica.* Adicionalmente se cumple el principio de que dos masas no pueden ocupar el mismo espacio. Las partículas gigantes pueden interactuar libremente, unas con otras, o bien, a través de enlaces.

Cuando dos partículas colisionan, el choque se puede definir en elástico, o sea, partículas que rebotan y en plástico, en el que buena parte de la energía se disipa en el choque y es tal que ambas partículas permanecen juntas.

Decimos que estas partículas son gigantes debido a que una computadora casera o incluso una estación de trabajo, no tienen suficiente poder para simular cientos o quizá miles de partículas interactuando a través de sus enlaces en tiempo real. De tal forma, haciendo que cada partícula abarque un mayor volumen reducimos considerablemente la cantidad de cálculos.

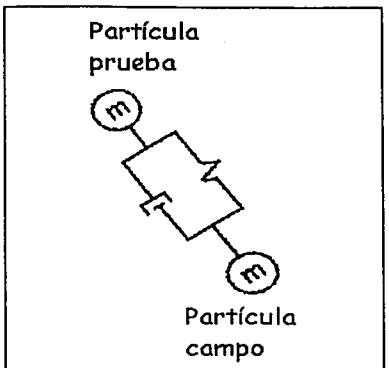


Figura 15-7
Partículas de prueba y de campo enlazadas por sistemas resorte-amortiguador.

Partícula de campo y partícula de prueba

Vamos ahora a definir un par de conceptos que suelen emplearse cuando se trabaja con partículas: partícula de campo y partícula de prueba.

Considere entonces un sistema de referencia newtoniano[3] en el que existe un campo X. Sabemos que existe este campo porque si colocamos una **partícula de prueba** en éste, la misma se verá afectada por una fuerza. Como una condición propuesta, la partícula se verá afectada sin importar su estado de movimiento o reposo. Ahora ¿quién produce este campo X? Vamos a definir que es otra segunda partícula la que produce el campo. En consecuencia a esta segunda partícula la llamaremos **partícula de campo**.

Sería agradable implicar conceptos como trabajo y energía potencial en nuestra investigación, sin embargo, como nuestro objetivo es animar una bandera, lo cual es un caso particular, podemos prescindir de estos conceptos para simplificar la longitud del escrito y así, el número de cálculos necesarios.

En general, cuando tratemos dos partículas unidas por un enlace, optaremos por una de ellas para que sea la partícula de campo y la otra, la partícula de prueba. La figura 15-7 ilustra el concepto.

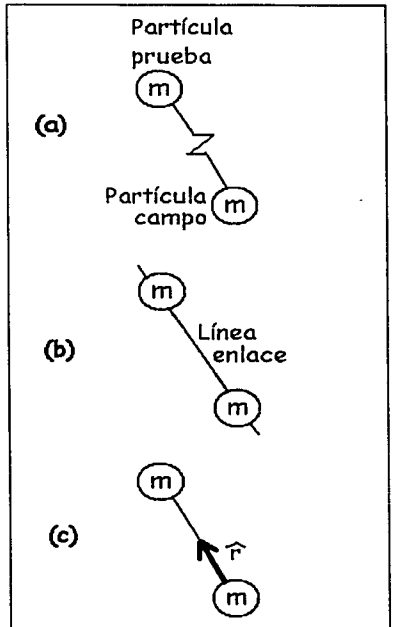


Figura 15-8
Enlace inductivo entre dos partículas.

15-3 Enlace inductivo (resorte) entre dos partículas

La figura 15-8.a nos muestra dos partículas unidas por un resorte. A este tipo de enlace le conocemos como enlace inductivo por su analogía con los circuitos eléctricos. El enlace inductivo actúa a lo largo de la recta que pasa por los centros de masa de ambas partículas. A esta línea la llamamos **línea de enlace**: figura 15-8.b. Definimos ahora un vector unitario $\hat{r}$ que parte de la partícula de campo y es paralelo a la línea de enlace

[3] Un sistema de referencia Newtoniano o inercial es aquel cuyos ejes de referencia tienen una orientación constante respecto de las estrellas y su origen debe estar, ya sea fijo en el centro de masa del Sol o moverse con velocidad constante respecto de éste.

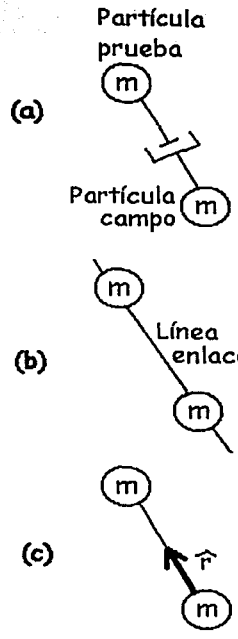


Figura 15-9
Enlace amortiguado entre dos partículas.

(figura 15-8.c). La fuerza del enlace inductivo está en función de un cambio en la longitud del enlace:

$$\begin{matrix}\textit{Cambio de}\\ \textit{longitud}\\ (\Delta L)\end{matrix} = \begin{matrix}\textit{Longitud}\\ \textit{actual}\end{matrix} - \begin{matrix}\textit{Longitud}\\ \textit{de reposo}\\ (\lambda)\end{matrix} \qquad (15\text{-}5)$$

En donde:

$$\begin{matrix}\textit{Longitud de}\\ \textit{reposo}\\ (\lambda)\end{matrix} >= \begin{matrix}\textit{Radio}\\ \textit{partícula}\\ \textit{prueba}\end{matrix} + \begin{matrix}\textit{Radio}\\ \textit{partícula}\\ \textit{campo}\end{matrix} \qquad (15\text{-}6)$$

Ahora podemos definir la fuerza que actúa en la partícula de prueba como

$$\vec{F}_{prueba} = -(k \cdot \Delta L)\hat{r} \qquad (15\text{-}7)$$

El signo negativo en la ecuación indica que la dirección de la fuerza es en contra del vector unitario $\hat{r}$. Por la tercera ley de Newton, la fuerza sobre la partícula de campo es:

$$\vec{F}_{campo} = -\vec{F}_{prueba} \qquad (15\text{-}8)$$

15-4 Enlace amortiguado entre dos partículas

La figura 15-9.a nos muestra dos partículas unidas por un amortiguador. A este tipo de enlace también se le llama enlace resistivo ya que se construye con un émbolo que se mueve en un medio viscoso. El enlace amortiguado actúa a lo largo de la línea recta que pasa por los centros de masa de ambas partículas (figura 15-9.b). A esta línea la llamamos **línea de enlace.** Definimos ahora un vector unitario $\hat{r}$ que parte de la partícula de campo y es paralelo a la línea de enlace (figura 15-9.c)

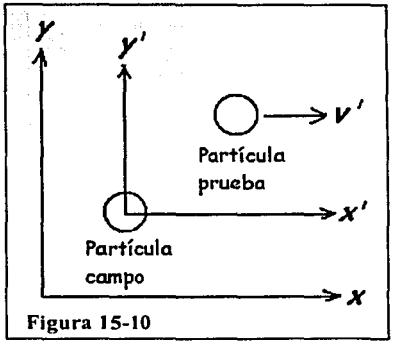


Figura 15-10

Consideremos ahora lo siguiente: la fuerza del enlace amortiguado está en función de la velocidad con la que el enlace cambia de longitud. Para obtener esta velocidad debemos revisar la siguiente situación:

- Una partícula de campo se mueve con velocidad v_{campo} en un sistema de referencia newtoniano xy.
- Una partícula de prueba se mueve con velocidad v_{prueba} en el mismo sistema de referencia.
- Definimos ahora un sistema de referencia $x'y'$ que se mueve junto con la partícula de campo, tal como se ilustra en la figura 15-10.

Habiendo considerado los puntos anteriores, podemos concluir en la siguiente ecuación:

$$v_{prueba} = v_{campo} + v'_{prueba} \qquad (15\text{-}9)$$

De la ecuación anterior despejamos la v'_{prueba} y ¿para qué nos sirve? Esta es la velocidad

de la partícula de prueba cuando la partícula de campo está fija. En consecuencia, v'_{prueba} *es también la velocidad con la cual el enlace cambia de longitud.*

Luego, la fuerza que actúa sobre la partícula de prueba debido al amortiguador es:

$$\vec{F}_{prueba} = -\left(B \cdot v'_{prueba}\right)\hat{r} \qquad (15\text{-}10)$$

El signo negativo indica que la dirección de la fuerza es en contra del vector unitario $\hat{r}$. Por la tercera ley de Newton, la fuerza sobre la partícula de campo es:

$$\vec{F}_{campo} = -\vec{F}_{prueba} \qquad (15\text{-}11)$$

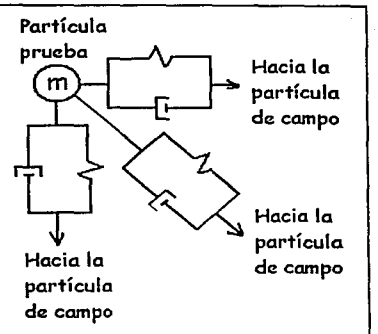


Figura 15-11
Diagrama general de una partícula de prueba con enlaces dedicados. Estos enlaces son del tipo resorte-amortiguador y siempre apuntan a una partícula de campo.

16-5 Partículas de prueba con enlaces

Existen dos formas en las que podemos organizar la información que representa a un sistema de partículas:

a) Con dos bases de datos, una de partículas y otra de enlaces.
b) La otra forma es una base de datos en la que cada partícula tenga definido un número de enlaces y cada uno de ellos con sus propios parámetros.

Ambos sistemas han sido probados previamente a la realización del presente escrito y aunque no lo crea, la opción *b)* ha dado resultados más estables en cuanto al movimiento de partículas. En lo correspondiente a la programación, he obtenido una relativa facilidad al momento de suministrar la información con la que se construye la red de partículas.

La figura 15-11 ilustra el concepto de una partícula de prueba de la cual emergen enlaces resorte-amortiguador. Estos enlaces apuntan siempre a partículas de campo.

El funcionamiento general de un sistema construido con este tipo de partículas es simple: recuerde que las partículas de campo ejercen una fuerza sobre la partícula de prueba. Este sistema también permite que una fuerza externa sea aplicada, por lo que podemos concluir que la ecuación general de movimiento puede escribirse como:

$$\sum Fuerzas\ externas + \sum fuerzas\ enlaces = ma \qquad (15\text{-}12)$$

Las fuerzas por enlaces corresponden a las parejas resorte-amortiguador. Ahora bien, esta ecuación, según puede ver, es la segunda ley de Newton. Parece simple verdad, no obstante, Newton formuló tres leyes de movimiento y de las tres nos hace falta aplicar la tercera, veamos, si una partícula de campo ejerce una fuerza $\vec{F}$ sobre una partícula de prueba, la partícula de prueba ejerce una fuerza $-\vec{F}$ (misma dirección, misma magnitud y sentido opuesto) sobre la partícula de campo. La pregunta ahora es ¿cómo influye la tercer ley de Newton en el sistema? La respuesta la tenemos en la figura 15-12: observe que ahora tenemos una red de partículas enlazadas.

De la figura 15-12 he resaltado la partícula 1. El conjunto de fuerzas que actúan en ella son: las fuerzas externas, las fuerzas que las partículas de campo 3 y 4 ejercen y si observa bien, la partícula 1 es una partícula de campo para la partícula 2, así, siguiendo la tercer ley de Newton, tenemos una fuerza de reacción que la partícula 2 ejerce sobre la

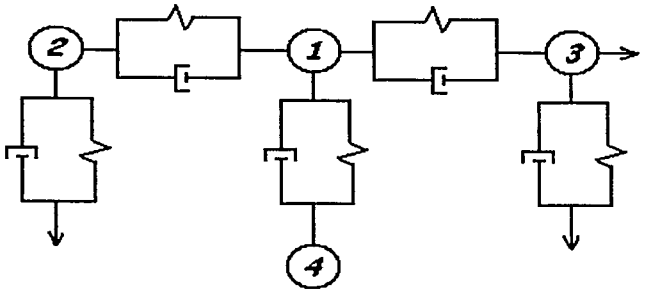


Figura 15-12
Forma en la que se construye una rede partículas mediante el concepto de partícula enlazada. En la figura, una partícula de prueba, rotulada como "1", es sometida a los efectos de otras partículas de campo. A su vez, la partícula "1" es partícula de campo para la partícula de prueba "2".

partícula 1. En resumen, la ecuación de movimiento correspondiente a toda partícula de prueba es:

$$\sum F.externas + \sum F.enlaces + \sum F.reacción = ma \tag{15-13}$$

Esta ecuación representa a una sola partícula, sin embargo, extendiendo la notación podemos obtener un sistema de ecuaciones que represente a todas las n partículas de nuestra red.

$$\sum F.externas_k + \sum F.enlaces_k + \sum F.reacción_k = m_k a_k \tag{15-14}$$

El conjunto de **fuerzas externas** lo constituyen la fuerza de gravedad y la acción del viento que hará ondear a nuestra bandera. A las fuerzas debidas a los enlaces y reacciones que se producen las podemos llamar **fuerzas internas**.

Por ahora, vamos a continuar describiendo las fuerzas que actúan sobre las partículas y será en la última sección cuando describa un método numérico que permita resolver este sistema de ecuaciones y además, nos de a conocer la posición y velocidad de cada partícula en el sistema.

15-6 Impacto entre partículas

Definimos **impacto** como un *choque que ocurre entre dos cuerpos en un intervalo muy pequeño de tiempo y durante el cual los dos cuerpos ejercen entre sí fuerzas relativamente grandes.*

Ya que trabajamos con partículas esféricas, podemos definir la **línea de impacto** como una línea que pasa por los centros de masa de ambas partículas y que es normal a sus superficies. Si los centros de masa de los cuerpos que chocan se localizan sobre esta línea decimos que se trata de un **impacto central**. Si los vectores de velocidad de ambas

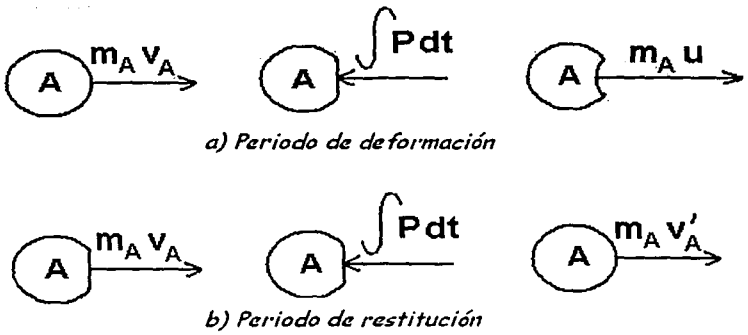


Figura 15-15
(a) Periodo de deformación de dos partículas en colisión. (b) Periodo de restitución de las dos partículas. Figuras de Ferdinand. P. Beer, E. Russell Johnston Jr. "*Mecánica Vectorial para Ingenieros*", McGraw Hill, 5° ed., 1990, tomo II; pg. 641.

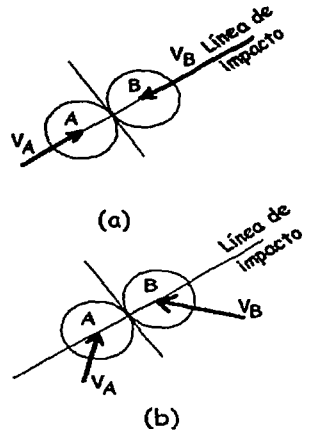


Figura 15-13
La figura ilustra dos conceptos de impacto: (a) impacto central directo y (b) impacto central oblicuo. Figuras de Ferdinand. P. Beer, E. Russell Johnston Jr. "*Mecánica Vectorial para Ingenieros*", McGraw Hill, 5° ed., 1990, pg. 640.

partículas son paralelos a la línea de impacto, se dice que ocurre un **impacto directo** (figura 15-13.a). Por otro lado, si una o ambas partículas se mueven con velocidades que no son paralelas a la línea de impacto, se dice que el **impacto es oblicuo** (figura 15-13.b).

Impacto central directo

Considere dos partículas *A* y *B* de masas m_A y m_B, que se mueven en línea recta hacia la derecha con velocidades v_A y v_B (figura 15-14.a). Si v_A es mayor que v_B, la partícula *A* golpeará finalmente a la *B*. Por el impacto, las dos partículas se deformarán y, al final del periodo de deformación, tendrán la misma velocidad u (figura 15-14.b). Entonces ocurrirá un periodo de restitución, al final del cual, las dos partículas habrán recuperado su forma original o quedarán deformadas permanentemente y esto en función de la magnitud de las velocidades de impacto y de los materiales de que estén hechas. Nuestro propósito ahora es encontrar las velocidades v'_A y v'_B de las partículas al final del periodo de restitución (figura 15-14.c)

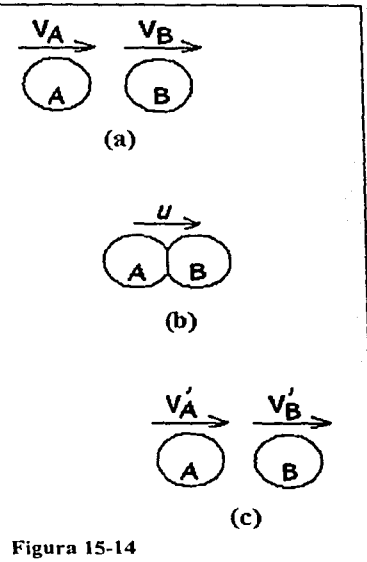


Figura 15-14
(a) Antes del impacto. (b) En su deformación máxima. (c) Después del impacto.

Como primer paso para hallar las velocidades luego del choque, considérese primero a las dos partículas dentro de un sistema cerrado a fin de que no haya fuerza externa impulsiva. Por consiguiente, la cantidad de movimiento lineal total de las dos partículas se conserva y escribimos:

$$m_A v_A + m_B v_B = m_A v'_A + m_B v'_B \qquad (15\text{-}15)$$

De esta ecuación note que tenemos dos incógnitas, por lo que nos falta definir otra ecuación, veamos, como segundo paso observe la figura 15-15.a correspondiente al periodo de deformación de las partículas que chocan. Según la figura, la única fuerza impulsiva que actúa sobre *A* durante este periodo es la fuerza *P* aplicada por *B*, así que escribimos:

$$m_A v_A - \int P dt = m_A u \qquad (15\text{-}16)$$

En la que la integral se extiende al periodo de deformación. Considérese ahora la figura 15-15.b que corresponde al movimiento de A durante el periodo de restitución. Si R es la fuerza aplicada sobre A por B durante este periodo, podemos escribir:

$$m_A u - \int R dt = m_A v'_A \qquad (15\text{-}17)$$

en la que la integral se extiende al periodo de restitución. Un tercer paso para determinar las velocidades luego de la colisión implica definir el **coeficiente de restitución** como el cociente del impulso durante la restitución entre el impulso durante la deformación.

$$e = \frac{\int R dt}{\int P dt} \qquad (15\text{-}18)$$

El valor de este coeficiente es siempre entre 0 y 1 y está en función de los materiales, las velocidades de impacto, la forma y masa de los cuerpos que intervienen, etc.

Si ahora remplazamos las integrales con las ecuaciones 15-16 y 15-17 obtenemos

$$e = \frac{u - v'_A}{v_A - u} \qquad (15\text{-}19)$$

Un análisis similar de la partícula B conduce a la relación,

$$e = \frac{v'_B - u}{u - v_B} \qquad (15\text{-}20)$$

Como los cocientes de las ecuaciones 15-19 y 15-20 son iguales , también son iguales al cociente obtenido de la suma de sus numeradores y sus denominadores, respectivamente. Por consiguiente tenemos:

$$e = \frac{(u - v'_A) + (v'_B - u)}{(v_A - u) + (u - v_B)} = \frac{v'_B - v'_A}{v_A - v_B} \qquad (15\text{-}21)$$

Resolviendo 15-21 tenemos

$$v'_B - v'_A = e(v_A - v_B) \qquad (15\text{-}22)$$

Finalmente, la pareja de ecuaciones que caracterizan al impacto central directo es:

$$v'_A = v_A + \frac{m_B}{m_A} v_B$$

$$v'_B = v'_A + e(v_A - v_B) \qquad (15\text{-}23)$$

Coeficiente de restitución: e

Este valor sabemos que está en función de los materiales, la velocidad, la masa, etc. Pero ¿qué significa? Pues bien, tenemos interés en dos casos particulares de choques.

Impacto perfectamente plástico (*e=0*). Cuando dos materiales se impactan no hay periodo de restitución y las partículas permanecen unidas luego del impacto.

Impacto perfectamente elástico (e=1). Cuando dos materiales impactan, sus velocidades luego de la colisión son iguales en magnitud a las velocidades antes de la colisión. Otra forma de verlo implica que los impulsos recibidos por cada partícula durante el periodo de deformación y durante el periodo de restitución son iguales.

Vale la pensar que toda colisión de dos cuerpos ocurre en un periodo de tiempo, mientras más blandos sean los cuerpos (menor valor para *e*) mayor será la duración de la colisión. Adicionalmente, se debe considerar que durante un choque, las fuerzas ejercidas por los cuerpos varían. Como ve, la situación se complica si tratamos con cuerpos grandes como pelotas, autos y hasta gomas de borrar. *Nosotros trabajamos con partículas, las cuales, al ser suficientemente pequeñas y compactas permiten que nuestras ecuaciones sean válidas.*

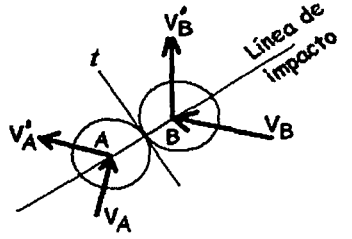


Figura 15-16
Impacto central oblicuo. Figura de Ferdinand. P. Beer, E. Russell Johnston Jr. "*Mecánica Vectorial para Ingenieros*", McGraw Hill, 5° ed., 1990, pg. 643.

Impacto central oblicuo

Consideremos ahora el caso en el que las velocidades de las dos partículas que chocan no son paralelas a la línea de impacto: esto es un impacto central oblicuo (figura 15-16). Puesto que las velocidades v_A y v_B de las partículas luego del impacto son de dirección y magnitud desconocida, su determinación necesitará que se usen cuatro ecuaciones independientes.

El sistema de referencia que emplearemos para determinar las velocidades luego de la colisión es el normal-tangencial. Bajo esa condición, la línea de impacto nos servirá para definir el eje *n* de nuestro sistema de referencia.

Suponiendo que las partículas son perfectamente lisas y que no hay rozamiento entre ellas, observamos que los únicos impulsos aplicados a las partículas durante el impacto son causadas por fuerzas internas dirigidas a lo largo de la línea de impacto, o sea, a lo largo del eje *n* (figura 15-17). Entonces:

- La componente tangencial de la cantidad de movimiento de cada partícula se conserva; en consecuencia, la componente tangencial de la velocidad de cada partícula tampoco se altera

$$v_{A\_t} = v'_{A\_t}; \qquad v_{B\_t} = v'_{B\_t} \tag{15-24}$$

- La componente normal de la cantidad de movimiento lineal total de las dos partículas se conserva.

$$m_{A\_n}v_{A\_n} + m_{B\_n}v_{B\_n} = m_{A\_n}v'_{A\_n} + m_{B\_n}v'_{B\_n} \tag{15-25}$$

- Podemos plantear la definición de un coeficiente de restitución a partir de las componentes normales de las velocidades de cada partícula: antes y después del impacto.

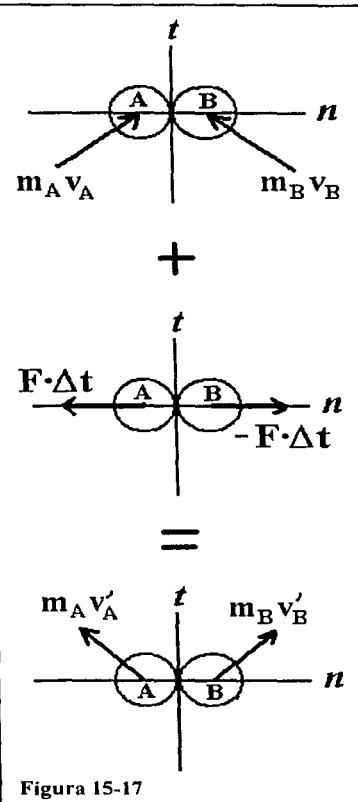


Figura 15-17
Figura de Ferdinand. P. Beer, E. Russell Johnston Jr. "*Mecánica Vectorial para Ingenieros*", McGraw Hill, 5° ed., 1990, pg. 644.

$$v'_{B\_n} - v'_{A\_n} = e\left(v_{A\_n} - v_{B\_n}\right) \qquad (15\text{-}26)$$

15-7 Fluido simulado con partículas gigantes

El viento es lo que llamamos un fluido, nuestro problema ahora es simularlo, y esto lo haremos empleando otra vez el concepto de partículas gigantes:

El comportamiento de un fluido, ya sea líquido o gas, puede ser modelado con partículas de masa m. No existe interacción entre estas partículas. La interacción con partículas de otro tipo se modela considerando los efectos de la fricción viscosa.

El párrafo anterior nos da la pauta para definir una partícula de fluido como una cierta cantidad de algún material con forma esférica y constante. La región que abarca esta partícula es tal que si un cuerpo pasa por ella, padecerá los efectos de una fricción viscosa.

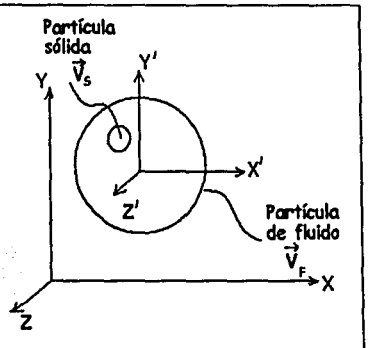


Figura 15-18
Interacción entre una partícula sólida y una partícula de fluido.

Decimos que estas partículas son gigantes debido a que una computadora casera e incluso una estación de trabajo, no tienen suficiente poder para simular, en tiempo real, la interacción de cientos o quizá miles de partículas de fluido con otro tipo de partículas. De tal forma, haciendo que cada una abarque un mayor volumen reducimos considerablemente la cantidad de cálculos.

¿Cómo afecta una partícula de fluido a una partícula sólida?

Para responder a esta pregunta recurrimos a la figura 15-18. Tal figura nos muestra una partícula que representa a un sólido atravesando a una partícula de fluido. Seguramente puede notar la diferencia en tamaños. Para nuestro caso en particular, es muy conveniente que una partícula de fluido ocupe mayor volumen que una partícula sólida.

Como ya sabemos, la partícula sólida padecerá los efectos de la fricción viscosa. Podemos cuantificar la fuerza de la fricción que la afecta como:

$$F = -B\left(v_{sólido} - v_{fluido}\right) \qquad (15\text{-}27)$$

Note que la fricción está en función de la velocidad relativa de la partícula de sólido respecto de la partícula de fluido. Es signo menos indica que la dirección de la fuerza es contraria a la velocidad relativa del sólido.

Simplificaciones

Solo en este caso, vamos a considerar dos situaciones que nos permitan simplificar en gran medida los cálculos a riesgo de restar realismo a nuestros modelos.

- Las partículas de fluido pueden ocupar el mismo espacio.
- Las partículas de fluido están libres de la tercera ley de Newton por lo que no modificarán su movimiento debido a la interacción con otras partículas de su mismo tipo o de otro.

Aunque los dos puntos anteriores violan las leyes de la Física, lo hago para simplificar en gran medida la cantidad de cálculos y decisiones que implica la simulación de fluidos.

Figura 15-19
En la figura se muestra un patrón de hilos correspondiente a un suéter deportivo.

15-8 Cálculo de las constantes del amortiguador y del resorte

Inicialmente voy a plantear la estructura general del sistema de partículas. Esta red debe conformar una lámina que simule la tela. Seguramente notará que no menciono cuales serán los valores de las constantes de cada resorte y de cada amortiguador. En las secciones que siguen veremos que la respuesta tiene algo de ciencia y mucho de conocimiento empírico ya que las constantes de resortes y amortiguadores están en función de las masas que soportan y de las posiciones relativas entre partículas

Planteamiento general de la red de partículas

Cuando examinamos un trozo de tela a través de una lente de aumento, notamos una red de hilos entretejidos. Los hilos de algunas telas presentan trayectorias que son ortogonales entre sí. Otras telas, en cambio se han tejido formando nudos muy cercanos unos de otros. La figura 15-19 nos muestra los patrones del tejido de un suéter deportivo.

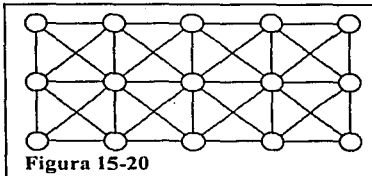


Figura 15-20
Patrón que muestra la distribución de partículas y enlaces para simular un trozo de tela cualquiera

El planteamiento general de nuestra red debiera simular las interacciones de los hilos tejidos de estas telas, así que propongo una red como la mostrada en la figura 15-20.

Pensemos un poco en las características de esta red:

- Todos los enlaces no están rígidamente sujetos a las partículas por lo que presentan flexibilidad en su rotación respecto del centro de cada una.
- Es deseable que los enlaces horizontales y verticales presenten una cierta rigidez en cuanto a su capacidad de estirarse o comprimirse.
- Los enlaces en diagonal si debieran ser blandos en cuanto a su capacidad de estirarse y comprimirse, de otra forma, la red sería completamente rígida

En realidad hace falta mencionar otras características importantes, pero por ahora estas justifican el desarrollo de la presente sección.

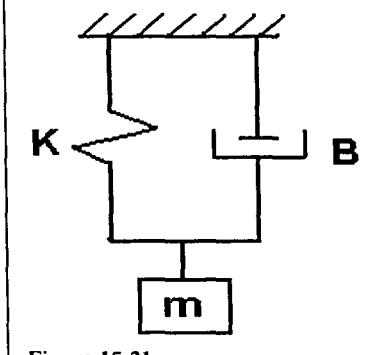


Figura 15-21
Sistema masa-resorte-amortiguador en equilibrio y sin excitaciones externas.

El sistema de segundo orden

Para poder fijar los valores de las constantes de cada resorte y cada amortiguador hace falta recurrir un poco a las matemáticas, veamos entonces la figura 15-21 que nos muestra una masa sostenida por un resorte y un amortiguador. El sistema está en equilibrio y no hemos considerado la acción de fuerzas externas para simplificar un análisis previo.

La ecuación que modela al sistema de la figura 15-21 es la siguiente:

$$\ddot{x} + \frac{B}{m}\dot{x} + \frac{k}{m}x = 0 \qquad (15\text{-}28)$$

Así es, se trata de un sistema de segundo orden y recordando las clases de cálculo, la ecuación homogénea general de cualquier sistema de segundo orden es:

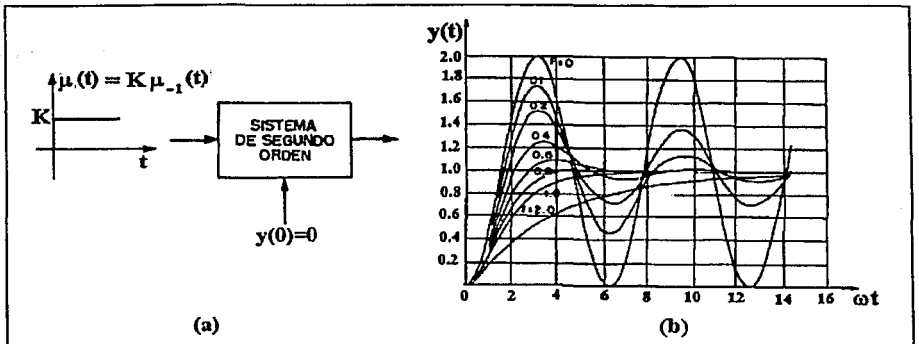


Figura 15-22
En (a) tenemos un sistema de segundo orden excitado por una función escalón unitario. En (b) el resultado que se observa en la gráfica normalizada indica lo que ocurre cuando se varía el factor de amortiguamiento del sistema. Francisco J. Rodríguez Ramírez, "*Dinámica de Sistemas Físicos*", Trillas, 1989, pg.283.

$$\ddot{x} + (2\gamma\omega_n)\dot{x} + \omega_n^2 x = 0 \qquad (15\text{-}29)$$

en donde:

- γ : Factor de amortiguamiento relativo del sistema.
- ω : Velocidad angular no amortiguada del sistema.

Si recuerdo bien, el factor de amortiguamiento determina el comportamiento general de nuestro sistema masa-resorte-amortiguador

- $\gamma = 0$: No amortiguado (como si no existiera el amortiguador).
- $\gamma \in (0..1)$: Subamortiguado.
- $\gamma = 1$: Críticamente amortiguado (el sistema se estabiliza rápidamente).
- $\gamma > 1$: Sobreamortiguado (el sistema se estabiliza con lentitud).

Ahora bien, es necesario mencionar la equivalencia entre la ecuación general de segundo orden y nuestra ecuación del sistema masa-resorte-amortiguador:

$$\gamma = \frac{B}{2\sqrt{km}} \qquad (15\text{-}30)$$

$$\omega_n^2 = \frac{k}{m}$$

Para comprender mejor el efecto del factor de amortiguamiento debemos revisar la respuesta de nuestro sistema cuando se excita con una función escalón unitario. La gráfica de la figura 15-22.a nos muestra la función escalón en tanto que la figura 15-22.b nos muestra una gráfica normalizada para diferentes valores de γ.

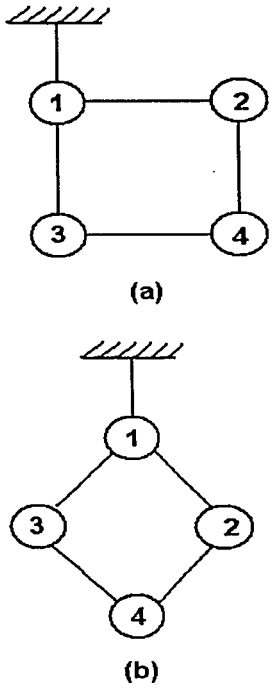


Figura 15-23
Ejemplo de un sistema de partículas. Note que en (a) el enlace "1" es afectado solo por la masa "3". Más tarde, en (b), el enlace "1" es afectado por las partículas "2", "3" y "4".

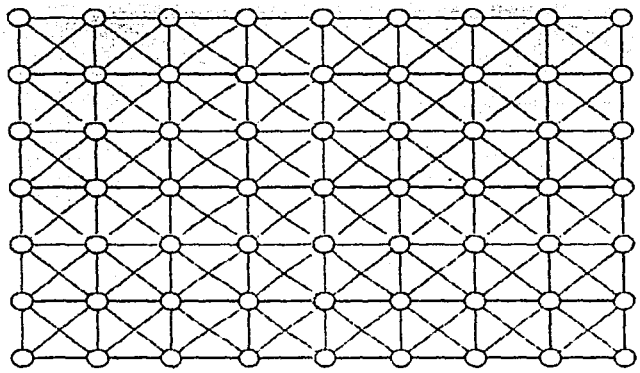


Figura 15-24
Este monótono patrón representa un sistema de 7 por 10 partículas enlazadas. Los enlaces horizontales y verticales requieren una gran fuerza para modificar su longitud, en tanto que los enlaces en diagonal se deforman fácilmente.

De la figura 15-22.b observamos que la masa oscilará alrededor de su posición cuando $\gamma = 0$. En el caso de que $0 < \gamma < 1$ el sistema oscilará hasta estabilizarse. También notamos que cuando $\gamma = 1$, el sistema tiende rápidamente a estabilizarse sin presentar oscilaciones. El último caso es el de $\gamma > 0$, que según observamos, cuanto mayor es este valor, al sistema le tomará más tiempo estabilizarse sin oscilaciones.

Aplicación empírica del sistema de segundo orden a un sistema de partículas

Cualquiera que sea la solución que empleemos para calcular los coeficientes de cada resorte y cada amortiguador, ya sea dependiente o independiente del tiempo, la masa será una variable que siempre encontraremos. Tratándose de un sistema de partículas enlazadas, la masa puede ser una cantidad variable dependiendo de las posiciones relativas de las partículas. Veamos entonces la figura 15-23.a. Tenemos una partícula fija y otras tres partículas afectadas por la fuerza de gravedad. En este instante, sólo la partícula 3 está afectando al enlace 1. Observe ahora la figura 15-23.b que ilustra al mismo sistema unos instantes después. Ahora son tres partículas actuando sobre el enlace 1. Las partículas 3 y 4 lo tensan hacia abajo, en tanto, la partícula 2 actúa sobre la partícula 4 empujándola a través de su enlace. Como puede ver, la masa que soporta cada enlace es difícil de determinar, no obstante, podemos considerar una buena aproximación si:

a) Todas las partículas del sistema tienen la misma masa.
b) El comportamiento de todos los enlaces del sistema tiende a ser el mismo cuando $\gamma > 0.8$.

Los incisos anteriores pueden resumirse en una función que acepta tres parámetros: masa, coeficiente del resorte y amortiguamiento relativo. El resultado será un coeficiente de amortiguamiento propio del enlace *B*. La ecuación que nos permite realizar este cálculo es la siguiente:

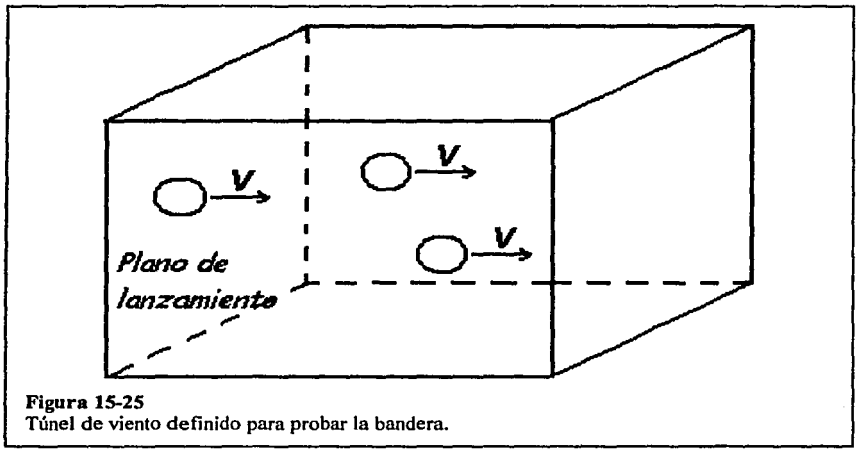


Figura 15-25
Túnel de viento definido para probar la bandera.

$$B = 2\gamma\sqrt{Km} \qquad (15\text{-}31)$$

Nuevamente, la masa que consideraremos en la ecuación corresponde a la de una partícula y no a la del sistema completo.

Aplicación empírica del sistema de segundo orden a un sistema de partículas

Voy a comenzar con la estructura que definirá nuestra bandera: esta la formará una matriz de 7 por 10 partículas. Las mismas estarán enlazadas según se muestra en la figura 15-24. El valor de la masa que se ha considerado para cada partícula es de 0.02 kg. Así que en total tenemos 1.4 kg, lo que implica una bandera grande. En cuanto al tamaño de las partículas, este debió escogerse de tal forma que diera una mejor presentación en pantalla. El radio es en sí de 0.5 pero sin unidades.

Respecto de los enlaces, deben cumplirse las siguientes condiciones:

- Todos los enlaces no están rígidamente sujetos a las partículas por lo que presentan flexibilidad en su rotación respecto del centro de cada una.
- Es deseable que los enlaces horizontales y verticales presenten cierta rigidez en cuanto a su capacidad de estirarse y comprimirse. Para éstos, el coeficiente del resorte se ha escogido de $k = 20\frac{N}{m}$ lo cual es muy rígido. El factor de amortiguamiento es de $\gamma = 2$, por lo que si consideramos las masa de 0.02kg, el factor de amortiguamiento es $B = 2.53\frac{N}{m/s}$. En general el comportamiento de estos enlaces es rígido ya que se requiere de una fuerza grande para deformar el enlace.
- Los enlaces en diagonal si debieran ser blandos en cuanto a su capacidad de estirarse y comprimirse, ya que de otra forma, la red sería completamente rígida. Para éstos, el coeficiente del resorte se ha escogido de $k = 0.02\frac{N}{m}$ lo cual es

bastante blando. El factor de amortiguamiento es $\gamma = 0.9$, por lo que si consideramos la masa de 0.02kg, el factor de amortiguamiento es $B = 0.036 \frac{N}{m/s}$. En general el comportamiento de estos enlaces es subamortiguado y basta una fuerza pequeña para deformar al enlace.

En cuanto a las fuerzas externas, debemos considerar las siguientes:

- La fuerza de atracción gravitacional. Esta fuerza se calcula a partir de la segunda ley de Newton: $W = mg$, o sea, peso igual a masa por aceleración debida a la gravedad. Si consideramos que la masa es de 0.02kg y la aceleración es de $4.5 {}^{m}/_{s^2}$, la fuerza de cada partícula es de $f = 0.196N$.
- La fuerza que ejerce el viento mientras pasa rozando a la bandera. Los detalles del viento los veremos a continuación.

En cuanto al viento, éste se simula con partículas que cumplen las siguientes condiciones:

- El viento se simula con partículas gigantes.
- Las partículas del viento pueden ocupar el mismo espacio. Esta condición reduce en gran medida la cantidad de cálculos que implica el que dos partículas gigantes de fluido colisionen tal como ocurre con dos gotas de agua.
- Las partículas de fluido están libres de la tercera ley de Newton por lo que no modificarán su movimiento debido a la interacción con otras partículas de su mismo tipo o de otro.
- Las partículas de viento son lanzadas desde posiciones aleatorias, siguiendo todas la misma dirección con velocidad aleatoria. Al respecto, la figura 15-25 nos ilustra la forma en que se simula el viento: observe la semejanza con un túnel de viento.
- En cuanto a sus características, el coeficiente de fricción viscosa se ha escogido de $1.7 \frac{N}{m/s}$. El radio de la partícula de viento es de 4 veces el tamaño de una partícula sólida, o sea, radio igual a 2 unidades. La masa de las partículas de viento, debido a las simplificaciones, ya no es importante.

15-9 Solución numérica

Recordemos la ecuación 15-4. Notará que es La Segunda Ley de Newton y no obstante esta maravilla del mundo no nos proporciona la posición o incluso la velocidad de cada partícula de la bandera. Para ahorrar espacio y tiempo, voy a anotar las ecuaciones que nos permiten calcular valores para las tres variables importantes de una partícula: aceleración, velocidad y posición:

$$a = \frac{\sum Fuerzas}{m}$$
$$v_2 = a\Delta t + v_1 \qquad (15\text{-}32)$$
$$x_2 = x_1 + \frac{\Delta t}{2}(v_2 + v_1)$$

Listo, la primera de las tres ecuaciones nos da la aceleración de la partícula en función de la fuerza neta que actúa sobre ella y de su masa. La segunda ecuación nos permite obtener

la velocidad de la misma partícula en función de la aceleración y de la velocidad que tenía la partícula previamente. La última ecuación nos da la posición de la partícula.

15-10 Conclusiones

Es posible que Ud. Se haga la siguiente pregunta: *y esto ¿funciona?* La respuesta es *sí, funciona.* No obstante, como se verá en las pruebas del siguiente capítulo, faltará algo de realismo en el movimiento.

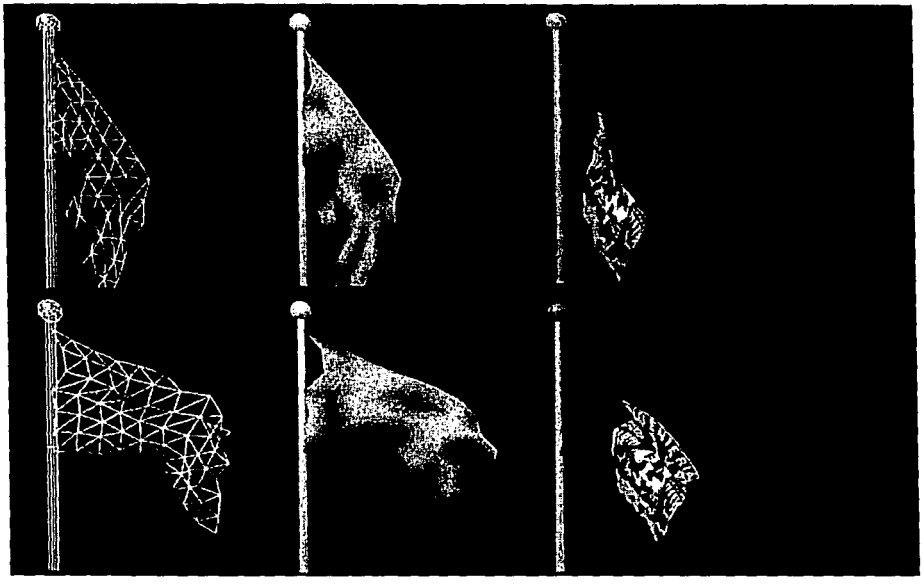
Me atrevo a pensar que la idea básica en la que se fundamentó el desarrollo de la bandera tiene grandes posibilidades si consideramos lo siguiente:

- Una mejor técnica para calcular los valores de resortes y amortiguadores.
- Evitar simplificaciones como las consideradas para simular el viento.
- Un mejor conocimiento de las capacidades del procesador principal y del subsistema de video.

El sistema de partículas que forma a la bandera pertenece a un tema de amplio desarrollo en la industria médica y cinematográfica. Grupos de investigación como INRIA, han desarrollado modelos basados en conceptos como deformaciones de cuerpos, lo que proporciona una forma simple de calcular los enlaces entre partículas.

Aún faltan capítulos por desarrollar, el penúltimo corresponde a las pruebas realizadas a mi modelo empírico y el último son las conclusiones que seguramente ya intuye.

Paciencia, falta poco.



Capítulo 16 El modelo de superficie y texturizado

Una vez creado un modelo de movimiento para nuestra bandera, se hace necesario decidir que es lo que se va a ver en pantalla. En fin, a este respecto tenemos entonces tres modelos de superficie que servirán para hacer nuestras pruebas.

- Modelo de alambre
- Superficie en color verde mate
- Superficie texturizada con el escudo de la Facultad de Ingeniería

Como ya sabemos, cada objeto que se dibuja en la pantalla se construye a base de primitivas. Para nuestro caso hemos elegido el triángulo, el cual representa un buen compromiso entre rendimiento y aspecto. Así, el modelo de alambre es una superficie que dibuja el perímetro de todos los triángulos en color verde sobre fondo negro.

El siguiente modelo de superficie también se construye con primitivas triangulares, sin embargo aquí si se deben dibujar los triángulos por lo que la bandera parece un trozo de tela verde sobre fondo negro.

El último modelo representa mi mejor aproximación a lo que sería una bandera azul portando el escudo de la Facultad de Ingeniería de la UNAM. Esta superficie también se construye con primitivas triangulares sobre fondo negro.

Debo aclarar que este capítulo, aunque requiere del conocimiento de OpenGL, trataré que sea lo más básico posible. Esto significa que no implicaré los tediosos detalles del lenguaje de programación, sino solamente las generalidades.

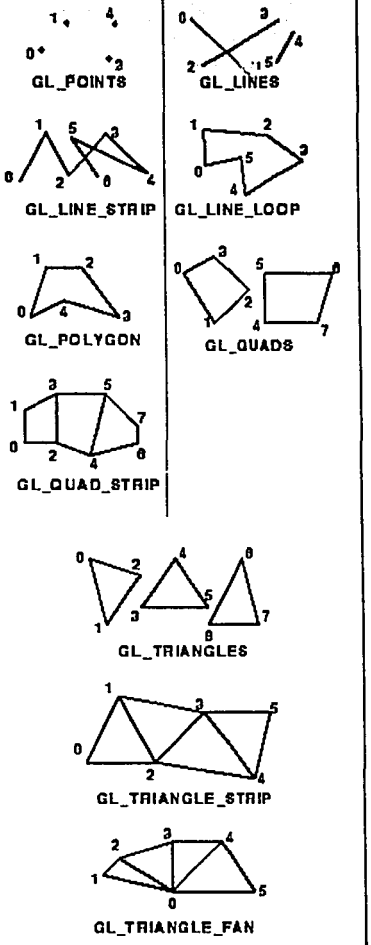


Figura 16-1
Tipos de primitivas soportadas por OpenGL. Cortesía de : http://www.sgi.com/developers/library/resources/openglxp.html#glw_widget

16-1 Las primitivas de OpenGL

Cuando se escribe un programa gráfico empleando OpenGL, se comienza con un conjunto simple y pequeño de primitivas. Siendo así, la sofisticación viene de la combinación y uso de éstas. La figura 16-1 muestra las primitivas geométricas disponibles en OpenGL. Note el orden en que se numeran los vértices, en particular para primitivas como GL_TRIANGLE y GL_TRIANGLE_STRIP.

La primera primitiva

Básicamente, OpenGL emplea un tipo de primitiva más primitiva que las otras y la conocemos como punto. Recordemos que un punto es una posición. Así, para definir una línea necesitamos dos puntos que sean los extremos. Luego entonces, para definir un triángulo, requerimos de tres puntos que sean los vértices y así sucesivamente. Concluyendo, esta API OpenGL permite al programador construir todo a partir de puntos, veamos entonces como se hace.

glBegin

Para iniciar una primitiva se requiere definir una rutina *glBegin*. A esta rutina se le pasa el tipo de primitiva deseado como un argumento. Los tipos de primitivas válidos se aprecian en la figura 16-1 y se identifican por constantes simbólicas. La misma figura muestra estas constantes. En nuestro caso el comando correspondiente es:

```
glBegin(GL_TRIANGLE_STRIP)
```

glVertex

OpenGL considera un vértice como una entidad caracterizada por su posición, un vector que parte de esta posición, un color y otras informaciones. Por simplicidad, en esta sección consideraremos que un vértice se caracteriza complemente por la posición.

OpenGL tiene una familia de rutinas que sirven para especificar vértices coordenados a partir de distintos tipos de datos: punto flotante, entero, etc. Todas las rutinas comienzan con el mismo nombre: *glVertex*. En resumen, podemos especificar un vértice a partir del siguiente comando:

```
glVertex3f ( x, y, z );
```

Como recordará, OpenGL trabaja en un mundo 3D, por ello vemos que el comando tiene tres parámetros: (x, y, z).

La rutina glVertex también acepta como parámetro un puntero a un arreglo con las coordenadas del vértice:

```
glVertex3fv ( v : pglfloat );
```

El parámetro *v* de la rutina es lo que se llama puntero, es decir, una variable que almacena la dirección en memoria de las coordenadas que definen la posición de un punto. En general, si deseamos generar un triángulo, sólo tenemos que escribir:

```
glVertex3fv ( vértice_1 );
glVertex3fv ( vértice_2 );
glVertex3fv ( vértice_3 );
```

glEnd

Una primitiva OpenGL es completada invocando la rutina *glEnd*. A lo largo de la definición de diversos vértices entre *glBegin* y *glEnd*, se anexan comandos para especificar color, material, vectores normales y coordenadas de aplicación de texturas. Entonces, la invocación de la rutina es como sigue:

```
glEnd;
```

glNewList y glEndList

Antes de definir normales, colores y otras coordenadas, vamos a considerar la siguiente situación: en OpenGL, existen dos modos de operación:

- Modo inmediato, o sea, es posible hacer que el sistema dibuje una primitiva apenas se haya definido.
- Modo lista, esto es, los comandos que definen una figura a base de primitivas se almacenan en una lista para su posterior ejecución.

El modo lista tiene una particular ventaja y consiste en una mayor rapidez en cuanto a la interpretación de los comandos y esto es porque los procesa en modo tubería. En el capítulo 4, acerca de microprocesadores, hablé de lo que es una tubería. En fin, para mayor comodidad, lo menciono otra vez. Una tubería es como una línea de montaje, en la que los artículos se ensamblan por etapas; cada vez que un futuro artículo deja una etapa, otro futuro artículo entra. Como consecuencia del trabajo en líneas de montaje, los artículos se ensamblan en menor tiempo.

OpenGL permite definir comandos que describen primitivas y otras informaciones en modo lista, de tal suerte, si queremos empezar una lista solo escribimos:

```
glNewList (handle)
```

En donde *handle* es un número que identifica a nuestra lista;

Para terminar la definición de una lista solo debemos invocar a continuación el comando

```
glEndList
```

glMaterial

Además de definir los vértices que forman a las primitivas, debemos también definir las características ópticas de las mismas; me refiero al color y en nuestro caso, la superficie tendrá un color verde terciopelo.

En OpenGL, se debe definir el color del material para cada modelo de iluminación disponible, así, nosotros contamos con un color ambiente y un color difuso: ambos definen el color del material. También contamos con un color especular, el cual ayuda a dar una definición ya sea mate, plástica, terciopelo y hasta metálica a cualquier superficie. Veamos como se hace.

Primero definimos algunas constantes numéricas que representen al verde

```
VerdeAmbiente    :  Array [1..4 ] of glfloat = (0.0215, 0.1745, 0.0215, 0.55);
VerdeDifuso      :  Array [1..4 ] of glfloat = (0.07568, 0.61424, 0.07568, 0.55);
VerdeEspecular   :  Array [1..4 ] of glfloat = (0.633, 0.727811, 0.633, 0.55);
VerdeBrillo      :  6.8;
```

Las primeras constantes, ambiente y difusa, definen el color del material. Las últimas dos, especular y brillo nos ayudarán a dar una apariencia de terciopelo a nuestra bandera. Ahora veamos como se aplican las constantes en la definición de algún material.

```
GlNewList( iden_bandera );
//
//     Características ópticas
glMaterialfv ( GL_FRONT, GL_AMBIENT, @VerdeAmbiente);
glMaterialfv ( GL_FRONT, GL_DIFFUSE, @VerdeDifuso);
glMaterialfv ( GL_FRONT, GL_SPECULAR, @VerdeEspcular);
glMaterialfv ( GL_FRONT, GL_SHININESS, VerdeBrillo);
//
//     Definición de primitivas
glBegin (GL_TRIANGLE_STRIP);
:
```

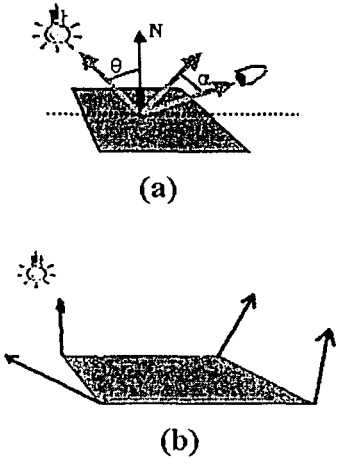


Figura 16-2
Uso de un vector normal para determinar el color que tendrá una superficie en función de una fuente de luz y de la orientación espacial de la misma superficie.

glNormal

Como ya mencioné arriba, en el segmento de *glVertex*, en OpenGL cada vértice se define como una posición, un color, un vector normal y otras informaciones. Nuestra pregunta ahora es ¿para qué nos sirve el vector normal? Observe la figura 16-2.a. el vector normal determina la orientación espacial de una superficie y es la referencia respecto de la cual se considera la dirección de la fuente de luz y la dirección de vista; de esta forma es como se determina el color de las superficies en OpenGL. Observe ahora la figura 16-2.b. En este caso tenemos una superficie cuadrilateral definida por cuatro vértices, cada vértice tiene definido un vector normal; OpenGL determinará el color de cada vértice y los colores internos a la superficie se obtienen por interpolación. Es posible llevar esto aún más lejos si consideramos una superficie compuesta con muchos cuadrilaterales.

En general, si deseamos definir una superficie triangular solo sigamos el siguiente código:

```
glNewList (Iden_Bandera );
//
//     Características ópticas
glMaterialfv ( GL_FRONT, GL_AMBIENT, @VerdeAmbiente);
glMaterialfv ( GL_FRONT, GL_DIFFUSE, @VerdeDifuso);
glMaterialfv ( GL_FRONT, GL_SPECULAR, @VerdeEspcular);
glMaterialfv ( GL_FRONT, GL_SHININESS, VerdeBrillo);
//
//     Definición de primitivas
glBegin (GL_TRINAGLE_STRIP);
//
glNormal3fv ( normal_1 );
glVertex3fv( vértice_1 );
//
glNormal3fv ( normal_2 );
glVertex3fv( vértice_2 );
//
```

```
glNormal3fv ( normal_2 );
glVertex3fv( vértice_2 );
//
glEnd;
glEndList;
```

16-2 El modelo de superficie de alambre

Es el modelo de movimiento el que tiene una estructura de datos que almacena todas las partículas que forman a la bandera. Como ya es sabido, una de las propiedades que definen a una partícula es su posición. A partir de este momento consideraremos la posición de cada partícula como la posición de un punto de control de nuestra bandera.

Adicionalmente, el modelo de movimiento almacena la información de las partículas en forma de lista. OpenGL también almacena la información de los vértices en forma de lista. Ahora bien, ambas listas son incompatibles por lo que se requiere agregar carga de trabajo al procesador para la administración de la información.

Definiendo la superficie de alambre

Como ya también sabemos, cada objeto que se dibuja en la pantalla se construye a partir de primitivas: puntos, líneas, polígonos, etc. En nuestro caso, la primitiva elegida es el triángulo. La figura 16-3.a nos muestra la definición de un triángulo a partir de la especificación de tres vértices. Note la forma en como se hace la definición.

Lo que haremos ahora es acomodar los triángulos en forma de listón, según se muestra en la figura 16-3.b. El código del listón se muestra a continuación:

```
//
//    Algoritmo para cálculo de vectores normales
//
:
//
//    Algoritmo para definir un listón de triángulos
glBegin (GL_TRINGLE_STRIP );
 For c:=0 to cols – 1 do begin
      glNormal3fv (@Normals [r,c]);
      ParticlesSystem.glPosition (r*cols+c);
      GlVertex3f(ParticlesSystem.Pos_X, ParticlesSystem.Pos_Y, ParticlesSystem.Pos_Z);
//
      glNormal3fv (@Normals[r+1,c]);
      ParticlesSystem.glPosition((r+1*Cols+c);
      GlVertex3f(ParticlesSystem.Pos_X, ParticlesSystem.Pos_Y, ParticlesSystem.Pos_Z);
 end;
glEnd;
```

En general, el código presentado se ve difícil de entender, sin embargo no es cosa que deba preocuparnos mucho. Veamos, podemos distinguir la instrucción que define la primitiva *glBegin(GL_TRINGLE_STRIP)*. Podemos ver también la definición de un vector normal antes de cada vértice *glNormal3fv(...)*. Como ya sabemos, el vector normal sirve en los modelos de iluminación. El comando *ParticlesSystem.glPosition(...)* no es propio de OpenGL, sin embargo es necesario para calcular la posición del actual punto de control. Finalmente, *glVertex(...)*; define un vértice de cada triángulo en OpenGL.

Ahora ya solo queda unir los listones para formar nuestra bandera. La figura 16-4 nos muestra la definición de cada listón a partir de los puntos de control: esta figura es un ejemplo y solo muestra un segmento de la bandera.

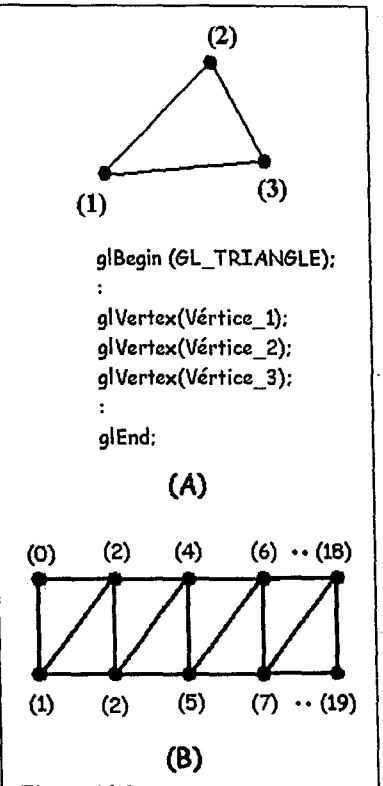


Figura 16-3
En (a) se muestra la construcción de una primitiva mediante la definición de tres puntos de control. En (b) se muestra un listón de triángulo que se construye especificando los vértices en el orden que indica la figura.

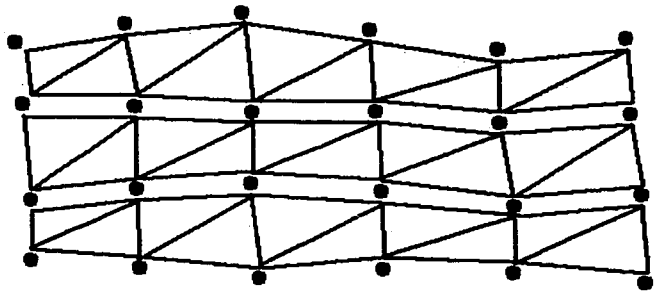


Figura 16-4
Construcción de la superficie de bandera mediante listones que siguen los puntos de control. En la representación final de la bandera, no se denota separación alguna entre los listones.

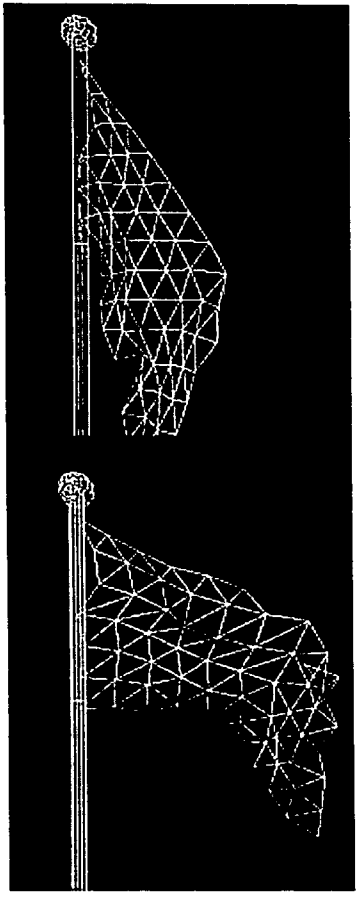


Figura 16-5
Dos escenas que muestran la construcción de nuestra bandera a partir de primitivas triangulares.

Bueno, podría anotar a continuación el código correspondiente, sin embargo es algo complejo de explicar. Espero que con la figura 16-4 queden entendidos los principios a partir de los cuales ensamblo la bandera.

Falta algo más, dentro de todo el conjunto de comandos de OpenGL hay dos que nos permiten elegir como será la representación de nuestra primitiva sen pantalla. Estos comandos son:

```
glSahdeModel( GL_SMOOTH );//Sombreado suave
glPolygonMode (GL_FRONT,GL_LINE);     //Solo caras frontales, no hay cuerpos transparentes
```

El primer comando *glSahdeModel(GL_SMOOTH)*, indica a OpenGL que el tipo de sombreado será suave, es decir, interpolará linealmente los colores de cada píxel entre dos extremos de cada línea del polígono.

Dentro del segundo comando, *glPolygonMode (GL_FRONT,GL_LINE)*, el parámetro *GL_LINE* indica a OpenGL que represente en pantalla solo el perímetro de cada primitiva; a consecuencia, solo veremos triángulos moverse en la pantalla. La figura 16-5 nos muestra varias escenas de una bandera representada con un armazón de alambres color verde: el mástil que sostiene a la bandera no es propio del proyecto, es una cortesía para aumentar el realismo de la escena.

16-3 El modelo de superficie verde mate

Este modelo se construye de manera idéntica al modelo de superficie de alambre, la única diferencia está en la definición del siguiente comando

```
glPolygonMode (GL_FRONT,GL_FILL);     //Solo caras frontales, no hay cuerpos transparentes
```

Observe el segundo parámetro: GL_FILL. Este parámetro indica a OpenGL que dibuje la superficie de cada primitiva. La figura 16-6 nos muestra dos tomas de la bandera ondeante.

Figura 16-7
Imagen que formará la textura de la bandera. Cortesía del Departamento de Telecomunicaciones de la Facultad de Ingeniería de la UNAM.

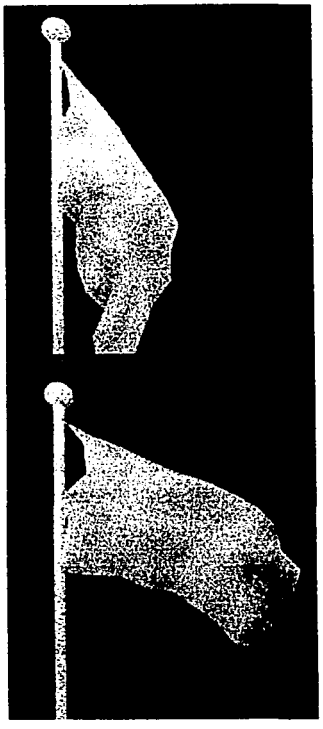


Figura 16-6
La figura exhibe dos tomas correspondientes a la bandera representada con una superficie que simula el terciopelo color verde. La apariencia es una mera elección del programador

16-4 Superficie texturizada con el escudo de la Facultad de Ingeniería

Este modelo de superficie es el más difícil de explicar debido a la compleja tarea de fragmentar el dibujo del escudo (figura 16-7) y de aplicarlo a la superficie de la bandera.

OpenGL permite aplicar texturas de hasta 256 por 256 téxeles a cualquier superficie. Sin embargo, estas dimensiones no proporcionan una imagen de calidad. Por esta razón, se propone fragmentar el dibujo en pequeñas texturas de 64 por 64 téxeles y aplicarlas a cada triángulo que conforma nuestra bandera.

El primer paso: fragmentar el dibujo de la bandera

La figura 16-7 nos muestra la imagen original, ahora, la figura 16-8 nos muestra el mismo dibujo a través de una rejilla que representa los fragmentos en los que será separada. Uno de los cuadros de la figura 16-8 tiene marcado un sistema de referencia en coordenadas de textura uv y que nos servirá de guía para la aplicación de texturas a los triángulos de nuestra bandera.

El segundo paso: construcción de la bandera con triángulos

La figura 16-9.a nos muestra algunos de los puntos de control de la bandera. En la misma figura podemos notar como se forma la superficie con pequeñas parejas de triángulos; cada pareja está formando un cuadrilateral, al cual se le aplicará la textura según indica 16-9.b

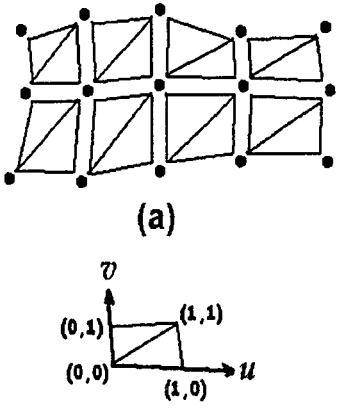


Figura 16-9
Definición de las primitivas que formarán la superficie de la bandera. La superficie ya no se construye con largos listones. En vez de ello, se requiere de parejas de triángulos. Cada pareja forma un cuadrilateral al cual aplicaremos una de las texturas obtenidas al fragmentar la imagen.

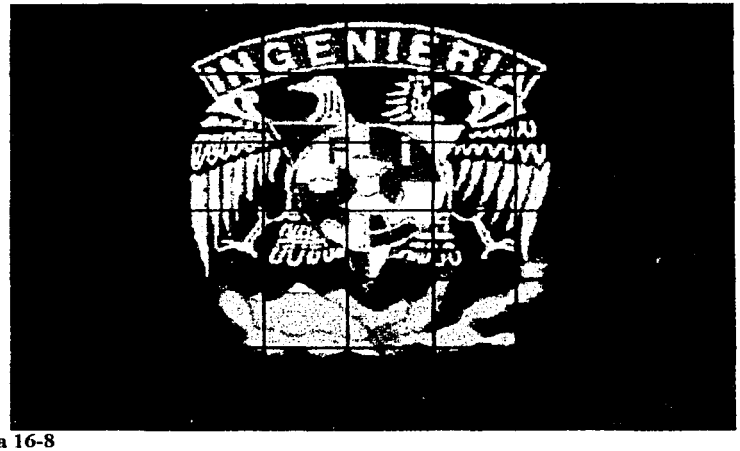


Figura 16-8
División de la imagen de la bandera en pequeñas superficies. Cada segmento será aplicado como una textura a las primitivas que definen la superficie de la bandera.

En este segundo paso, cada vértice en OpenGL se define como una posición, un vector normal que indica la orientación y coordenadas de textura como las que se aprecian en 16-9.b.

El tercer paso implicaría la visualización de cada triángulo y la aplicación de la textura, no obstante, el código que realiza la tarea es complejo y difícil de explicar, por eso preferí emplear dibujos. La figura 16-10 nos muestra una bandera ondeante.

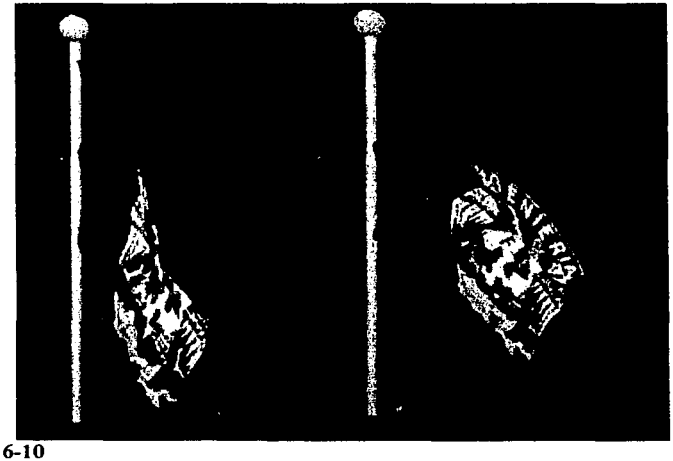
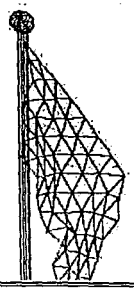
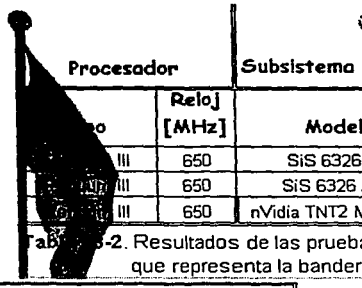


Figura 16-10
Estas escenas representan dos momentos consecutivos de la bandera que ondea en un viento de partículas.



Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium III	650	SiS 6326	20,444
Pentium III	650	SiS 6326 AGP	21,47
Pentium III	650	nVidia TNT2 M64	31,346

Tabla 18-2. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.



Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS 6326 PCI	15,738
Pentium II	450	S3VIRGE DX/GX PCI	19,085
Pentium III	650	SiS 6326 PCI	20,444

Tabla 18-1. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.

Capítulo 17 Pruebas, resultados y conclusiones

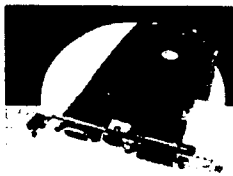
Primero he de mostrarles los fundamentos del algoritmo que controla la simulación de nuestra bandera. Hecho esto, haremos las pruebas que corresponden a la ecuación de movimiento y a tres modelos de superficie en computadoras personales con diferentes tipos de subsistemas de video.

17-1 El benchmark

Un benchmark es un patrón de referencia contra el cual, algunas cosas son medidas. En una medición, un bench mark (dos palabras) es un poste u otra marca permanente establecida en una elevación conocida que es usada como la base para medir la elevación de otros puntos topográficos.

En el entorno de las computadoras y del Internet, el término puede tener cualquiera de los siguientes significados:

- Un conjunto de condiciones contra las cuales un producto o sistema es medido. Los laboratorios PC Magazine frecuentemente examinan y comparan nuevas y diversas computadoras o dispositivos de computadora contra el mismo conjunto de programas de aplicación, interacciones del usuario y otras diversas situaciones.



Benchmarks with Anti-Aliasing Disabled

	GeForce3, P4 1.4	GeForce2 U, P4 1.4	GeForce3, P-III 733	GeForce2 U, P-III 733
Quake III demo001 1024x768x32-bit	131.3fps	128fps	96fps	95.4fps
Quake III demo001 1280x1024x32-bit	86.8fps	81.3fps	75.5fps	72.1fps
Quake III Quaver 1024x768x32-bit	131.3fps	117.9fps	85.3fps	84.1fps
AquaNox 1024x768x32	23fps	5.7fps	20.3fps	4.5fps

Figura 17-1
Cuadro de resultados obtenidos al evaluar diferentes subsistemas gráficos. El software que se probó corresponde a diversas versiones de un popular juego de video. Reproducido de un artículo de Will Smith; GeForce 3; Maximum PC, Vol 6; Num 4; Abril 01, 2001; pg. 36.

- Un programa que es especialmente diseñado para proveer mediciones acerca de un sistema operativo en particular o bien, de algunas aplicaciones.
- Un producto con el cual, los usuarios estén familiarizados y que sirve de punto de comparación contra otros nuevos productos.
- Es un conjunto de criterios o parámetros de desempeño que se espera encontrar en un producto.

Los benchmarks que generan los laboratorios, en ocasiones no reflejan el uso de un producto en el mundo real, por esta razón, muchos usuarios definen un benchmark como "*una medida imprecisa del desempeño de una computadora*". En una cita de cierto tipo de aficionados a las computadoras se puede leer "*en la industria de las computadoras hay tres tipos de mentiras: las mentiras, las mentirotas y los benchmark*".

17-2 Las pruebas

A veces, en una tienda, vemos una revista sobre computadoras en cuya presentación se incluye un CD con un juego de video. Al poco rato de haberla comprado comenzamos a leer los artículos que contiene y entonces encontramos la evaluación de diversos subsistemas de video instalados en equipos del mismo fabricante. La figura 17-1 nos muestra un ejemplo obtenido de una famosa revista para PC.

Los datos que presenta la figura 17-1 son lo llamamos benchmarks y las unidades de medición que logramos ver se abrevian en *fps*. *fps* es la abreviación de "*frames per second*" o bien, al buen español, **cuadros por segundo**. Esto nos indica que las pruebas aplicadas a estos equipos evalúan su capacidad para exhibir en pantalla un cierto número de escenas construidas por segundo. Las pruebas que aplicaré a nuestro modelo son como

estos benchmarks, evalúan la capacidad del computador para realizar el trabajo de simulación de la bandera. Estas pruebas consisten en:

- Evaluar el modelo de movimiento.
- Evaluar el modelo de movimiento y el despliegue de las primitivas que forman una superficie.
- Evaluar el modelo de movimiento y el despliegue de una superficie verde mate.
- Evaluar el modelo de movimiento y el despliegue de una superficie texturizada con la imagen de una bandera.

Son solo cuatro pruebas, su implementación es muy simple y sin embargo no deben despreciarse: muchos programadores, entre ellos yo, pasamos horas y horas frente a las pantallas para desarrollar las herramientas que me permiten dibujar una bandera ondeando en un viento de partículas.

El entorno

¿Cómo es el entorno en que se programó la bandera? Este se puede describir tanto en hardware como en software. El hardware implica a cualquier PC, alto, así es, el tipo de computadora menos eficiente del mercado. Las pruebas se realizan en equipos mostrados en la tabla 17-1.

Procesador		Subsistema gráfico			Datos Generales del sistema	
Tipo	Reloj [MHz]	Modelo	Memoria [Mbytes]	AGP/PCI	Sistema Operativo	Memoria del sistema [Mbytes]
Pentium MMX	200	SiS 6326	4	PCI	Windows 98	64
Pentium II	330	S3VIRGE DX/GX	4	PCI	Windows 98	64
Pentium III	650	SiS 6326	4	PCI	Windows 2000	512
Pentium III	650	SiS 6326	8	AGP2X	Windows 2000	512
Pentium III	650	nVidia TNT2 M64	32	AGP2X	Windows 2000	512

Tabla 17-1. Datos de las computadoras que serán evaluadas con nuestro modelo de bandera ondeando en el viento.

A mí me hubiera gustado, y aún más a mi director de Tesis, emplear un equipo *Silicon Graphics*. La razón de no incluirlo se la debo al escaso soporte que se puede encontrar: de hecho, el obstáculo que me detuvo era el tipo de compilador incompleto proporcionado con el sistema, es decir, un compilador de línea llamado *gcc*.

La selección del hardware mostrado en la tabla 17-1 puede parecer extraña, sin embargo, como explicaré a continuación tiene bastante lógica. Los tres primeros equipos representan algunos peldaños del desarrollo histórico de los procesadores Intel: Pentium MMX, Pentium II y Pentium III[1]. Estos tres distintivos tecnológicos servirán para verificar si los cambios en la tecnología son un avance o un retroceso en cuanto a la tarea de sintetizar imágenes. Así que la pregunta a plantear es:

[1] Pentium MMX, Pentium II y Pentium III son marcas registradas de Intel Corp.

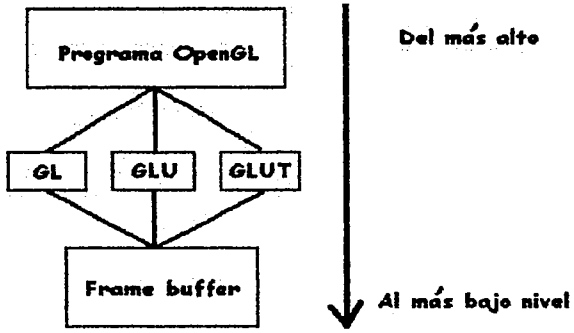


Figura 17-2
Elementos que forman a OpenGL: GL es una librería de comandos básicos, GLU es una extensión que introduce el concepto de cámaras y cuerpos geométricos como cilindros, donas, esferas, teteras, etc. GLUT es una interfaz gráfica que permite al usuario interaccionar con su programa. GLUT permite el uso de ventanas, menús y el control de eventos del teclado y el ratón.

¿Los cambios tecnológicos en el procesador principal y placa base de una computadora representan un avance o un retroceso para la síntesis de imágenes?

Regresemos otra vez a la tabla 17-1 y verifiquemos que los últimos tres equipos son en realidad el mismo. Lo que cambia en cada caso es el subsistema gráfico. Así que ahora tenemos una segunda pregunta por responder.

¿Los cambios tecnológicos en el subsistema gráfico (procesador de video y circuitos de memoria) representan un avance o un retroceso para la síntesis de imágenes?

En cuanto al software, la misma tabla 17-1 nos permite ver que he recurrido al conocido *Windows 98*[2], el sistema operativo más despreciado y también el más necesitado en su tiempo. Puede notar que también recurrí a un sistema *Windows 2000*[3], no tan despreciado. Ambos sistemas son sumamente amigables, a diferencia de UNIX. Además, hay una amplia cantidad de información tanto en libros como en la Internet sobre Windows y cualquier herramienta desarrollada en él.

No solo se requiere de un sistema operativo, también se requiere de un lenguaje de programación y el elegido fue *Delphi*[4]. La razón de esta elección es la simpleza de su lenguaje nativo: Pascal, que en comparación con el lenguaje C es mucho más amigable. El entorno de Delphi también hace que el usuario se olvide de la administración de las ventanas permitiéndole así dedicarse a la solución de algún problema.

El último elemento de software que falta por considerar es OpenGL[5]. OpenGL, como ya sabemos es una API gráfica muy poderosa para equipo gráfico que permite a los programadores producir imágenes en color de alta calidad a partir de objetos 2D y 3D. La figura 17-2 nos muestra los componentes básicos de OpenGL: *GL* es un conjunto de 120

[2] Windows 98 es una marca registrada de Microsoft Corporation.
[3] Windows 2000 es una marca registrada de Microsoft Corporation
[4] Delphi es una marca registrada de Borland International Inc.
[5] OpenGL es un producto controlado por el Consorcio OpenGL Architectural Review Board (ARB).

comandos básicos para construir cuerpos complejos a partir de primitivas. *GLU* extiende los comandos para agregar la definición de cámaras, vista en perspectiva y objetos geométricos comunes como cilindros, esferas, etc. *GLUT*, a diferencia de los anteriores, proporciona comandos para generar una interfaz con el usuario. Esta interfaz implica ventanas y control de eventos desde el teclado o el ratón de manera simple y así, libera al programador de la compleja tarea de generar y administrar ventanas.

Evaluación de modelos o aplicaciones

Tal vez no parezca claro, pero tenemos otra pregunta que hacer. Veamos, tenemos los siguientes modelos, los cuales en conjunto, dan una bandera ondeante:

1. Modelo de movimiento.
2. Modelo de superficie.
3. Dos modelos de sombreado, uno es verde mate y otro representa los colores de la bandera.

El modelo de movimiento tiene que ver con el cálculo de posiciones, el modelo de superficie ubica algunos puntos muestra de la bandera en la pantalla del ordenador. Estos puntos muestra corresponden a los vértices de facetas triangulares, mismos que deben ser coloreados por los modelos de sombreado.

Estos tres modelos no se evalúan como tal, en vez de eso se evalúan como las siguientes aplicaciones:

1. Modelo de movimiento.
2. Modelo de superficie de alambre, que implica al modelo de movimiento, a un modelo de superficie y a un modelo de sombreado que dibuja el perímetro de los polígonos que definen la superficie.
3. Modelo de superficie verde mate, que también implica al modelo de movimiento, a un modelo de superficie y a un modelo de sombreado que dibuja triángulos verdes bajo los conceptos de luz y sombra.
4. Modelo de superficie de bandera, que implica al modelo de movimiento, al modelo de superficie y a un modelo de sombreado que nos da la textura de la bandera bajo los conceptos de luz y sombra.

Observe que tenemos cuatro aplicaciones que dibujan una bandera en la pantalla del ordenador excepto por la primera aplicación. Entonces la pregunta obligada es:

¿Cómo se ve afectado el rendimiento del sistema por cada una de las cuatro aplicaciones arriba mencionadas?

17-3 Prueba al modelo de movimiento

Del capítulo 15 tenemos la definición del modelo de movimiento: una bandera construida por partículas que se enlazan con resortes y amortiguadores. La bandera no se mueve por sí sola, hacen falta dos fuerzas externas, la fuerza de gravedad que la hace caer y la fuerza del viento que le hace ondear. El viento también está formado por partículas que actúan sobre las partículas de la bandera de la misma manera en que un fluido actúa sobre un sólido.

Entonces, por el párrafo anterior, el modelo de movimiento para cada partícula es muy simple:

$$a = \frac{\sum Fuerzas}{m}$$
$$v_2 = a\Delta t + v_1 \qquad (17\text{-}1)$$
$$x_2 = x_1 + \frac{\Delta t}{2}(v_2 + v_1)$$

La primera de las tres ecuaciones nos da la aceleración de la partícula en función de la fuerza neta que actúa sobre ella y de su masa. La segunda ecuación nos permite obtener la velocidad de la misma partícula en función de la aceleración y de la velocidad que tenía la partícula previamente. La última ecuación nos habla de la posición de la partícula. Por supuesto, aún falta considerar las operaciones necesarias para detectar colisiones entre las partículas del viento y las partículas de la bandera, sin embargo, para simplificar la explicación, es posible dejar fuera las colisiones.

Estas tres ecuaciones representan una solución numérica por el método de Runge-Kutta de primer orden. Una de las desventajas del método implica su inestabilidad cuando los valores Δt son muy grandes. A consecuencia se escogen incrementos de tiempo más pequeños y es así que se incrementa el número de veces que debe evaluarse el modelo de movimiento. En resumen, una solución válida para aceleración, velocidad y posición de una partícula, viene dada por 10 iteraciones antes de mostrar cualquier resultado en pantalla.

Por el párrafo anterior, por cada diez iteraciones el programa generará un cuadro. Al final de diez minutos, aproximadamente, contaremos un cierto número de cuadros generados por segundo.

El caso del modelo de movimiento es especial ya que debieran contarse solo las iteraciones, sin embargo, por defectos de la prueba, por cada diez cálculos de las variables de aceleración, velocidad y posición, se muestra un cuadro vacío.

Finalmente, las unidades en las cuales se mide el desempeño de los diversos sistemas es el de cuadros por segundo o *fps*. La tabla 17-2 muestra los resultados al evaluar el modelo de movimiento en distintos equipos.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS 6326 PCI	13,389
Pentium II	330	S3VIRGE DX/GX PCI	27,502
Pentium III	650	SiS 6326 PCI	43,948
Pentium III	650	SiS 6326 AGP	43,731
Pentium III	650	nVidia TNT2 M64 AGP	58,321

Tabla 17-2. Resultados de las pruebas al modelo de movimiento.

17-4 Pruebas al modelo de movimiento de superficie de alambre.

Este modelo implica lo siguiente: por cada diez iteraciones se dibuja en una ventana de 640x480 pixeles un modelo de alambre que representa a las primitivas que forman a la bandera. Los resultados se muestran en la tabla 17-3.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS 6326 PCI	11,222
Pentium II	330	S3VIRGE DX/GX PCI	22,865
Pentium III	650	SiS 6326 PCI	36,076
Pentium III	650	SiS 6326 AGP	38,378
Pentium III	650	nVidia TNT2 M64 AGP	58,287

Tabla 17-3. Resultados de las pruebas al modelo de superficie de alambre.

17-5 Pruebas al modelo de movimiento de superficie verde mate

Este modelo implica lo siguiente: por cada diez iteraciones se dibuja en una ventana de 640x480 pixeles un modelo de superficie verde mate que representa a la bandera. Los resultados se muestran en la tabla 17-4.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS 6326 PCI	10,082
Pentium II	330	S3VIRGE DX/GX PCI	20,880
Pentium III	650	SiS 6326 PCI	32,339
Pentium III	650	SiS 6326 AGP	35,019
Pentium III	650	nVidia TNT2 M64 AGP	59,516

Tabla 17-4. Resultados de las pruebas al modelo de superficie verde mate.

17-6 Pruebas al modelo de superficie de bandera.

Este modelo implica lo siguiente: por cada diez iteraciones se dibuja en una ventana de 640x480 pixeles un modelo de superficie de bandera de la Facultad de Ingeniería de la UNAM. Los resultados se muestran en la tabla 17-5.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SIS 6326 PCI	5,738
Pentium II	330	S3VIRGE DX/GX PCI	13,085
Pentium III	650	SIS 6326 PCI	20,444
Pentium III	650	SiS 6326 AGP	21,470
Pentium III	650	nVidia TNT2 M64 AGP	31,346

Tabla 17-5. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.

17-7 No se pierda...

No se pierda nuestro próximo capítulo en donde analizaremos los resultados obtenidos en las pruebas y concluiremos el presente trabajo escrito.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium III	650	SiS 6326 P	20,444
Pentium III	650	SiS 6326 A	21,47
Pentium III	650	nVidia TNT2 M64 AGP	31,346

Tabla 18-2. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS 6326 PCI	5,738
Pentium II	450	S3VIRGE DX/GX PCI	13,085
Pentium III	650	SiS 6326 PCI	20,444

Tabla 18-1. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.

Capítulo 18 Conclusiones

Como decía Roy Disney, "la historia de la animación es la historia de una relación dinámica entre el hombre y la máquina...". Tal relación es una consecuencia de nuestra necesidad de generar imágenes y animaciones más realistas. Para poder satisfacer esta necesidad hemos forzado el desarrollo de los cinco componentes principales de una computadora, ya sea PC o estación de trabajo:

- Procesador.
- Memoria.
- Periféricos.
- Subsistema gráfico.
- Controlador de memoria y controlador de entrada-salida.

18-1 Comentarios sobre los resultados

Vamos entonces a tratar de responder a las tres preguntas que nos hemos planteado. La primera pregunta era:

¿Los cambios tecnológicos en el procesador principal y en la placa base de una computadora representan un avance o un retroceso para la síntesis de imágenes?

Los primeros tres equipos referidos en la tabla 17-5 y que ahora son mostrados en la tabla 18-1 nos ayudarán a responder la primer pregunta.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium MMX	200	SiS 6326 PCI	5,738
Pentium II	330	S3VIRGE DX/GX PCI	13,085
Pentium III	650	SiS 6326 PCI	20,444

Tabla 18-1. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.

Los tres modelos de computadora de la tabla 18-1 emplean un subsistema gráfico que se instala en una ranura PCI y aunque diferente uno de ellos[1], el rendimiento de los tres subsistemas es muy similar.

Podemos en este momento considerar un valor de cuadros por segundo que define a un sistema de cómputo como bueno o malo para la síntesis de imágenes. Ese valor es de 30 cuadros por segundo.

De la tabla 18-1 puede notar que con cada cambio en la tecnología también se presenta un rendimiento creciente, no obstante, ni siquiera se logran los 24 cuadros por segundo de una sala de cine.

La segunda pregunta a responder es:

Los cambios tecnológicos en el subsistema gráfico (procesador de video y circuitos de memoria) ¿representan un avance o un retroceso para la síntesis de imágenes?

La tabla 17-5 también nos da la respuesta a nuestra segunda pregunta, ahora bien, los datos que nos interesan son los correspondientes al sistema Pentium III y que muestro en la tabla 18-2.

Procesador		Subsistema gráfico	Resultados [fps]
Tipo	Reloj [MHz]	Modelo	
Pentium III	650	SiS 6326 PCI	20,444
Pentium III	650	SiS 6326 AGP	21,47
Pentium III	650	nVidia TNT2 M64 AGP	31,346

Tabla 18-2. Resultados de las pruebas al modelo de superficie que representa la bandera de la UNAM.

La tabla nos muestra un aumento en el rendimiento del sistema al mejorar la tecnología del subsistema gráfico. Note una ligera diferencia en los dos subsistemas Sis6326. No es mucha debido a que solo es necesario pasar órdenes de dibujo por el camino memoria-

[1] La razón de que uno de los subsistemas gráficos sea diferente es debido a que el propietario no tenía suficiente confianza en mi habilidad de reconstituir una computadora sin dañarla y no puedo culparlo.

procesador-tarjeta de video. La diferencia que si podemos observar en la tabla 18-2 radica en el subsistema gráfico nVidia TNT2. Tal subsistema contiene un procesador gráfico que se encarga de los procesos de la tubería gráfica así como del proceso de texturizado, liberando algo de carga del CPU. Observe que el sistema desarrollado por nVidia logra una calidad muy aceptable al generar hasta 31 cuadros por segundo. Definitivamente se está avanzado en el mundo de la síntesis de imágenes.

La última pregunta que nos hemos planteado es:

¿Cómo se ve afectado el rendimiento del sistema por cada uno de los cuatro modelos: de movimiento y de superficies?

La tabla 18-3 nos muestra el rendimiento del sistema Pentium III con subsistema gráfico SiS. Observe el decaimiento en la capacidad del sistema para efectuar una tarea cada vez más compleja.

Modelo de superficie	Resultados
Modelo de movimiento	43,731
Modelo de superficie de alambre	38,378
Modelo de superficie verde mate	35,019
Modelo de superficie con bandera de la UNAM	21,47

Tabla 18-3. Rendimiento de un sistema basado en Pentium III con subsistema gráfico SiS 6326 (AGP)

La tabla 18-4 nos muestra que un subsistema nVidia mantiene en todo caso un rendimiento aceptable, no obstante, decae cuando debe realizar la tarea más compleja de todas, representar la bandera de la UNAM.

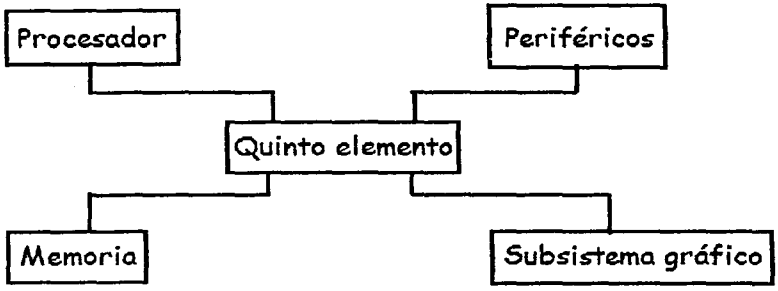
Modelo de superficie	Resultados
Modelo de movimiento	58,321
Modelo de superficie de alambre	58,287
Modelo de superficie verde mate	59,516
Modelo de superficie con bandera de la UNAM	31,346

Tabla 18-4. Rendimiento de un sistema basado en Pentium III con subsistema gráfico nVidia TNT2

18-2 Comentarios finales

La síntesis de imágenes, entre otras actividades, crea la necesidad de equipos más eficientes y veloces en cuanto al manejo que le dan a la información. En consecuencia podemos enunciar el siguiente teorema: "*Para una misma marca de computadoras monoprocesador, la que es más reciente en el mercado, es la más eficiente y veloz*". Por supuesto, los teoremas siempre son proposiciones que deben comprobarse.

Repasemos entonces los avances que hemos visto en el diseño de computadoras monoprocesador para la síntesis de imágenes. En el capítulo 3, notamos que los tres sistemas implicados tienen la siguiente configuración general:



Así que en la tabla periódica de las computadoras existen cuatro elementos y uno quinto. He dejado este nombre de quinto elemento tan lleno de eufenismo debido a que cada empresa de computadoras lo fabrica diferente y le da su propio nombre. Por ejemplo, en el mundo del PC, este quinto elemento es en realidad una pareja de circuitos llamados GMCH (Controlador de Gráficos y Memoria) e ICH (Controlador de Entrada y Salida). Para mayor información debiera consultar el capítulo de Arquitectura especializadas.

En cuanto al procesador, hemos visto que se trata de un híbrido entre la arquitectura Harvard y la arquitectura von Newmmann. A su vez, estos circuitos incluyen una "*línea de montaje*" o más bien, una tubería de ejecución superescalar, o sea, pueden terminar la ejecución de más de una instrucción por ciclo de tubería. Las últimas versiones de estos circuitos ya no incluyen predictor de saltos por lo que siempre están en el camino correcto. Otra situación que también hemos notado, es la longitud de palabra del procesador, Intel trabaja con palabras de 32 bits, en tanto que MIPS y Ultra Sparc trabajan con palabras de 64 bits. Finalmente, los procesadores logran ya los 2 GHz como frecuencia de operación; por cierto, la meta de Intel es lograr un procesador operando a 20 GHz.

El último comentario en el párrafo anterior nos deja ver que el mundo de la computadora monoprocesador aún tiene futuro comercial; no obstante, cuántos de nosotros no desearíamos tener una computadora multiprocesador como la Origin 2000 de Silicon Graphics: Esta computadora puede simular una prueba de aerodinámica en un túnel de viento en tiempo real.

Otro elemento que quiero comentar es el de la memoria RAM; 2 GB/s es un ancho de banda sorprendente aunque también lo es el que podamos colocar en un sistema hasta 2GB de RAM. ¿Cuánto más se desarrollarán estos circuitos? Bueno, si alguna vez paseamos por una tienda de computadoras, deberíamos observar las tarjetas de video. Éstas tienen un disipador de calor montado sobre los circuitos de memoria.

Finalmente, las tarjetas gráficas. Empresas como nVidia toman la ventaja de poder diseñar ellas mismas sus circuitos. Los procesadores gráficos que estas empresas fabrican son en realidad varios procesadores montados e interconectados en un mismo encapsulado; además, toman ventaja de tecnologías como la arquitectura Harvard, una memoria caché asociativa, unidades de punto flotante que trabajan igual que en un DSP, la línea de montaje de instrucciones, etc. Y eso que aún debemos considerar el desarrollo de los programas que hacen funcionar a estos circuitos: los controladores de dispositivos.

El software

El software tiene gran influencia en el desempeño de un equipo y por más ineficiente que sea, no debemos olvidar que la persona o personas que los desarrollaron pasaron muchas

horas pensando y evaluando posibles soluciones. La diferencia entre una solución y otras muchas está en función de los conocimientos y experiencia que cada individuo posee. Así, mi modelo de movimiento, aunque es ineficiente, se trata de mi primera solución en el largo camino de experiencias que deben adquirirse programando este tipo de simuladores.

Ahora, el porqué emplee un entorno de desarrollo basado en Pascal, se lo debo a ciertas dificultades con:

- La sintaxis del lenguaje C.
- La definición de parámetros y directivas del compilador. En Delphi esto ya se encuentra definido.

PC y no O2

O_2 es un equipo de Silicon Graphics y representa su mejor solución, y quizá la mejor solución del mercado, al problema de sintetizar imágenes y animaciones a bajo costo y de calidad.

¿Ha leído bien en el párrafo anterior? ¿Bajo costo? Bueno, O_2 es un equipo sumamente costoso y se requiere que los usuarios de esta máquina reciban entrenamiento de la empresa fabricante para lograr el máximo aprovechamiento. Por supuesto, equipo, entrenamiento, reparación y mantenimiento, refacciones, y demás, tienen un costo elevado; además, considere que si iniciamos una empresa de publicidad o de juegos de video, recurriremos a la PC, la cual es una herramienta que cuesta mucho menos que cualquier estación de trabajo.

El sistema de partículas

Yo le tengo fe a mi sistema de partículas ya que sus usos pueden extenderse a:

- La representación de gelatinas colisionando con cuerpos sólidos o entre ellas.
- También es posible extender su aplicación a la representación de sistemas planetarios y galaxias.
- Los sistemas de partículas se pueden aplicar al diseño de modelos meteorológicos para predecir tormentas, huracanes, tornados y otros fenómenos naturales.
- Es posible que un modelo de partículas logre predecir el comportamiento de la armadura de un edificio durante un terremoto.
- También es posible representar fluidos, ya sea que escurran o interaccionen entre ellos.

Lo nuevo y lo viejo

No es posible jactarme de haber incluido en mi trabajo escrito las últimas tecnologías, ya que éstas evolucionan tan rápido que sólo en un mes pueden haber sido desplazadas por otras nuevas soluciones. Mi consuelo es que muchas de estas nuevas soluciones que son comerciales, tienen sus antecedentes en viejas tecnologías que sólo estaban disponibles para los científicos y diseñadores más especializados.

Despedida

Espero que mi trabajo no haya ofendido mucho a los intelectos que le han leído. Para mí fue, en algunos casos un placer conocer todo este mundo y apabullarme ante su grandeza y diversidad. El suplicio consistía en escribir sobre esta tierra que había sido desconocida para mí. Mi trabajo y mi historia quedan entonces a su disposición y en espera de nuevas generaciones que continúen el camino que he elegido.

Bibliografía

Libros

Ramón García
Pelayo y Gross
"*Pequeño Larousse Ilustrado*"
1995

Tom Prideaux
"*El Hombre de Cro-Magnon*"
TIME-LIFE International de México, S.A. de C.V.
1981

Donald Hearn
M. Pauline Baker
"*Gráficas por Computadora*"
Prentice Hall Hispanoamericana, S.A.
Segunda edición en español
1995

David A. Patterson
John L. Henneesy
"*Organización y Diseño de Computadoras, La Interfaz Hardware/Software*"
McGraw Hill
2ª edición
1995

Foley
Van Dam
Feinderm Hughes
"*Computer Graphics, Principles and Practrice*"
2ª edición
1995.

Mark H. Horestein
"*Microelectronic, Circuits and Devices*"
Prentice Hall
2ª edición
1996
pg986.

David F. Rogers
J. Alan Adams
"*Mathematical Elementes for Computer Graphics*"
McGraw-Hill Publishing Company
2° edición
1990

OpenGL and OS/2
"*All About NURBS*"
Escrito Por Perry Newhook
1990

Herbert Schildt
"*Turbo C/C++ Manual de Referencia*"
McGraw Hill
1992.

Murphy-Smoot,
"*Física, Principios y Problemas*"
CECSA
4° ed.
1986

Ferdinand. P. Beer
E. Russell Johnston Jr.
"*Mecánica Vectorial para Ingenieros*"
McGraw Hill
5° ed.
1990

Francisco J. Rodríguez Ramírez
"*Dinámica de Sistemas Físicos*"
Trillas
1989

Michelle M. Manning
"*Delphi 2, Guía Oficial de Borland*"
Prentice Halll
1996

Louis Nashelsky
"Fundamentos de Tecnología Digital"
Limusa
1993

Referencias técnicas

"Intel ® 815 chipset family: Graphics and Memory Controller Hub"
Intel
Junio del 2000

"*UPA-Sun's High Performance Graphics Connection. Technical White Paper*"
Sun Microsystems
1999.

"*TMS626162, TMS626812 16M-bit Synchronous DRAMs Technical Reference*"
Texas Instruments
1996.

"*Accelerated Graphics Port Interface Specificaction*"
Intel Corp.
Revisión 2.0
Mayo de 1998

"*Mips Microprocessor, User's manual*"
Mips technologies Inc.
Versión 2.0
1996

Sitios en Internet

http://home.cfl.rr.com/eaa/MemoryTypes.htm
Referencia general de circuitos de memorias RAM

http://www.conozcasuhardware.com/articulo/ramcant1.htm#presentation
El sitio es un conjunto de páginas con referencias técnicas sobre los componentes de una PC: memorias, procesadores, placas base, monitores, etc.

http://serdis.dis.ulpgc.es/~atrujill/iga/
El sitio corresponde a una actividad llamada IGA (Información Gráfica Aplicada)
Es un sitio que contiene diapositivas de un curso de OpenGL: Librería auxiliar, transformaciones, proyecciones, color e iluminación, primitivas, texturas.

http://www.cs.indiana.edu:800/cstr/search?motion+synthesis
Este sitio contiene varios documentos de publicación periódica de diversas universidades de los Estados Unidos de Norteamérica. El sitio es patrocinado por la universidad de Indiana.

http://www.inria.fr/recherche/equipes/projets_theme3.en.html
Este sitio es la página de proyectos de investigación correspondientes al Tema 3: computación. El tema tres está dividido en desarrollo de herramientas hardware y herramientas software para computadores de escritorio. La orientación del desarrollo en software es la síntesis de imágenes para la medicina, la industria cinematográfica, la Arquitectura, Ingeniería, etc.

http://www.sgi.com/developers/library/resources/openglxp.html#glw_widget
Este sitio es una colección de recursos, técnicas y repuestas sobre la programación con OpenGL y GLX en el entorno de desarrollo X Windows.
Autor:Marc Romankewicz, Applications Consulting.
Versión inicial: 9/14/98. último cambio: 10/13/98.

http://www.intel.com
Esta es la página principal de Intel. La página proporciona acceso a anuncios publicitarios de sus últimos productos. También se incluye un buscador a través del cual puede solicitar toda clase de información técnica o de eventos.

http://www.finalfantasy.com/
Sitio en la red que contiene fotografías e información general sobre la producción de la película

Tesis y Reportes de investigación

John Christensen
"Managing Desing Complexity: Using Stochastic in the Production of Computer Graphics"
Harvard
Junio 1995

Gilles Rebunne
Mathieu Desbrun
Alan Barr
Marie-Paule Cani
"Interactive Resolution Animation of Deformable Models"
iMagis
1999.

Publicaciones periódicas

Will Smith
"GeForce 3"
Maximum PC,
vol. 6,
No. 64
Imagine
Abril20001, pg. 32.

Referencias

Pentium IV es una marca registrada de Intel, Corp.

Desktop 815 e Intel Pentium III son marcas registradas de Intel Corp.

Mips R10000 es una marca registrada de Mips Technologies, Inc.

Power PC es una marca registrada de Motorla, Inc.

Origin 2000 es una marca registrada de Silicon Graphics, Inc.

O_2 es una marca registrada de Silicon Graphics, Inc.

Power Iris es una marca registrada de Silicon Graphics Inc.

UPA (Ultra Port Architecture) es una marca registrada de Sun Microsystems.

Sequent Symmetry es una marca registrada de Sequent Computer Systmes, Inc.

Trimedia es una marca registrada de Phillips Inc.

GeForces es una marca registrada de nVidia Inc.

Delphi es una marca registrada de Borland International Inc.

OpenGL es un producto controlado por el Consorcio OpenGL Architectural Review Board (ARB).