Cálculo de Reflexiones, Refracciones y Sombras

Computación Gráfica

Alumnas:

Mariana Eveleens

Silvia Isabel Goya

Director:

Dr. Gustavo Rossi

Codirector:

Lic. Gustavo Ariel Patow

Dedicatorias

A nuestros padres, hermanos, amigos y a Guillermo que nos acompañaron durante estos años.

Agradecimientos

Al Lic. Gustavo Patow, por el apoyo y la colaboración brindados incondicionalmente a lo largo de estos dos años.

Al Dr. Gustavo Rossi.

Tabla de contenidos

1. Introducción

1.1. Presentación del problema

El problema está enmarcado dentro de la Computación Gráfica, y consiste en lograr la mayor sensación de fotorrealismo a través de la representación de imágenes 3D en computadora.

La mejor aproximación al modo en que trabaja el mundo real es la técnica de trazado de rayos. Pero hay un problema con esta simulación directa, y es la cantidad de tiempo que llevaría producir una imagen. Consideremos que cada fuente de luz en la imagen genera millones de rayos dirigiéndose en direcciones levemente diferentes donde muchos de estos rayos alcanzan objetos que nunca veremos, ni siquiera indirectamente. La solución a este problema es seguir los rayos hacia atrás en vez de hacia delante, es decir, desde el observador hacia las fuentes.

Se han propuestos otros algoritmos a lo largo de los años para el cálculo de reflexiones, refracciones y sombras. El primer algoritmo en proponer una solución rápida para calcular reflexiones fue el de Jim Blinn y Martin Newell en 1976 [Blinn76], y está basado en la utilización de mapas de entorno, uno para cada objeto reflectivo. El principal problema de esta aproximación es que sólo modelaba correctamente objetos colocados infinitamente lejos del centro de proyección del mapa de entorno, resultando una aproximación muy pobre en otro caso.

En 1995, Gustavo Patow, desarrolló los mapas de entorno Z-Bufferados [Patow95], en donde se guarda para cada pixel una lista de z's (distancias al centro de proyección del mapa), además de su color. Pero su técnica posee un grave problema (no notado por el autor en ese momento), que es el tratar la información en los pixeles como puntos, sin guardar ningún tipo de conectividad entre ellos, por lo que reflexiones erróneas pueden ser computadas si la densidad del muestreo no es lo suficientemente alta. Además, solo fue utilizado para calcular con mayor exactitud una única reflexión. El problema principal con estos mapas de entorno Z_Buffereados, al igual que con en el modelo original de Blinn, es que por problemas de ocultamiento de algunas reflexiones por objetos más cercanos al centro de proyección del mapa, no se pueden calcular las interreflexiones.

Con la espectativa de resolver los problemas anteriormente enumerados es que encaramos el algoritmo que presentamos en este trabajo. En dicho algoritmo se guarda una lista ordenada de distancias z para cada pixel, en una estructura de datos A_Buffer [Carpenter84]. Con esto se soluciona el problema del ocultamiento de las reflexiones debido a errores en la conectividad de los objetos. Además, la estructura del mapa de entorno A_Buffereado, al guardar información de toda la escena, hace que sea suficiente tener un mapa de

entorno único centrado en el observador, en lugar de uno para cada objeto Reflectivo. Esto supone un considerable ahorro en los requerimientos de memoria y cálculo de mapas de entorno. Así mismo nos permite calcular sombras sin necesidad de usar algoritmos de sombra de precisión de objeto, shadow buffers o incluso volúmenes de sombra, ahorrando nuevamente un tiempo de cálculo considerable. Explotar la coherencia espacial implícita en esta estructura de datos será la clave de la optimización propuesta.

1.2. Motivación y objetivos

Es nuestra motivación lograr mejorar el fotorrealismo de la presentación de una escena con costos de implementación y ejecución razonable solucionando algunos problemas que se presentaban con algoritmos anteriores, debido a que actualmente han crecido notoriamente las aplicaciones que utilizan la graficación por computadora. Nos interesa, en la medida de nuestras posibilidades, hacer un aporte a la investigación en esta rama de la computación.

Es nuestro objetivo implementar un nuevo algoritmo de cálculo de reflexiones, refracciones y sombras desde un único mapa de entorno **A_Bufferado** que solucione los problemas de interreflexión y aliasing de los algoritmos anteriores.

1.3. Materiales y herramientas

Se trabajó con un equipo 486 con 32 Mb de RAM. El programa se implementó en Delphi 3 corriendo bajo plataforma Windows NT o 95. Las imágenes presentadas en el capítulo 11 (Resultados) fueron generadas por nuestra aplicación, y para mostrarlas generamos imágenes en formato BMP.

2. Computación Gráfica

La computación gráfica es una rama de la informática, pero su atractivo se extiende mucho más allá de las fronteras de dicho campo relativamente especializado. En su corta vida ha atraído a algunas de las personas más creativas del mundo, provenientes de todas las disciplinas: arte, ciencia, música, danza cinematografía. La mejor forma de dar a conocer la emoción y la diversidad de la computación gráfica es a través de sus aplicaciones, veamos algunas de ellas con mayor detalle.

2.1. Algunas aplicaciones de la Computación Gráfica

La computación gráfica se emplea en la actualidad en varias áreas de la industria, los negocios, el gobierno, la educación y el entretenimiento. La lista de aplicaciones crece a medida que las computadoras con capacidades gráficas se convierten en artículos de consumo. A continuación veremos algunas de esas áreas.

* **Interfaces con el usuario**: la mayoría de las aplicaciones que se ejecutan en PC o estaciones de trabajo tienen interfaces con el usuario que se apoyan en sistemas de ventanas para administrar múltiples actividades simultáneas, así como seleccionar elementos de los menús, iconos y objetos de la pantalla. Los programas de procesamiento de texto y hoja de cálculo son aplicaciones típicas que aprovechan estas técnicas de interfaz con el usuario. La computación gráfica desempeña un papel integral tanto en las funciones de entrada como en las de salida de las interfaces con el usuario.

* **Graficación interactiva en los negocios, la ciencia y la tecnología**: una de las aplicaciones más usuales de los gráficos es crear gráficos bidimensionales y tridimensionales de funciones matemáticas, físicas y económicas; histogramas y diagramas de barras y circulares; diagramas de programación de actividades; diagramas de inventario y producción, etc. Todos estos gráficos se usan para presentar de manera significativa y concisa las tendencias y los patrones extraídos de los datos, para así esclarecer fenómenos complejos y facilitar la toma de decisiones informada.

* **Cartografía**: la computación gráfica se emplea para producir representaciones precisas y esquemáticas de fenómenos geográficos y de otro tipo, a partir de datos de mediciones. Como ejemplos están los mapas geográficos, de relieve, de exploración para perforaciones y minería, oceanográficos, meteorológicos, de contorno y de densidad de población.

* **Medicina**: la computación gráfica tiene una función cada vez más importante en campos como la medicina de diagnóstico y la planificación de

cirugías. Los cirujanos usan los gráficos como auxiliares para guiar instrumentos y determinar precisamente donde hay que eliminar tejidos enfermos. Como ejemplo de su utilización en el diagnóstico, los diagnosticadores pueden sintetizar una representación tridimensional del cerebro humano. El usuario puede manipular el modelo en forma interactiva para revelar información detallada acerca de la condición del cerebro. Otra aplicación interesante se presenta al observar gráficos tridimensionales de los datos obtenidos al iluminar tejido vivo con luz láser, técnica conocida como biopsia óptica. Por ejemplo un hígado canino normal muestra huellas ópticas muy distintas que un hígado canino canceroso, lo que ofrece la esperanza de un diagnóstico no quirúrgico.

* **Bosquejo y diseño asistido por computadora**: en el diseño asistido por computadora los usuarios emplean la graficación interactiva para diseñar componentes y sistemas de dispositivos mecánicos, eléctricos, electromecánicos y electrónicos, incluyendo estructuras tales como edificios, carrocerías de automóviles, cascos de aviones y barcos, pastillas de circuitos integrados a escalas muy grande y redes telefónicas y de computadoras.

* **Simulación y animación para visualización científica y entretenimiento**: los filmes de animación generados por computadora y la presentación del comportamiento variable en el tiempo de objetos reales y simulados constituyen una herramienta cada vez más común para la visualización científica y del campo de la ingeniería. Podemos utilizar estos gráficos para estudiar entidades matemáticas abstractas o modelos matemáticos de fenómenos como el flujo de fluidos, la relatividad, las reacciones nucleares y químicas, la función de los órganos y el sistema fisiológico, y la deformación de estructuras mecánicas sujetas a diferentes tipos de carga. Otra área de tecnología avanzada es la producción de efectos especiales de gran elegancia para filmes. Existen mecanismos complejos que permiten modelar los objetos, representar luces y sombras.

2.2. Breve resumen de la evolución de la computación gráfica

Las interfaces gráficas han sustituido a las interfaces textuales como el medio habitual para la interacción usuario-computadora. Hasta finales de la década de 1980, la mayor parte de las aplicaciones de la computación gráfica tenía que ver con objetos bidimensionales; las aplicaciones tridimensionales eran relativamente raras, ya que el software de este tipo es más complejo que el bidimensional y porque se requiere mucho poder de cálculo para producir imágenes seudorrealistas. Por lo tanto, hasta no hace mucho tiempo la interacción en tiempo real entre el usuario y los modelos tridimensionales e imágenes seudorrealista sólo era factible en costosas estaciones de trabajo de alto rendimiento que usaban hardware gráfico de propósito especial. El espectacular avance de la tecnología de semiconductores VLSI, responsable

del advenimiento de microprocesadores y la memoria de bajo costo, fue lo que condujo, a principios de los ochenta, a la creación de interfaces con PC basadas en gráficos bidimensionales de mapas de bits. La misma tecnología que ha permitido crear, menos de una década después, subsistemas con unos cuantos chips que llevan a cabo animaciones tridimensionales en tiempo real con imágenes sombreadas en color de objetos complejos, por lo general descriptos por miles de polígonos. Estos subsistemas se pueden añadir como aceleradores tridimensionales a las estaciones de trabajo o incluso a las PC que emplean microprocesadores comerciales. Es obvio que el explosivo crecimiento de las aplicaciones tridimensionales será paralelo al crecimiento actual de las aplicaciones bidimensionales. Así mismo, las áreas como la presentación fotorrealista, que en una época se consideraba como algo exótico, ahora forman parte de la tecnología de alto nivel y son comunes en el software gráfico y cada vez con mayor frecuencia en el hardware gráfico.

Gran parte del trabajo para crear una comunicación gráfica efectiva, ya sea en dos o en tres dimensiones, se relaciona con el modelado de los objetos cuyas imágenes queremos producir. El sistema gráfico actúa como intermediario entre el modelo de aplicación y el dispositivo de salida. El programa de aplicación es responsable de crear y actualizar el modelo con base en la interacción con el usuario; el sistema gráfico lleva a cabo la parte más rutinaria y mejor comprendida del trabajo, creando vistas de objetos y transmitiendo los eventos del usuario a la aplicación. La literatura cada vez más abundante sobre los distintos tipos de modelado basado en la física nos indica que la graficación está evolucionando para incluir algo más que la presentación y el manejo de la interacción. Las imágenes y las animaciones ya no son simples ilustraciones en la ciencia y la ingeniería: se han convertido en parte del contenido científico y de ingeniería y han influido en la forma en que los científicos y los ingenieros llevan a cabo su trabajo cotidiano.

2.3. *Introducción al modelado de objetos en computación gráfica*

El mundo que nos rodea presenta una riqueza que difícilmente el hombre pueda alguna vez igualar, pero no por eso dejará de intentarlo tantas veces como pueda. Las ciencias han tratado, desde su comienzo, de dar una explicación racional a este mundo, exigiendo una complejidad cada vez mayor en los modelos matemáticos utilizados para representarla. Por otro lado, la increíble evolución de las computadoras en las últimas décadas ha permitido realizar cálculos hasta ahora prácticamente imposibles para el hombre, permitiéndole confirmar, mediante la simulación, teorías de otra forma imposibles de verificar. Además, la continua caída de costos de estos equipos ponen al alcance de una gran cantidad de personas una alta potencia de cálculo.

La programación de imágenes tridimensionales requiere un cierto conocimiento de matemática y, en particular, los gráficos 3D exigen el dominio

de los rudimentos del álgebra vectorial. Pero, a medida que se busque un mayor fotorrealismo, será necesario agregar algunos conceptos de física y, obviamente, más matemática avanzada.

En el Apéndice, introducimos algunas definiciones y propiedades elementales del álgebra vectorial que utilizaremos en nuestro trabajo.

3. Mirando en 3D

El proceso de vista tridimensional es más complejo que el de vista bidimensional. En el caso bidimensional, basta especificar una ventana en el mundo bidimensional y un área de vista en la superficie de la vista bidimensional. En teoría los objetos en el mundo se recortan con respecto a la ventana y luego se transforman al área de vista para la presentación. La complejidad adicional de la vista tridimensional es ocasionada en parte por la otra dimensión y en parte por el hecho de que los dispositivos de presentación son bidimensionales.

Una metáfora útil para la creación de escenas tridimensionales es el concepto de la cámara. Imagine que podemos mover la cámara a cualquier posición, orientarla como queremos y, con un disparo del obturador, crear una imagen bidimensional de un objeto tridimensional. Por supuesto la cámara no es más que un programa de computación que produce una imagen en la pantalla y el objeto es una base de datos tridimensional que comprende una colección de puntos, líneas y superficies. Tanto la cámara como el objeto tridimensional tienen su propio sistema de coordenadas que ofrecen una importante independencia de representación, como veremos más adelante en el desarrollo de este capítulo.

Para crear una fotografía se llevan a cabo varias etapas que describiremos genéricamente a continuación:

Proyección: la diferencia entre los objetos tridimensionales y las pantallas bidimensionales se resuelve con la introducción de proyecciones, que transforman objetos tridimensionales a un plano de proyección bidimensional.

Especificación de los parámetros de visualización: es necesario especificar las condiciones en las cuales queremos ver el conjunto de datos tridimensionales del mundo real o la escena que se generará. Con base en las coordenadas del mundo del conjunto de datos, esta información incluye la posición del ojo del observador y la ubicación del plano de observación.

Recorte en tres dimensiones: Hay que eliminar de una escena tridimensional las partes que no son candidatas para la representación final, como por ejemplo las partes de la escena que están detrás de nosotros o se encuentran demasiado lejos para ser claramente visibles. Para esto es necesario recortar con respecto a un volumen de vista.

Presentación: por último, el contenido de la proyección del volumen de vista sobre el plano de proyección, llamado ventanas, se transforma al área de vista para la presentación.

A continuación presentaremos como representar los objetos de una escena, desde sus propios espacios hasta el espacio de la pantalla física.

Como así también las relaciones y transformaciones que existen entre los distintos espacios.

3.1. La representación de objetos en el Espacio del Objeto

Sabemos que todo objeto (o, al menos, su superficie exterior) puede representarse, matemáticamente, como un conjunto de puntos. De esta manera, una cáscara esférica estará dada por todos los puntos **P** del espacio que verifiquen:

$$\mathbf{P} = (x, y, z) \quad \text{tal que} \quad x^2 + y^2 + z^2 = R^2 \quad \text{ó} \quad |\mathbf{P}|^2 = \mathbf{P} \bullet \mathbf{P} = R^2$$

Para una esfera "sólida", los puntos serían $|\mathbf{P}|^2 <= R^2$. Aquí, $|\mathbf{P}|$ significa el módulo del vector **P** y • representa el producto escalar entre vectores.

Esta ecuación es excelente si el objeto que buscamos representar es una esfera. Pero si buscamos representar un automóvil, por ejemplo, encontrar dicha función no es nada fácil. Podría decirse que es casi imposible. Por ello es mucho más fácil aproximar el objeto con un conjunto limitado de puntos, buscando que estos aproximen lo mejor posible a la forma deseada.

Se han desarrollado varios métodos para esto último, pero el que utilizaremos será, dada su simplicidad, el aproximar las superficies mediante triángulos. De esta forma, cualquier superficie curva nos quedará representada por pequeñas caras triangulares unidas por sus aristas. Por supuesto, la calidad de la aproximación dependerá del número y tamaño de dichos triángulos. Obviamente, objetos simples como cubos, pirámides y planos tendrán una representación exacta en términos de triángulos. En cambio, una esfera quedará como un poliedro de muchas caras, que con suerte, se asemejará a una esfera. El error en la representación (la diferencia entre la superficie real y la aproximación poligonal) es visualmente disminuida por un algoritmo de formación interpolativo; y el éxito de estos algoritmos, junto con la facilidad general de componer objetos triangulares, son las razones de la popularidad de esta aproximación. El mayor defecto visual del modelo poligonal es la visibilidad de la linealidad de los distintos polígonos que componen las caras de los objetos tridimensionales. La única forma en que esto puede ser superado es aumentado la resolución poligonal (usando más y más triángulos para representar áreas de alta curvatura). En el lenguaje de la computación gráfica, se dice que la superficie está **facetada**.

En esta aproximación, la geometría de nuestros objetos se definen dando los vértices (o aristas) y las conexiones entre ellos. Por ejemplo, un cubo de 20 unidades de lado y centrado en el origen, se definiría en nuestro código de la siguiente manera:

```
Type
    Vector = Array [0..3] of Double;
Const
    Points: Array [0..7] of Vector =
                        ((10,10,10,1),(10,10,-10,1),
                        (10,-10,10,1),(10,-10,-10,1),
                        (-10,10,10,1),(-10,10-10,1)
                        (-10,-10,10,1),(-10,-10,-10,1))
```

Pero aún falta definir la forma en que estos vértices se conectan entre sí. Para ello podemos guardar en alguna estructura los índices del arreglo de vectores que definen los vértices. Como cada triángulo tiene tres vértices, podemos definir al triángulo como un arreglo de 3 componentes, donde cada componente es el índice del vértice correspondiente en el arreglo de vectores.

```
Type
    PolyLength = 0..2;
    Triangle = record
        Point: array [PolyLength] of word;
    End;
```

En nuestro ejemplo del cubo, los vértices están dados en Points (ver recuadro anterior). Por lo tanto, la definición de los 12 triángulos que forman las caras sería:

```
Faces: array [0..11] of Triangle =
((Point:(1,3,2)),(Point:(1,2,0)),(Point(5,7,3)),(Point(5,3,1)),
((Point(4,5,1)),(Point(4,1,0)),(Point(3,7,6)),(Point(3,6,2)),
((Point(5,4,6)),(Point(5,6,7)),(Point(0,2,6)),(Point(0,6,4)));
```

Como puede verse fácilmente, nuestro cubo está centrado en el (0,0,0,1), la cuarta componente, una vez más, no tiene ningún significado por ahora. A este espacio, en el que hemos definido nuestro objeto, se le llama, usualmente, **espacio del objeto** (Object Space) [Watt92]. De esta forma, definimos cada objeto en su propio espacio, independientemente de lo que ocurre con las definiciones de los otros objetos en el mundo que queremos simular. Su correcta ubicación en el lugar final correspondiente se explicará en la siguiente sección.

3.2. La representación de objetos en el Espacio del Mundo

Hemos decidido asignarle un espacio propio a cada objeto, y lo llamamos Espacio del Objeto. Pero necesitamos definir un espacio en el que estén todos los objetos juntos, colocados en sus respectivas posiciones finales, por ejemplo, la taza arriba del escritorio, al lado de la lámpara. A este conjunto se lo denomina **espacio del mundo** (World Space) [Watt92]. Para lograr esto es necesario definir una transformación, para cada objeto, que lo lleve a su posición final. Obviamente, esto se logrará utilizando una matriz de 4 x 4.

Las únicas transformaciones que nos interesan son las llamadas transformaciones rígidas, compuestas sólo por traslaciones, rotaciones y cambios de tamaño, pero no deformaciones de otro tipo. Esto significa que, si nuestros objetos están definidos mediante triángulos, estos no perderán su carácter triangular y los vértices que lo componen seguirán siendo los mismos. En el lenguaje matemático, decimos que no alteramos la forma, o topología, del objeto.

Puede verse que, para aplicar la transformación a un objeto y llevarlo a su posición final en la escena, debemos aplicarle dicha transformación a todos los vértices que lo componen. La definición de sus triángulos no es alterada en la transformación, por lo que no debemos modificarlos al transformar sus vértices.

Pero esta transformación es particular de cada objeto, por lo que debemos asociarla respectivamente a cada objeto, almacenándola junto a la información geométrica del mismo, en su estructura de datos originales.

Dicha estructura de datos podría ser, genéricamente, algo como:

```
Type
    VectorArray = array [0..800] of Vector;
    PVectorArray = ^VectorArray;
    TriangleArray = array [0..800] of Triangle;
    PTriangleArray = ^TriangleArray;

PolygonalObjectToRender = record
    NumVertex:Word,          {número total de vértices}
    Vertex:PVectorArray;     {arreglo de vértices}
    NumFaces:Word;           {número de caras triángulares}
    Faces:PTriangleArray;    {arreglo de triángulos}
    WorldTransform:Matrix;   {matriz de transformación}
    Surface:Pmaterial;       {características del objeto}
End;
```

En este listado, definimos a los conjuntos de puntos y de triángulos del objeto como dos arreglos monodimensionales. La variable Surface de

momento no posee ninguna utilidad, pero más adelante la utilizaremos para definir características de la forma en que se verá nuestro objeto, si es de metal, madera, plástico, etc. Aquí también se guardan otros datos importantes de la apariencia del objeto.

Gráficamente, podemos decir que hemos transformado nuestro objeto de su propio espacio al Espacio del Mundo. Esto se hizo mediante alguna transformación (World Transform), cuya estructura interna no conocemos (no en este punto), y no nos interesa, pero lo que sí es importante es que tiene como función realizar dicha transformación de un espacio a otro como muestra la figura 3.1. También, que dicha transformación es única para cada objeto que hallamos definido en nuestra escena.

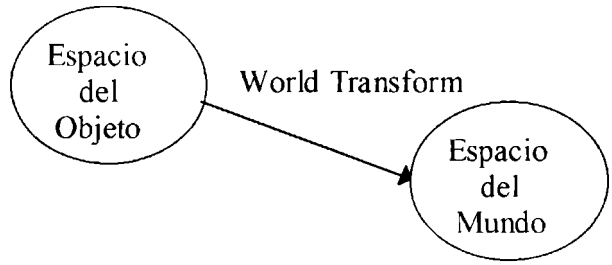


Figura 3.1: La transformación del espacio del objeto al espacio del mundo

3.3. La representación de objetos en el Espacio de la Cámara

La transformación del espacio del mundo al **espacio de la cámara** [Watt92], se realiza simplemente agregando una transformación extra, que llamaremos WorldToCamera, representada por una matriz de 4 x 4.

Este es un espacio utilizado para establecer parámetros y un volumen de visión. Como ya mencionamos anteriormente una cámara virtual es usada como una ayuda conceptual. La cámara puede ser posicionada en cualquier lugar en el espacio del mundo y apuntar hacia alguna dirección. La analogía con el plano de computación, en computación gráfica es el plano de visión sobre el cual la escena es proyectada. En nuestra implementación nosotros requeriremos que:

1. El origen del Espacio de Visión y el centro de proyección coincidan.
2. El normal al plano de visión (la dirección de visión) coincida con el eje Z.
3. La dirección de visión es tal que con la cámara mirando hacia los valores de z negativos, los mismos decrecen a medida que nos alejamos del centro de proyección.

Adoptando un sistema de este tipo y usando una cámara virtual análoga tenemos una cámara que puede ser posicionada en cualquier lugar en el

Espacio del Mundo y apuntar en cualquier dirección. Adicionalmente permite a la cámara rotar en la dirección de visión. Nosotros podemos especificar esto por tres vectores y una posición como mostramos en la figura 3.2:

1. Una posición de la cámara C. Esta posición es también el centro de proyección

2. Un vector de visión **N** (el eje negativo de las Z), normal al plano de visión

3. Un vector hacia arriba V que orienta la cámara acerca de la dirección de visión y establece con **N** la orientación de la ventana del plano de visión dentro del propio plano.

4. La dirección del eje U se define de manera tal que U, V y N formen un sistema de coordenadas.

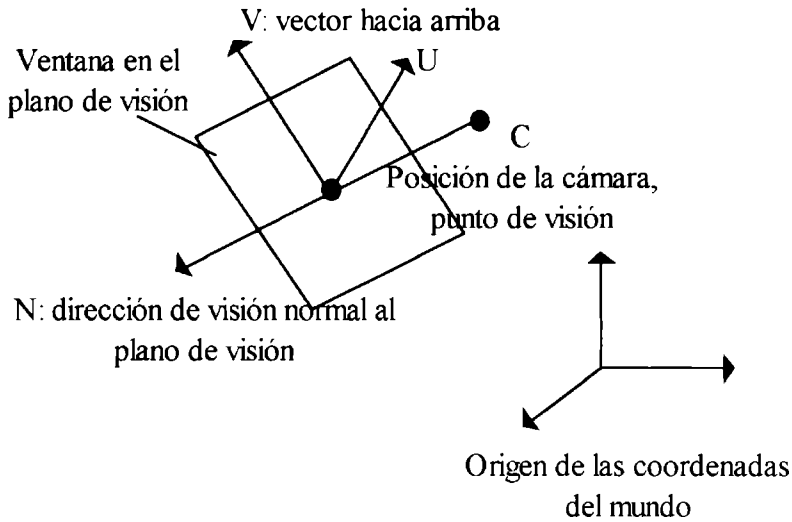


Figura 3.2: establecimiento del sistema de coordenadas del ojo en el espacio del mundo

Hallamos entonces una matriz **E** usando un cálculo simple donde la transformación desde un punto en un espacio de coordenadas del mundo en un espacio de coordenadas de la cámara está dado por:

(xe,ye,ze,1) = (xw,yw,zw,1) **E**

Donde **E** consiste en una traslación **T** y un cambio de bases **B** (ver Apéndice):

E = T B

donde:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Cx & -Cy & -Cz & 1 \end{bmatrix}$$

B rota algún vector expresado en las coordenadas del mundo dentro de las coordenadas de la cámara. En particular los tres vectores axisas del sistema de la cámara (expresados en las coordenadas del mundo) son transformados dentro del espacio de coordenadas de la cámara como (1,0,0), (0,1,0) y (0,0,1). Esto es **B** mapea **U** a (1,0,0), **V** a (0,1,0) y **N** a (0,0,1) o en forma de matriz:

$$\begin{bmatrix} Ux & Uy & Uz & 1 \\ Vx & Vy & Vz & 1 \\ Nx & Ny & Nz & 1 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Como U, V y N son vectores unidades, sabemos que:

$$Ux^2 + Uy^2 + Uz^2 = 1$$

y similarmente para V y N. También U, V y N son mutuamente ortogonales por lo que:

$$Ux.Vx + Uy.Vy + Uz.Vz = 0$$

y similarmente para otras coordenadas. Esto es, **B** esta dada por:

$$\mathbf{B} = \begin{bmatrix} Ux & Vx & Nx & 0 \\ Uy & Vy & Ny & 0 \\ Uz & Vz & Nz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.4. La representación de objetos en el Espacio de la Pantalla

El observador se introduce al definir un punto de vista, que representa el ojo del observador, o la lente de una cámara fotográfica de obturador muy pequeño y a un plano, que representa la pantalla del monitor, o la película de nuestra cámara.

La proyección en perspectiva es necesaria para lograr un cierto realismo visual. En el mundo en el que vivimos, los objetos parecen más pequeños a medida que se alejan de nosotros.

Para simplificar los cálculos, definiremos al punto de vista como en el origen de coordenadas, mirando hacia abajo (el eje negativo de las Z's) e Y hacia arriba. Así el eje de las X's queda apuntando hacia la derecha (ver figura 3.3). Al punto de vista se lo denomina, en el lenguaje de la computación gráfica, Centro de Proyección (Center of Projection o Projection Reference Point, PRP).

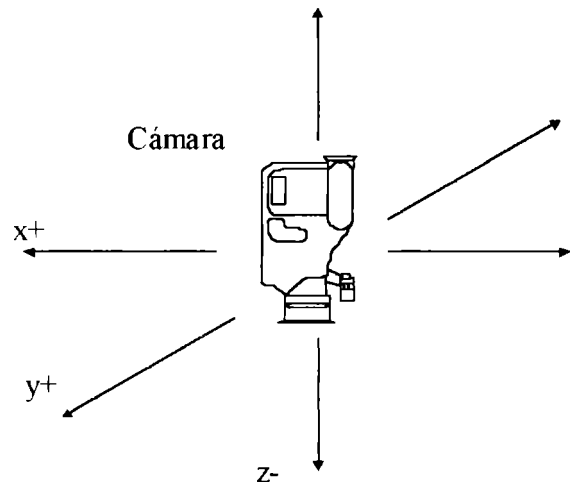


Figura 3.3: Definición del espacio de la pantalla

El punto de vista y el plano de la pantalla definen una pirámide, cuyos lados, si se prolongan hasta el infinito, definen el llamado cono (o pirámide) de observación, puesto que sólo aquello que se encuentra en su interior será visto en la pantalla. Ahora, cuáles objetos son visibles, e incluso qué partes de los que lo son se encuentran en este cono, es parte de un proceso llamado recorte o clipping (ver pág. 32).

Veamos la transformación del Espacio de la Cámara al Espacio de la Pantalla [Foley96], detallando la forma en que se obtiene esta transformación. Ahora, ¿cómo se vería un punto cualquiera P = (x,y,z,1) desde nuestro punto de vista? Para ello nos referimos a la figura 3.4

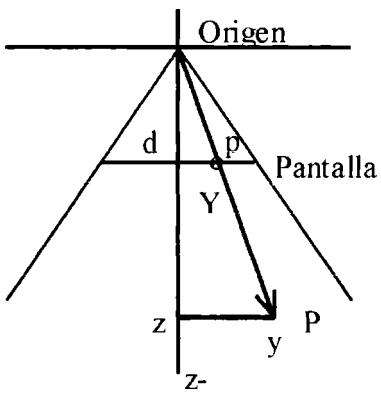


Figura 3.4: Geometría de la proyección en perspectiva

Como vemos, el punto P = (x,y,z,1) se proyecta en el p = (X,Y,Z,1), en la pantalla donde buscamos formar la imagen. Pero sabemos que la pantalla está ubicada a una distancia d del eje de las z's. Por ello, podemos decir que las coordenadas de nuestro punto proyectado p serán (X,Y,d,1). El problema es, ahora, hallar las coordenadas de nuestro punto. Pero ello se reduce a un simple problema de semejanza de triángulos: el triángulo definido por el Origen, el punto P y su componente z, en el eje de los z's negativos, es

semejante al formado por el Origen, la proyección de P en la pantalla, es decir, p y la distancia d.

Usando semejanza de triángulos, podemos decir que y / Y = z / d , lo mismo para x / X. Entonces, despejando Y, obtenemos que Y = y d / z.

Cómo podemos escribir esto utilizando notación matricial?

Recordemos que representamos a nuestros objetos mediante coordenadas homogéneas, lo que significa que siempre serán de la forma (X,Y,Z,1), (ver Apéndice). Por ello, si multiplicamos el vector P por la matriz de proyección en perspectiva, llamémosla Mper, obtendremos el vector P':

$$Mper = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{vmatrix}$$

y, así:

$$\begin{vmatrix} X' \\ Y' \\ Z' \\ W \end{vmatrix} = \mathbf{Mper} \bullet \mathbf{P} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{vmatrix} \bullet \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

donde W = z / d.

Como verán, el vector resultante tiene como cuarta componente a z / d. Si dividimos todas sus componentes por este vector, obtendremos:

(X' / W, Y' / W, Z' / W, 1) = (X, Y, Z, 1) = (x / z/d, y / z/d, d, 1)

que es el resultado correcto.

Finalmente, podemos armar una matriz, llamada CameraToScreen, que hará la transformación desde el Espacio de la Cámara al Espacio de la Pantalla, realizando para ello la transformación en perspectiva. Podemos aprovechar para incluir en esta matriz otra que nos achique la escena, de forma tal que estemos seguros que está toda contenida en el cubo [-1..1, -1..1, -1..1] [1]. Una matriz que haga esto es, simplemente la dada por una matriz de cambio de escala:

Scale(Factor1, Factor2, Factor3) =

[1] Dado que es más fácil hacer recortes con respecto a ciertos conos de observación que con respecto al general, se define el cono de observación de perspectiva por los planos x = z, x = -z, y = z, y = -z, z = $-z_{mín}$ y z = -1.

$$\begin{vmatrix} Factor1 & 0 & 0 & 0 \\ 0 & Factor2 & 0 & 0 \\ 0 & 0 & Factor3 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

por lo que podemos escribir nuestra transformación CameraToScreen:

CameraToScreen = Mper • Scale(1/número, 1/número, 1/número);

Donde número es un número lo suficientemente grande como para asegurarnos la condición de que la escena completa entra en el cubo unitario.

Si se observa con cuidado, se notará que nuestra pantalla ha sido ajustada con dimensiones [-1..1, -1..1]. Pero ninguna pantalla real tendrá estas coordenadas como resolución. Por hoy, hay que llevarla a coordenadas comunes, es decir, una placa VGA común tiene resolución [0..319, 0..199]; una SGA tiene 640 x 480 ([0..639, 0..479]). Esto se puede lograr fácilmente, multiplicando los valores que obtuvimos luego de dividir por W, la cuarta componente que nos queda distinta de 1 al aplicar Mper, por la matriz Mvv3dv, definida como una matriz de escala al tamaño de la pantalla:

Mvv3dv = Scale(ScreenXSize/2, ScreenYSize/2, 1) • Translation(1, 1, 1)

3.5. Mapa de Transformaciones. Caminos Inversos

Pueden presentarse transformaciones mucho más complejas, pero con estas que hemos presentado aquí es suficiente para realizar nuestros primeros cálculos en 3D.

Ahora multiplicar primero por WorldTransform y luego por WorldToCamera es equivalente a multiplicar por una única matriz, que podemos llamar WorkingTransform o Transformación de Trabajo, y estará definida por:

WorkingTransform = WorldToCamera • WorldTransform

La operación • entre matrices significa producto, en ese orden. Las matrices no conmutan.

Lo mismo podemos hacer con WorldToCamera y CameraToScreen, obteniendo como resultado WorldToScreenTransform. Y con WokingTransform y CameraToScreen, armamos la transformación final que llamamos FinalTransform. Por lo tanto, ambas estarán definidas de la siguiente manera:

WorldToScreenTransform = CameraToScreen • WorldToCamera;

FinalTransform = CameraToScreen • WorkingTransform;

Como puede verse, todas estas matrices son de similar nombre pero con diferente función. En realidad, lo que ocurre es que estamos trabajando simultáneamente con cuatro espacios: el Espacio del Objeto, el Espacio del Mundo, el Espacio de la Cámara y, finalmente, el Espacio de la Pantalla, donde se formará la imagen final, luego de la proyección en perspectiva. Veamos la siguiente tabla:

Matriz	**Transformación que realiza**
WorldTransform	Espacio del Objeto al Espacio del Mundo
WorldToCamera *	Espacio del Mundo al Espacio de la Cámara
WorkingTransform	Espacio del Objeto al Espacio de la Cámara
CameraToScreen *	Espacio de la Cámara al Espacio de la Pantalla
WorldToScreenTransform *	Espacio del Mundo al Espacio de la Pantalla
FinalTransform	Espacio del Objeto al Espacio de la Pantalla

Donde los * significan que esas matrices son independientes de los objetos, sólo dependen de la escena y de la definición de nuestra cámara y de la pantalla.

Resumimos todo este conjunto de espacios y transformaciones entre ellos con el siguiente mapa (ver figura 3.5):

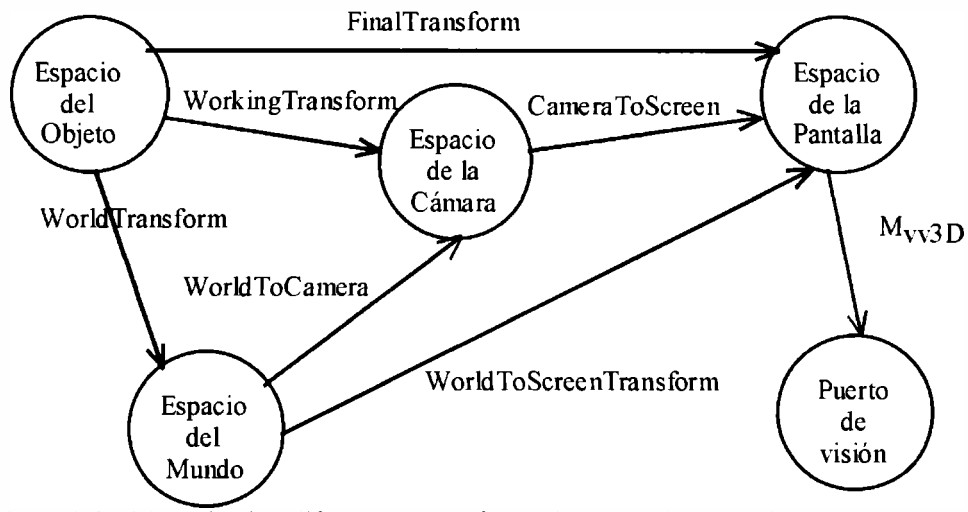


Figura 3.5: Mapa de las diferentes transformaciones y los espacios asociados

Qué ocurre si, por alguna razón, calculamos un vector en un espacio dado, por ejemplo el de la Cámara, y deseamos hallar su equivalente en el espacio del Mundo? Conocemos la transformación que nos lleva del segundo al primero, WorldToCamera, pero no la que realiza el camino inverso. La

matriz que realizará dicha transformación será la matriz inversa de la dada, y la podremos notar como:

$$CameraToWorld = WorldToCamera^{-1}$$

Con $WorldToCamera^{-1}$ nos referimos a la matriz inversa de WorldToCamera, esta propiedad es trivial de demostrar usando álgebra matricial.

En general, dada una matriz cualquiera, no podemos afirmar que posea inversa. Pero nuestras matrices son siempre matrices formadas por operaciones elementales, traslaciones, rotaciones y cambios de tamaño, por lo que podemos afirmar que siempre tendrán inversa. La inversa de una matriz compuesta será, simplemente, la composición en orden inverso de las inversas de las componentes. Un claro ejemplo de una matriz no invertible es Mper, puesto que esta transformación no conserva a nuestros vectores en el espacio de los vectores homogéneos (cuarta componente igual a 1). Por lo tanto, cualquier matriz que contenga a ésta, no será invertible, por ejemplo, FinalTransform.

3.6. Dibujando nuestros objetos

Finalmente, para representar nuestros objetos, debemos armar primeramente todas estas matrices. Es importante no olvidar que la mayoría de estas matrices son propias de cada objeto, puesto que WorldTransform lo es. Pero otras, como WorldToCamera o CameraToScreen no dependen de los objetos en la escena, por lo que pueden ser calculados sólo una vez durante todo el proceso.

Una vez hecho esto, transformamos nuestros objetos al mundo de la cámara, usando WorkingTransform, verificando si son visibles y qué parte de ellos lo son (clipping, ver pág. 32). Si son visibles, entonces los proyectamos a la pantalla con CameraToScreen, los terminamos de proyectar, dividiendo respectivamente cada vértice que lo forma por su cuarta componente y terminamos de transformarlo a la pantalla física con Mvv3Dv y lo dibujamos en pantalla.

El pseudocódigo que hace todo esto es:

Armar todas las matrices independientes de los objetos
Para cada objeto en escena
  Armar las restantes matrices
  Para cada triángulo que lo forma
    Transformar todos sus puntos mediante WorkingTransform
    Verificar que partes se ven y recortarlo según sea conveniente
    Si (se ve)
      Hacer la transformación final con CameraToScreen
      Dividir por la cuarta componente
      Transformar a la pantalla con M_{vv3Dv}
      Dibujarlo

4. Proceso de renderización de objetos poligonales

[Watt92] Durante algunos años el trabajo de gráficos tridimensionales a sido un sistema básico para renderear[2] objetos representados por un conjunto de polígonos. La aproximación tradicional para renderear objetos tridimensionales es construir un render básico, e ir agregando algunas mejoras. Muchos renders trabajan con objetos que son representados por polígonos (en nuestro caso particular son triángulos). Esta aproximación resuelve los siguientes problemas:

- Modelar objetos usando polígonos es correcto. Los efectos visuales de formar objetos mediante la utilización de polígonos (linealidades) pueden resultar invisibles al ser rendereados debido a la técnica de shading (ver pág. 38).

- La información geométrica es solamente almacenada en los vértices, y la información requerida para el modelo de reflexión que evalúa pixel por pixel es interpolada desde la información de los vértices. Esto es hecho para un formado de la imagen más rápido.

- Un render basado en polígonos también puede ser usado para renderear objetos que son hechos de un conjunto de piezas bicúbicas. Una forma fácil de conducirnos con estos objetos es subdividir las piezas y convertirlas en polígonos planares, encajando así el render de polígonos.

Por otro lado el render de objetos poligonales esta sujeto a ciertas desventajas. Por ejemplo, es difícil mapear texturas desde un dominio bidimensional sobre la superficie de un objeto. Los algoritmos de sombras que están basados en objetos poligonales tienen alta complejidad de codificación y producen sombras duras en los bordes. Un objeto complejo que es representado con un gran nivel de detalle puede generar un gran número de triángulos.

Los principales pasos al renderear objetos poligonales son:

1. Los polígonos que representan un objeto son extraídos de la base de datos y transformados dentro del sistema de coordenadas del mundo usando transformaciones lineales tales como traslaciones y modificaciones de las escalas. Ver la representación de objetos en el Espacio del Mundo (pág. 18).

[2] La palabra renderear es un neologismo, dado que no existe en castellano. Significa sintetizar la imagen a partir de los datos de la escena.

2. Una escena construida de esta manera es transformada en un sistema de coordenadas basado en el punto de vista o la dirección de visión. Ver la representación de objetos en el Espacio de la Cámara (pág. 19).

3. Los polígonos son entonces sujetos a un test de visibilidad llamado "eliminación de caras traseras" que elimina aquellos polígonos que forman caras que están fuera del ángulo de visión del observador. Típicamente la mitad de los polígonos son eliminados con este test, y no tienen que ser tratados posteriormente con un algoritmo más costoso que remueve los objetos escondidos. (Ver Eliminación de Caras Traseras, pág. 30).

4. Los polígonos no eliminados por el test anterior son cortados contra el volumen de visión tridimensional por medio de un proceso llamado "clipping". (Ver Recorte de Triángulos, pág. 32)

5. Los polígonos cortados son entonces proyectados sobre un plano de visión o plano de la pantalla. Ver la representación de objetos en el Espacio de la Pantalla (pág. 21).

6. Luego los polígonos proyectados son armados por un algoritmo de shading incremental. El algoritmo corre tres procesos en paralelo. Primero los polígonos son rasterizados, es decir, son determinados los pixeles que contienen los lados de los polígonos. En segundo lugar es evaluada la profundidad para cada pixel y se implementa el cálculo de superficies escondidas. Tercero el polígono es armado. Ambos, el segundo y el tercer proceso usan información geométrica desde el espacio del mundo tridimensional o del espacio de visión, pero sobre todo el proceso es controlado por el espacio de la pantalla.

Los pasos desde el 1 hasta el 5 son estándar para muchos renders e involucran operaciones geométricas e información de los vértices. El sexto paso involucra combinación de rasterización, determinación de superficies visibles y armado de polígonos. En esta fase hay muchas metodologías posibles. Las mayores opciones están dadas por el algoritmo de determinación de superficies visibles y los métodos de antialiasing (ver pág. 100). La aproximación descripta en el paso 6 depende en la práctica del uso de un algoritmo Z_Buffer (ver pág. 43) para resolver el problema de eliminar superficies escondidas. Este tiene baja complejidad pero altos requerimientos de memoria. Se ha vuelto el algoritmo más popular para eliminar superficies. En este trabajo proponemos mejoras con el manejo de otra estructura: el A_Buffer (ver pág. 46).

Aunque el enunciado de la idea de determinar superficies y líneas visibles es sencillo, su implementación requiere gran poder de procesamiento y por ende comprende grandes cantidades de tiempo de máquinas convencionales. Estos requisitos han alentado el desarrollo de varios

algoritmos cuidadosamente estructurados para la determinación de superficies visibles. Nosotros presentaremos los que usaremos en nuestro código a continuación.

4.1. Eliminación de Caras Traseras

Si se aproxima un objeto sólido por un poliedro, podemos asegurar que sus caras encierran completamente a su volumen. Eso significa que no podemos ver su interior desde ningún punto externo, puesto que no tienen agujeros que nos lo permita. Asumamos que todos los triángulos están definidos en forma tal que el normal a su superficie apunta hacia afuera del objeto aproximado (normal externo a la superficie del objeto). Entonces, podemos asegurar que ningún triángulo trasero (aquellos cuyo normal externo apunta en dirección contraria al Centro de Proyección) será visible por un observador externo, debido a que siempre habrá un triángulo más cercano. Estos triángulos invisibles por un observador pueden descartarse trivialmente por una técnica conocida como Eliminación de Caras Trasera o Back-Face Culling o Back-Face Removal [Foley90]. Ver la figura 4.1 para ver un ejemplo de caras traseras y caras delanteras de un polígono.

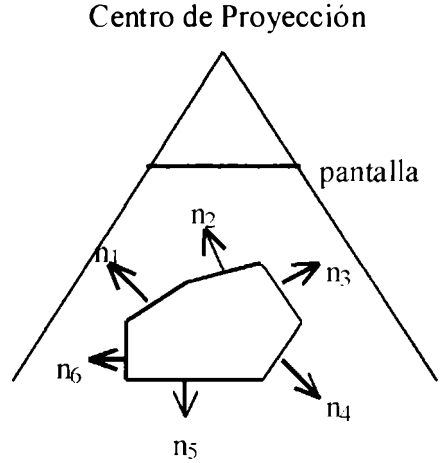
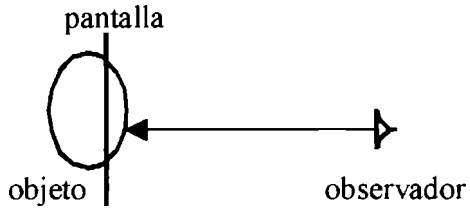


Figura 4.1: Normales externos "delanteros" (n_1, n_2, n_3) y normales externos "traseros" de un polígono (n_4, n_5, n_6), desde el centro de proyección

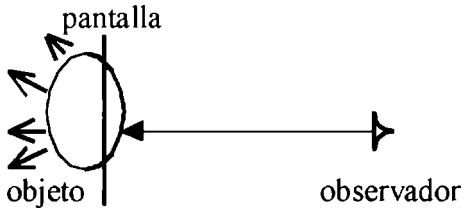
Qué ventajas presenta esta eliminación? Evita analizar y procesar triángulos innecesarios. Groseramente, se puede decir que, en promedio, de esta forma se elimina aproximadamente la mitad de los triángulos en una escena, lo cual es muy importante en términos de performance. Por supuesto que existen ejemplos en donde esto no es cierto, pero en general toda escena tiene objetos con tantos triángulos traseros como triángulos delanteros.

Estamos suponiendo que nuestros objetos no han sido recortados por el proceso de clipping contra el plano de la pantalla, puesto que esto podría hacer que nuestros objetos tuviesen huecos, como se muestra en la figura 4.2. En general, ambos procesos se implementan independientemente, por lo que, si se producen huecos por el clipping, de todas formas los polígonos traseros

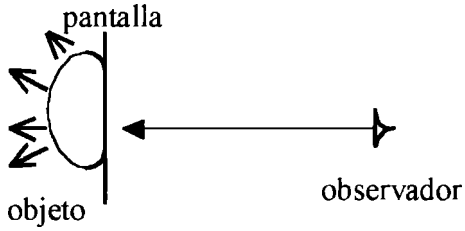
serían eliminados. Es por ello que, si se buscan fines estéticos, el animador, usuario final de nuestros algoritmos de rendering, será el que deba cuidar estos detalles.



a)presentación de situación



b)situación posterior a la eliminación de caras traseras



c)situación posterior al recorte respecto al volumen de visión

Figura 4.2: hueco generado por el recorte contra el plano de la pantalla que hace visible las caras traseras que ya han sido eliminadas por el algoritmo de recorte de caras traseras

En coordenadas de la cámara, un triángulo trasero puede ser identificado porque el producto escalar entre su normal y el vector desde el Centro de Proyección hacia cualquier punto del mismo es mayor que cero, es decir, no negativo. Si el producto escalar es cero, significa que ambos vectores son perpendiculares. El polígono está siendo visto desde el costado, por lo que sólo se ve como una línea. Si asumimos que se realizó una proyección en perspectiva a un plano (x, y), por ejemplo, el plano de la pantalla, entonces la dirección de proyección será (0,0,-1). En este caso, la verificación del producto escalar puede ser reemplazada por la verificación de que la tercera componente del vector normal sea negativa, es decir, que z sea negativo. En este caso, el polígono es trasero y debe ser rechazado.

Cuándo es el momento ideal de remover las caras traseras? Precisamente en la verificación de si se debe dibujar o no antes de realizar todos lo cálculos de iluminación y sombras. Para ello, los vértices de cada triángulo ya han sido transformados a nuestra pantalla (mediante Mvv3Dv, porque la proyección en perspectiva puede alterar la visibilidad de un polígono), bastando ahora con tomar los tres vértices que lo forman (en el

plano de la pantalla) y calcular dos vectores, uno desde el primero hasta el segundo vértice, y otro del primero al tercero. La componente z de su producto vectorial nos dirá si debemos dibujarlo o no.

De esta explicación se desprende que, al definir nuestros triángulos, el orden en que escribamos los vértices definirá la orientación de un triángulo. Por ello, si se observa nuestra definición del cubo (ver Representación de objetos en el Espacio del Objeto, pág. 16) veremos que todos los triángulos están definidos de forma tal que su normal, definido por el mencionado producto vectorial, siempre apunte hacia afuera del mismo.

4.2. Recorte de Triángulos (Poligon Clipping)

El problema del Recorte de Triángulos surge como una necesidad cuando debemos determinar qué partes de un objeto son visibles desde un observador, representado por nuestra cámara, formada por el centro de proyección y la pantalla. Por lo tanto, en general deberemos determinar qué triángulos y que partes de los mismos son visibles. El problema, para una cámara en general, queda limitado a hallar el recorte de los polígonos según el volumen de visibilidad, mostrado en la figura 4.3 por la parte sombreada.

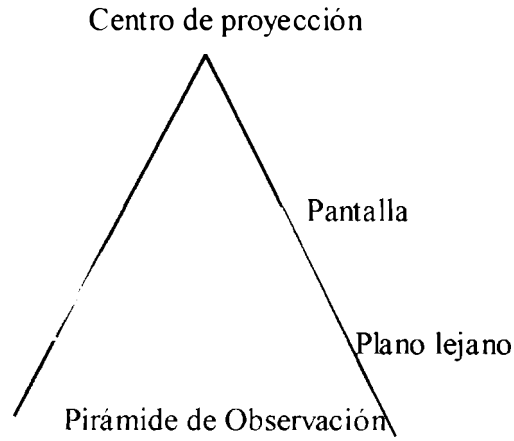


Figura 4.3: El volumen de visibilida
(encerrado entre líneas gruesas)

Los seis planos que forman el volumen de visibilidad están dados por la pantalla, los cuatro planos de la pirámide de observación, y un plano llamado "plano lejano", usado para recortar aquellos objetos que se encuentran lejos del observador. Este último plano es muy usado para reducir aún más el número de polígonos examinados, pero, en general, no es indispensable, por lo que el volumen de visibilidad se extendería hasta el infinito por este lado.

Para determinar qué parte de un polígono es visible, habría que calcular qué partes del mismo quedan dentro del volumen, y recortarlo para que sólo estas partes queden. Para esto, podemos aplicar algún algoritmo que calcule el recorte contra un único plano, y repetir la operación para cada uno de los planos que forman dicho volumen.

Existe una solución mucho más económica, pero menos elegante: realizar el recorte sólo con el plano de la pantalla, dejando para más adelante el recorte contra los otros planos que forman el volumen, mediante el facilista método de no dibujar aquello que cae fuera de los límites de la pantalla física.

Para un triángulo, en general sólo existirán los cuatro casos presentados en la figura 4.4. Como se puede ver fácilmente en la figura, para el caso particular de recortar triángulos, sólo cuatro casos pueden presentarse: triángulo completamente adentro, cortado en dos triángulos menores (un tercero queda descartado por estar afuera), triángulo cortado en sólo uno, y, finalmente, polígono descartado por quedar completamente fuera de espacio visible.

De esta manera, a la entrada del algoritmo tenemos un triángulo a ser examinado, y de salida sólo podemos tener tres casos posibles: ninguno, uno o dos triángulos.

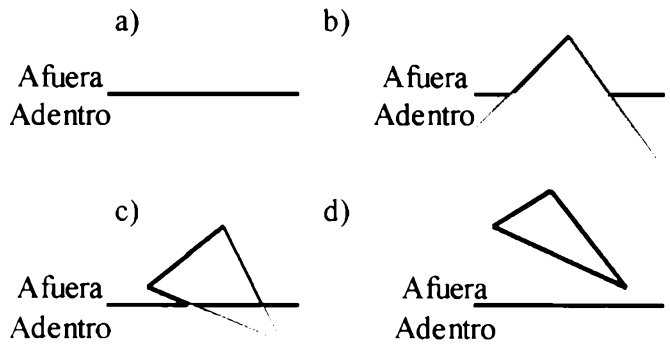


Figura 4.4:a)Polígono completamente dentro del espacio visible
b)Polígono cortado en dos por el plano de separación
c)Polígono cortado en uno por el plano de separación
d)Polígono eliminado por estar completamente fuera

Uno de los algoritmos más utilizados es el de Sutherland-Hodgman [Watt92], que verifica cada una de las aristas del polígono contra el plano, recortándolas según sea conveniente. Es importante destacar que el algoritmo presentado es suficientemente general para recortar cualquier polígono que reciba. Nosotros analizaremos aquí sólo su implementación para triángulos, dado que es la representación elegida en nuestro algoritmo. Sin embargo, su extensión es casi trivial.

El algoritmo se basa en calcular los siguientes cuatro casos posibles, presentados en la Figura 4.5:

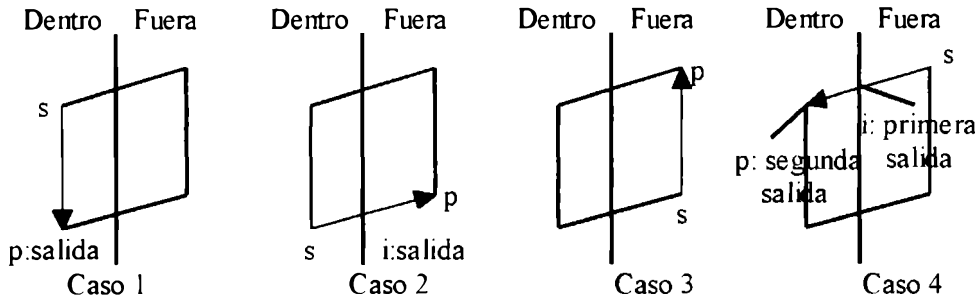


Figura 4.5: Los posibles casos analizados en el algoritmo de Sutherland-Hodgman

El algoritmo recibe un arreglo de entrada con los vértices originales del polígono, en éste ejemplo, tenemos tres elementos. Comenzamos con el último vértice del polígono, luego procedemos a examinar el primero e iteramos hasta llegar nuevamente al último.

Consideremos la arista de los vértices **s** al **p**, supongamos que **s** fue analizado en la iteración anterior. En el caso 1, la arista está completamente dentro del hemisferio que nos interesa, entonces el vértice **p** es agregado a la lista de salida. En el siguiente caso, el punto de intersección **i** se agrega a la salida, puesto que es en este punto que intercepta al contorno. En el caso 3, ambos vértices están fuera, por lo que no hay salida. En el último caso, los puntos de intersección **i** y **p** son agregados a la lista.

El código que implementa el algoritmo de clipping de Sutherland-Hodgman para triángulos utiliza tres procedimientos internos: Inside, que es una función que devuelve verdadero si el vértice que recibe como parámetro está dentro del volumen que nos interesa, en este caso solo verifica si está del lado del plano de la pantalla opuesto al Centro de Proyección. Intersect, que calcula todos los valores **i** cuando estamos en el caso 2 o 4; y Output, que carga los puntos de salida en un arreglo de cuatro elementos, de donde, al finalizar el clipping obtendremos los valores para formar los triángulos resultantes. Como sólo podemos obtener dos triángulos como máximo, el procedimiento devuelve siempre dos triángulos. Si alguno de los dos, o ambos, no sirve, el programa le asigna a su componente z un valor por el cual seguramente queda descartado, (1 en este caso ya que todos los polígonos visibles tienen su tercera componente menor a -d).

De esta forma, la verificación de que si un triángulo es visible o no se limita, simplemente, a verificar si alguno de sus vértices tiene z > -d. En ese caso, el triángulo deberá ser descartado.

4.3. Procesamiento a nivel del pixel

Rasterización, remover superficies escondidas y shading son tres procesos implementados en los loops más internos del proceso de render. Estos procesos operan sobre un polígono por vez y son organizados alrededor

de spans y pixeles. Un span es la intersección de una línea de scaneado con dos lados de un polígono que es convertida en una corrida de pixeles consecutivos.

Un problema asociado es como encontrar el par de lados apropiados desde la estructura de datos para una línea de scan adecuada. Los lados son interpolados desde los dos vértices que lo definen.

Podemos ver los procesos de rasterización, remover superficies escondidas y shading como tres procesos de interpolación en dos dimensiones. La rasterización consiste en interpolar entre vértices para encontrar las coordenadas X que definen el span, y entonces encontrar los pixeles entre estos vértices. Remover superficies escondidas consiste en la interpolación de los valores Z del espacio de la pantalla para obtener la profundidad de cada pixel (o subpixeles en el caso del A_Buffer) desde los valores de profundidad del vértice. Shading implica realizar una interpolación de la intensidad de los vértices para encontrar una intensidad para cada pixel. Para resolver estos últimos puntos se ha utilizado el Z_Buffer. Para implementar la interpolación usamos la misma ecuación y se la aplicamos a las coordenadas de los vértices (rasterización), los valores de profundidad de los vértices (remover superficies escondidas) o a la intensidad de los vértices (shading). Las interpolaciones son encajadas en loops anidados. Finalmente, la estructura completa del proceso de render convencional es [Watt92]:

```
para cada polígono
  implementar transformaciones geométricas dentro del espacio de la
pantalla
  para cada línea de scan dentro de un polígono
    encontrar los span por interpolación y rasterizar el span
    para cada pixel dentro del span
    implementar los procesos de remover superficies escondidas y shade
```

En nuestro algoritmo, en la implementación del proceso de render, invertimos los bucles "para cada polígono" y "para cada línea de scan" como se muestra en la siguiente estructura:

```
para cada polígono
    implementar transformaciones geométricas dentro del espacio de la
pantalla
para cada línea de scan
    para cada polígono transformado en esa línea
        encontrar los span por interpolación y rasterizar el span
        para cada pixel dentro del span
            implementar los procesos de remover superficies escondidas y
shade
```

Especialmente hicimos esto para ahorrar memoria. Es decir, en vez de tener un buffer de n*m pixeles solo necesitamos tener uno de 1*m pixeles debido a que trabajamos con una sola línea de scan por vez, pero con todos

los polígonos que aparecen en esa línea. Aumenta la complejidad de implementación del algoritmo, pero mejora mucho el uso de memoria.

4.3.1. Rasterización

Como ya lo mencionamos anteriormente es el proceso de encontrar cuales son los pixeles de un polígono que se proyectan en la pantalla.

Para cada scan dentro del polígono nosotros calculamos mediante una interpolación lineal, los valores de X de los pixeles del lado, Xstart y Xend. Esto da como resultado un span de pixeles entre dos lados que tienen que ser rendereados. Hay, sin embargo un número de problemas. Por ejemplo ¿dónde comienza o termina exactamente un lado? ¿Dónde debería ser muestreado un punto exactamente dentro del pixel? ¿Deberían las coordenadas de la pantalla ser redondeadas a enteros o guardadas en números reales? Respuestas incorrectas a estas preguntas puede llevarnos a obtener agujeros dentro de los polígonos, polígonos sobrelapeados (lo cual es desastroso si estamos usando transparencias), discontinuidades sobre las superficies sobre las que tienen que mapearse texturas e inexactitudes en el antialiasing. Pequeñas grietas aparecen frecuentemente en el software de render. Muchas de ellas se deben a una rasterización incorrecta.

Consideremos el primer problema de las coordenadas de la pantalla. Claramente si ellas son redondeadas a valores enteros los polígonos tendrán sus formas cambiadas levemente. Cada vértice será llevado a su valor más cercano en el punto de la grilla como se muestra en la figura 4.6:

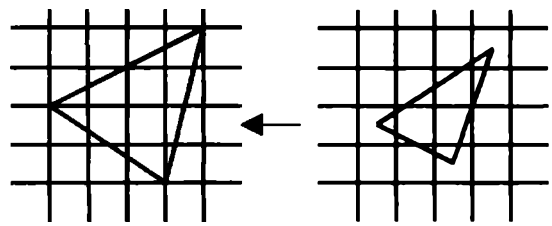


Figura 4.6: Coordenadas de la pantalla redondeadas
a resultados enetros

También un leve cambio en las formas puede producir vacilaciones si la proyección es cambiante en una secuencia de animación.

Otro efecto de las coordenadas enteras ocurre cuando hay un paso en un lado vertical cercano. Si se están usando enteros en la rasterización con un shading interpolativo, este introducirá una discontinuidad en los valores del shading del pixel entre dos líneas de scan. Para shading interpolativo esto no es un problema pero puede ser desastroso en el mapeo de texturas.

En nuestro algoritmo no redondeamos a valores enteros las coordenadas de la pantalla.

El siguiente punto a considerar es la posición del punto muestreado dentro del pixel. Una imagen rendereada es calculada como un conjunto de puntos muestreados distribuido en forma de arreglos bidimensionales. Muchos autores consideran el punto de muestreo en el centro del pixel. Esto conduce a calcular la intersección con el scan en y + 0.5 en vez de en el entero y. No hay problema en cuanto al lugar donde el punto muestreado este en el pixel en tanto seamos consistentes. Usar el punto de muestreo en una de las esquinas del pixel significará que la imagen rendereada estará desplazada medio pixel de su posición original.

En nuestro algoritmo consideramos el punto de muestreo en el centro del pixel.

Para un punto muestreado en un pixel consideramos la rasterización en dos partes. Primero, dado los valores izquierdo y derecho para un span horizontal en una línea de scan consideramos como redondear sus valores a las coordenadas enteras de la pantalla. Segundo consideramos como detectar los valores de los lados que definirán el span.

4.3.1.1. Rasterización de spans y líneas de scan

Un span es la intersección entre una línea de scan y el triángulo que se está rasterizando. En nuestro algoritmo un span es simplemente la línea entre los puntos X_{start} y X_{end}. Para spans horizontales, si nosotros estamos usando números reales, entonces dados X_{start} y X_{end}, la estrategia es:

Redondear X_{start} hacia arriba.

Redondear X_{end} hacia abajo.

Si la parte fraccionaria de X_{end} es 0 entonces substraerle 1.

Alternativamente para aumentar la eficiencia podemos usar aritmética entera y, para aumentar la precisión, deberíamos escalar todos los enteros hacia arriba por el mismo factor (un múltiplo de 16 será suficiente). Usando enteros:

Redondeamos X_{start} hacia arriba hasta el próximo múltiplo de 16.

Redondeamos X_{end} hacia abajo hasta el próximo múltiplo de 16.

Si X_{end} ya es múltiplo de 16 le restamos 16.

Entonces las coordenadas enteras de la pantalla son generadas para todo múltiplo de 16 entre X_{start} y X_{end} inclusive.

4.3.1.2. Rasterización e interpolación

Como ya hemos discutido muchas de las grietas resultantes en la rasterización ocurren al redondear los valores de X en el span o al redondear los valores de Y sobre las líneas de scan de los triángulos. El éxito de la rasterización depende de la implementación de estos puntos apropiadamente. El resto de los problemas asociados a la rasterización son bastante simples.

Para cada par de lados debemos encontrar los valores de X_{start} y X_{end}. Esto lo hacemos por interpolación lineal, incrementalmente scan por scan, dividiendo la diferencia entre la coordenada X de los puntos finales por su peso en la línea de scan. Este incremento es sumado a los valores previos de X_{start} y X_{end} para obtener el valor corriente. La interpolación de los parámetros de shading se realizan a la vez y del mismo modo.

También es interpolada la profundidad de la pantalla, y ésta es completada sobre una base pixel por pixel del algoritmo de superficies escondidas.

En lo que se refiere a la interpolación llamaremos P al parámetro de shading y el valor Δp a lo largo de la línea de scan está dado por:

$$\Delta p = (P_{end} - P_{start}) / (X_{end} - X_{start})$$

Podríamos no usar el valor P_{start} como el valor de comienzo para la interpolación a lo largo del span. Como siempre redondeamos X_{start} hacia arriba, el primer pixel estará en algún lugar a la derecha del comienzo del span. El valor de comienzo, es decir P_o, es calculado desde el valor del lado interpolado como:

$$P_o = P_{start} + K * \Delta p$$

donde $K = \text{redondeo}(X_{start}) - X_{start}$.

4.3.2. Algoritmos para remover superficies escondidas

Dado un conjunto de objetos tridimensionales y una especificación de vista, queremos determinar cuales son las superficies visibles de los objetos desde el centro de proyección, de manera que podamos presentar únicamente las superficies visibles. Este proceso se conoce como determinación de superficies visibles o eliminación de superficies escondidas [Foley96].

El hecho de que su implantación requiere gran poder de procesamiento ha alentado el desarrollo de varios algoritmos cuidadosamente estructurados para la determinación de superficies visibles.

Hay dos métodos fundamentales para resolver el problema.

El primer método consiste en determinar cuál de los n objetos es visible en cada pixel de la imagen. El pseudocódigo para este método tiene el siguiente aspecto:

```
Para cada pixel en la imagen
    determinar el objeto más cercano al observador
    dibujar el pixel con el color apropiado
```

La forma más burda y directa de hacer esto con un pixel requiere la revisión de los n objetos para determinar cual es el más cercano al observador. En el caso de p pixeles este esfuerzo es proporcional a n*p, donde p es superior al millón en una pantalla de alta resolución.

El segundo método es comparar los objetos directamente entre sí, eliminando objetos enteros o porciones de ellos que no sean visibles. En pseudocódigo, esto es:

```
para cada objeto en el mundo
    determinar aquellas partes del objeto cuya vista no está obstruida
por otras partes del mismo objeto o por otro objeto
    dibujar estas partes con el color apropiado
```

Esto se puede hacer en forma sencilla comparando cada uno de los n objetos consigo mismo y con los otros y descartando las porciones invisibles. En este caso el esfuerzo computacional es proporcional a n^2 . Aunque este segundo método podría parecer superior si n < p, sus pasos individuales son generalmente más complejos y consumen más tiempo, de manera que en la mayoría de los casos es más lento y difícil de implantar. Nos referimos a estos métodos como algoritmos de precisión de la imagen y de precisión del objeto, respectivamente.

Presentaremos a continuación una breve descripción de los principales métodos para determinar superficies visibles.

4.3.2.1. Algoritmo de Z_Buffer

El algoritmo de precisión de imagen, Z_Buffer [Foley96], desarrollado por Catmull en 1974, es uno de los algoritmos de superficies visibles más fáciles de implantar en software y en hardware. Requiere que se tenga una memoria gráfica donde se almacenan los valores de los colores y una memoria de profundidad z, con el mismo número de entradas, donde se almacena un valor de z para cada pixel.

En esta estructura el valor de z que se almacena es el más cercano al punto de observación.

Aclaremos que Catmull sugirió, en esos días este algoritmo, ya que hablar de guardar una matriz de números reales del tamaño de la pantalla era imposible por la limitación de memoria que había en ese momento. Nunca llego a desarrollarlo, al menos, no en esos trabajos pioneros.

Veremos este algoritmo en detalle en el próximo capítulo.

4.3.2.2. Traza de rayos (ray tracing)

Este método determina la visibilidad de las superficies trazando rayos luminosos imaginarios del ojo del observador a los objetos en la escena [Foley96]. Este es un algoritmo de precisión de imagen. Se seleccionan un centro de proyección (el ojo del observador) y una ventana en un plano de vista arbitrario. La ventana se puede considerar como una malla regular cuyos elementos corresponden a los pixeles con la resolución deseada. Entonces, para cada pixel en la ventana se dispara un rayo ocular que va del centro de proyección a la escena, a través del centro del pixel. El color del pixel se asigna igual al del objeto en el punto de intersección más cercano. El pseudocódigo para este sencillo trazador de rayos se presenta a continuación:

```
Para cada línea de rastreo en la imagen
  Para cada pixel en la línea de rastreo
    determinar el rayo del centro de proyección a través del pixel
    Para cada objeto en la escena
      Si se interseca el objeto y es el más cercano considerado hasta
ahora
        registrar intersección y nombre del objeto
    asignar el color del pixel igual al de la intersección del objeto más
cercano
```

4.3.2.3. Algoritmos de prioridad de listas

Estos algoritmos [Foley96] determinan un ordenamiento de visibilidad para los objetos, asegurando que se produzca una imagen correcta si los objetos se generan en dicho orden, por ejemplo si ningún objeto se sobrepone a otro en z, sólo hay que ordenar los objetos en forma ascendente de acuerdo con z y luego generarlos en ese orden. Los objetos más lejanos estarán oscurecidos por los más cercanos ya que los pixeles de los polígonos más próximos sobreescriben los de los más distantes. Si los objetos se sobreponen en z, aún es posible determinar el orden correcto, como en el figura 4.7 a). Si los objetos se sobreponen cíclicamente, como en la figura 4.7 b) y c) o son interpenetrantes, no hay un orden correcto. En estos casos será necesario dividir uno o más objetos para que sea posible el ordenamiento lineal.

Los algoritmos de prioridad de listas son híbridos que combinan operaciones, tanto de precisión de la imagen como de precisión del objeto. Las comparaciones de profundidad y la división de objetos se llevan a cabo con precisión de objetos. Sólo la discretización, que se basa en la capacidad del dispositivo gráfico para sobreescribir los pixeles de los objetos previamente dibujados se realiza con precisión de imagen. Sin embargo, como la lista de objetos ordenados se crea con precisión de objeto, se puede volver a presentar correctamente a cualquier resolución

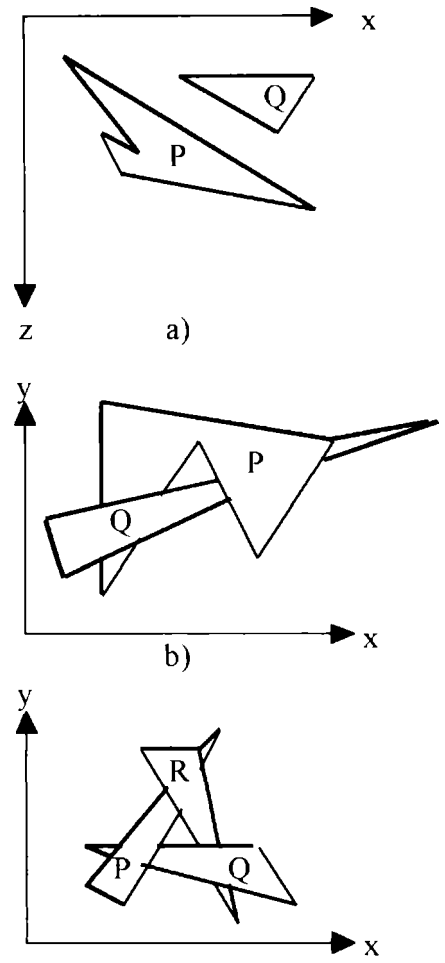


Figura 4.7: Algunos casos en los que se sobreponen las extensiones z de los polígonos

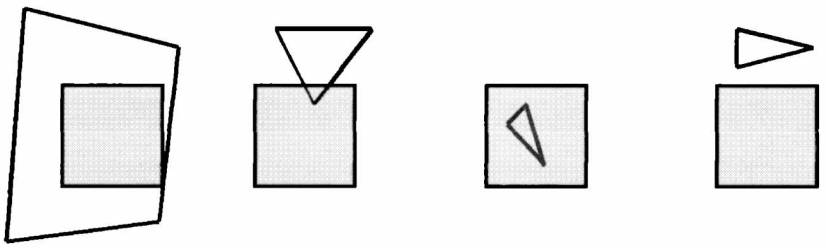
Algunos de estos algoritmos son el del Pintor (de ordenamiento por profundidad) y árboles binarios para partición de espacio [Foley96].

4.3.2.4. Algoritmo de subdivisión de área

Todos los algoritmos de subdivisión de áreas siguen la estrategia "divide y vencerás" de partición espacial en el plano de proyección [Foley96]. Se examina un área de la imagen proyectada; si es fácil decidir cuáles son los polígonos visibles en el área, éstos se dibujan. En caso contrario, el área se subdivide en áreas más pequeñas a las cuales se aplica recursivamente la lógica de decisión. A medida que las áreas se van haciendo más pequeñas, son menos los polígonos que se sobreponen en cada área y al final de cuentas es posible tomar una decisión. Esta estrategia aprovecha la coherencia de áreas, ya que las áreas suficientemente pequeñas de una imagen estarán contenidas a lo sumo en un sólo polígono visible.

Por ejemplo el algoritmo de Warnock subdivide cada área en cuatro cuadrados iguales. En cada etapa del proceso de subdivisión recursiva, la

proyección de cada polígono tiene una de cuatro relaciones con el área de interés (ver figura 4.8)



a)Circundante b)Intersecante c)Contenido d)Disjunto

Figura 4.8: Cuatro relaciones de proyecciones de polígonos a un elemento de área

5. Algoritmos de memoria de profundidad

El algoritmo Z_Buffer es un algoritmo de memoria de profundidad estándar en computación gráfica que almacena para cada pixel el objeto más cercano al observador en esa posición, junto con su profundidad en z y el valor de su color.

El A_Buffer es un algoritmo similar al Z_Buffer que almacena para cada pixel todos los objetos que se encuentran en una posición (x,y) determinada, ordenados por profundidad. Se ordenan desde el más cercano al más lejano al punto de observación y permite superar algunas limitaciones que se plantean en el algoritmo anterior. Esta es la estructura que usaremos en nuestro trabajo.

A continuación presentamos una descripción detallada de ambos algoritmos.

5.1. Z_Buffer

El algoritmo Z_Buffer es de baja complejidad, y aunque es de alto costo en memoria, las terminales gráficas actuales lo superan teniendo una memoria dedicada al Z_Buffer [Foley90].

Este algoritmo es visto mejor operando en un espacio de pantalla tridimensional. Cada pixel es asociado con una coordenada de pantalla bidimensional (X_s, Y_s) junto con un valor de profundidad Z interpolado desde la profundidad del vértice en el espacio de la cámara a la vez que los valores de shading son interpolados y la rasterización es implementada. Dentro de la estructura del Z_Buffer almacenamos también, los valores de color para cada pixel. El Z_Buffer es inicializado con el valor del plano de recorte más lejano y el color es inicializado con el color del background. Para cada pixel comparamos su profundidad con la profundidad ya almacenada en el Z_Buffer (X_s, Y_s). Si este valor es menor que el valor almacenado entonces el pixel es más cercano al observador con respecto al pixel encontrado previamente, en cuyo caso el valor del color corriente es escrito en la memoria de la pantalla y la profundidad corriente ubicada en el Z_Buffer. En esta estructura guardamos siempre el pixel cuyo valor de Z es más cercano a la pantalla. El pseudocódigo asociado a este algoritmo es:

```
procedure zBuffer
var
   pz: integer;
begin
   for y := 0 to YMAX do
    for x := 0 to XMAX do
      begin
         WritePixel (x,y,BACKGROUND_VALOR);
         WriteZ (x,y,0);
      end
   for cada polígono do
    for cada pixel en la proyección del polígono do
      begin
         pz := el valor z del polígono en las coordenadas (x,y);
         if pz >= ReadZ (x,y) then
          begin
            WriteZ (x, y, pz);
            WritePixel (x, y, color del polígono en las coordenadas
del pixel (x,y));
          end
      end
end;
```

El algoritmo es equivalente en efecto, pero no en ejecución, a una búsqueda por cada pixel, a través de todos los polígonos que interceptan el pixel para encontrar el más cercano al punto de vista.

El algoritmo de Z_Buffer no impone restricciones acerca de la organización de la base de datos y, en su forma más simple, puede ser manejado en una base triángulo por triángulo, con los triángulos presentados en el proceso de rendering en algún orden. Los triángulos aparecen en la pantalla en el orden en que ellos se extraen de la base de datos. Claramente algunos triángulos aparecerán y desaparecerán lo que aumenta la ineficiencia del algoritmo, debido a que cálculos de valores de color son implementados sobre pixeles que son posteriormente sobrescritos. Como el algoritmo trabaja con un polígono por vez, no impone restricciones sobre la complejidad de la escena y ésta ha sido una razón para su popularidad en una década, que tiene un crecimiento seguro en la resolución o la complejidad de los objetos.

Para calcular Z en cada punto sobre la línea de scan podemos utilizar el hecho de que cada triángulo es plano. Normalmente para calcular Z, podemos resolver la ecuación del plano: Ax + By + Cz + D = 0, para la variable z entonces:

$$z = (-D - Ax - By) / C$$

Ahora, si z1 es el valor de z en el punto (x,y), entonces el valor de z en el punto $(x+\Delta x,y)$ es

$$z1 - (A/C)*(\Delta x)$$

Solamente una substracción es necesaria para calcular z(x + 1,y) dado z(x,y), dado que A/C es constante y $\Delta x = 1$. Un cálculo incremental similar puede ser implementado para determinar el primer valor de z sobre la próxima línea de scan, restando B/C para cada Δy.

No es necesario calcular el color de un pixel si no se satisface la condición que determina su visibilidad. Por lo tanto, si el cálculo del sombreado consume mucho tiempo, se puede obtener mayor eficiencia si se lleva a cabo un ordenamiento general de los objetos por profundidades, de anterior a posterior para representar primero los objetos más cercanos.

La precisión de la profundidad es muy importante y escenas muy complejas u objetos pueden requerir 32 bits de precisión. Por ejemplo, objetos definidos con posición milimétrica y ubicados a un kilómetro de distancia Las proyecciones en perspectiva aumentan el problema debido a que la compresión de los valores z distantes que resultan de la división de perspectiva tiene un serio efecto en el ordenamiento por profundidad y en las intersecciones de objetos distantes. Dos puntos que se transformarían a valores z enteros diferentes si estuvieran cerca del plano de vista podrían transformarse al mismo valor z si estuvieran lejanos.

El algoritmo opera con unidades que son pixeles. No necesariamente requiere polígonos y puede ser usado con cualquier representación de objetos. De hecho, uno de sus aspectos más atractivos es que se puede usar para generar la imagen de cualquier objeto si es posible determinar un matiz y un valor z para cada punto de su proyección. No hay que escribir algoritmos de intersección explícito.

El tiempo necesario para los cálculos de superficie visible tiende a ser independiente del número de polígonos en los objetos, ya que, en promedio, el número de pixeles cubiertos por cada polígono disminuye al aumentar el número de polígonos en el volumen de vista.

Como este algoritmo opera con precisión de imagen, está sujeto a artefactos de discretización (aliasing, ver pág. 100). El algoritmo de A_Buffer, que se presenta a continuación, trata este problema usando una aproximación discreta para el muestreo de área no ponderada (ver pág. 103).

5.2. A_Buffer

Este algoritmo es el que utilizamos en nuestro código dado que como veremos a lo largo de este trabajo la estructura de datos permite el tratamiento de los distintos temas que proponemos en el desarrollo del mismo.

A pesar de ser extremadamente rápido y simple el Z_Buffer provoca mucho aliasing y no puede renderear objetos transparentes correctamente. Catmull introdujo una técnica para superar esto donde el color de un pixel esta dado por el promedio pesado de todos los fragmentos de polígonos visibles del pixel. Lo que se necesita es un método que combine la simplicidad y velocidad

del Z_Buffer y los beneficios de antialiasing del proceso de Catmull. El método debe soportar todas las posibles primitivas geométricas que se pueden modelar. Debe poder manejar transparencias y superficies intersectantes (además de superficies transparentes intersectantes).

La información geométrica dentro de un pixel es vital para mostrar correctamente un pixel. Es necesario usar más resolución. Después de algunos estudios una máscara de bits de 4 x 8 (figura 5.1) fue seleccionada para representar los subpixeles de un polígono. Hacer un clipping de un polígono contra otro se vuelve entonces una simple operación booleana. Los contornos de las figuras aún exhiben algunos efectos producto de la cuantificación, así que el área actual de la pantalla del tamaño de los subpixeles de los polígonos son guardados con las máscaras. Cuando es posible el área actual es usada en vez de contar los bits en la máscara.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figura 5.1: Máscara de bits de pixel

5.2.1. Algoritmo del A_Buffer

El A_Buffer trabaja con dos tipos de datos diferentes: "pixelstruct" y "fragments" [Carpenter84]. Un pixelstruct son dos palabras de 32 bits (figura 5.2), una conteniendo la profundidad en Z y la otra conteniendo o bien un color o bien un puntero. Un fragmento (figura 5.3) es un polígono cortado en los límites del pixel. Pixelstuct aparece en un arreglo del tamaño y forma del tamaño de la imagen final (como el Z_Buffer). Si un pixel es simple, es decir está completamente cubierto por un polígono, el valor de Z es positivo y pixelstruct tiene el color. Si no el valor de Z es negativo y el puntero apunta a una lista de fragmentos ordenados de adelante hacia atrás por el z más cercano.

```
float          z;          /*Z negativo*/
fragment_ptr   flist;      /*nunca nulo*/
                  (o)
float          z           /*Z positivo*/
byte           r,g,b;      /*color*/
byte           a;          /*covarage*/
```

Figura 5.2: definición de Pixelstruct[3]

```
fragment_ptr   next;
short_int      r,g,b;                /* color, 12 bits*/
short_int      opacidad;             /*1 - transparencia*/
short_int      área;                 /*12 bits de precisión*/
short_int      marcador del objeto;  /*desde la superficie padre*/
pixelmask      m;                    /*4 x 8 bits*/
float          z_max, z_min;         /*positivo*/
```

Figura 5.3: definición de un fragmento

En nuestro algoritmo pixelstruct es una lista de fragmentos donde cada fragmento está constituido de la forma descripta en la figura 5.3.

La siguiente discusión contiene algunos símbolos que definimos a continuación:

M: una máscara de 4 X 8

A: área (0..1)

C: color (r,g,b)

Opacidad: 1 - fracción de transmisión

α: coverage

[3] Definición utilizada por Carpenter en su trabajo original.

Ordenar en Z es necesario por dos razones. El cálculo adecuado de transparencias requiere que todas las superficies transparentes estén ordenadas en Z. El otro beneficio de ordenar por valores de Z es que los fragmentos del mismo objeto tienden a estar agrupados juntos en la lista y entonces pueden ser fácilmente mezclados. Por ejemplo una figura bicúbica puede ser dividida en muchos polígonos. Estos polígonos provienen todos de la misma superficie, pero podrían ser dividida en fragmentos en un orden impredecible (dependiendo de la orientación de la pantalla, etc.) (figura 5.4). Mezclar dos o más fragmentos simplifica la estructura de datos y modifica el espacio usado por los fragmentos mezclados. Si el resultado es opaco y cubre todo el pixel, no podemos con certeza modificar los fragmentos escondidos, como si ellos fueran parte de una superficie intersectante incompleta. Recordemos que los fragmentos son partes de un mismo pixel y son mezclados para mejorar la precisión de la imagen final cuando esta es muestreada.

Los objetos son mezclados si y sólo si tienen el mismo identificador de objeto y ellos se superponen en Z. Este testeo es implementado cuando un nuevo fragmento es agregado a la lista. Los identificadores de objetos son enteros asignados a objetos geométricos primitivos que no intercecan consigo mismos, como por ejemplo esferas. El identificador es completado por un bit indicando cuando la cara de la superficie es delantera o trasera para prevenir que se mezclen fragmentos en forma inadecuada. Si los fragmentos no se superponen en la pantalla ($M1 \cap M2 = \varnothing$), entonces las máscaras son ordenadas, los colores son mezclados:

$$C = C1 \times A1 + C2 \times A2$$

y las áreas son sumadas. Si los fragmentos se superponen (lo que es altamente improbable), ellos son divididos en tres partes:

$$M_{front\_only} = M_{front} \cap \sim M_{back}$$

$$M_{back\_only} = M_{back} \cap \sim M_{front}$$

$$M_{overlap} = M_{back} \cap M_{front}$$

La contribución del fragmento frontal es computada:

$$\alpha_{front} = A_{front\_only} + Opacity_{front} \times A_{overlap}$$

los colores mezclados:

$$C = \alpha_{front} \times C_{front} + (1-\alpha_{front}) \times C_{back}$$

y el área computada es :

$$A = A_{front} + A_{back} \times A_{back\_only} / (A_{back\_only} + A_{overlap})$$

Cuando ningún fragmento más debe ser enviado a Pixelstruct, el color de Pixelstruct es determinado y escrito dentro del cuadro. Generalmente, el pixel será completamente cubierto por algún objeto y unos pocos fragmentos del tamaño del pixel sobrarán. Si un fragmento está presente un proceso de packing recursivo es llamado. En el gráfico 5.4 vemos la división de un polígono en fragmentos y el proceso de mezcla de dos fragmentos.

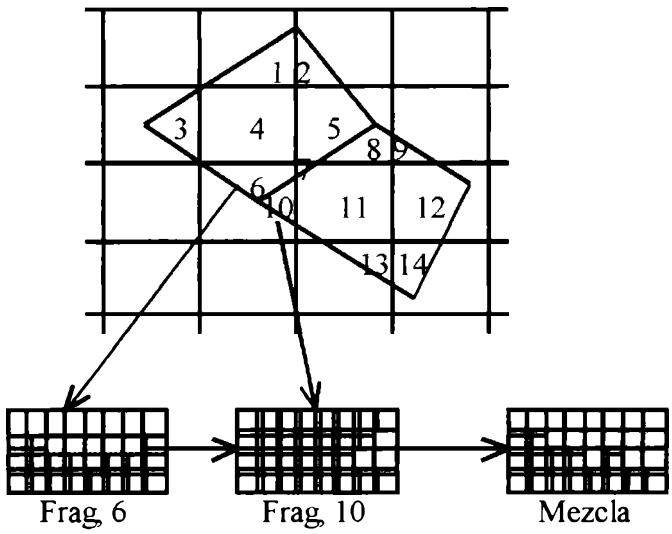


Figura 5.4: Proceso de mezcla de fragmentos de un mismo objeto

5.2.1.1. *Rutina de packing de los fragmentos*

Promediar el área significa que el color de un pixel es computado por el promedio pesado de los colores de las superficies visibles contenidas en el pixel. El problema es, entonces, como determinar los fragmentos visibles, y las partes visibles de los fragmentos.

Para entender el método usado en el A_Buffer consideremos el siguiente ejemplo simplificado. Asumamos, por el momento, que no hay transparencias ni superficies intersecantes. Si el fragmento que está al frente de la lista cubre el pixel, entonces estamos hechos, de otro modo cubre sólo parte del pixel. Nosotros dividimos el pixel en dos partes, dentro y fuera, usando la máscara del fragmento. La contribución de la parte de adentro es el color del fragmento pesado por su área. La contribución de la parte de afuera es algún color que aún debe ser descubierto pesado por el complemento del área del fragmento.

$$C = C_{in} \times A_{in} + C_{out} \times (1-A_{in})$$

El color que aún debe ser descubierto es encontrado recursivamente llamando a la rutina de packing con la parte de afuera de la máscara que representa el resto del pixel y un puntero al próximo ítem en la lista de fragmentos.

Podemos descubrir el método ahora en más detalle. Comenzamos el proceso de packing con una máscara de búsqueda de 32 bits que representa el pixel completo. Los fragmentos son considerados solamente si se superponen a la máscara de búsqueda. Cuando todo o parte de un fragmento es encontrado dentro de la máscara de búsqueda, la misma es particionada usando la máscara del fragmento.

$$M_{in} = M_{search} \cap M_f$$

$$M_{out} = M_{search} \cap \sim M_f$$

Si M_{out} <> 0 usamos un llamado recursivo con M_{out} como máscara de búsqueda para encontrar el color del resto del área buscada. Si el fragmento es transparente, un llamado recursivo usando M_{in} como máscara de búsqueda es usado para encontrar el color de las superficies detrás del fragmento para ser filtrado con el color del fragmento.

$$C_{in} = Opacity_f \times C_f + (1-Opacity_f) \times C_{behind}$$

El coverage es computado similarmente:

$$\alpha_{in} = Opacity_f \times \alpha_f + (1-Opacity_f) \times \alpha_{behind}$$

Si el fragmento no es transparente, su color es suficiente para C_{in}. Cuando tenemos los colores de las regiones de adentro y de afuera mezclamos a éstos pesados por su coverage:

$$C_{returned} = (\alpha_{in} \times C_{in} + \alpha_{out} \times C_{out}) / (\alpha_{in} + \alpha_{out})$$

Usamos el número de bits unos en la máscara para estimar el área.

5.2.1.2. *Ahora para intersecciones.*

Pixeles donde las superficies intersecantes son visibles usualmente son decenas o centenas en un cuadro de resolución 521 X 512. Además el antialiasing en una línea de intersección no es tan crítico como en el contorno, por ejemplo porque el contraste es más bajo. Estas observaciones sugieren que podemos tomar una aproximación simple. Como la información sobre orientación (vértices y ecuaciones del plano) no es guardada en un fragmento, nosotros definimos que una intersección sucede cuando el identificador del objeto difiere y los fragmentos se superponen en Z. Esto trabaja satisfactoriamente en todos los casos a excepción de unos pocos. Como nosotros no sabemos exactamente cuanto del fragmento frontal es visible, estimamos este valor desde los valores máximo y mínimo de Z. (ver figura 5.5).

$$Vis_{front} = (Zmax_{next} - Zmin_{front}) / ((Zmax - Zmin)_{front} + (Zmax - Zmin)_{next})$$

Dado que parte del fragmento frontal oscurece el próximo fragmento y viceversa, nosotros necesitamos estimar el factor pesado para usarlo al mezclar los colores de los dos fragmentos,

$$\alpha_{in} = Vis_{front} \times Opacity_{front} + (1-Vis_{front}) \times (1-\alpha_{next})$$

Esto es la suma de la parte no oscurecida del fragmento frontal y la parte del fragmento frontal filtrada por los otros fragmentos.

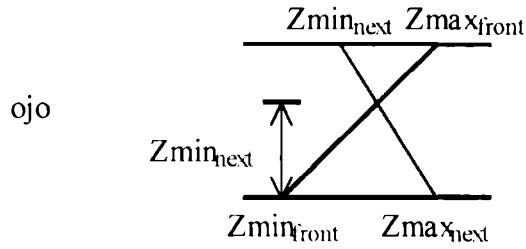


Figura 5.5: Fracción visible del fragmento

Dados estos factores, mezclamos el fragmento frontal con el otro fragmento en la máscara de adentro,

$$C_{in} = \alpha_{in} \times C_{front} + (1 - \alpha_{in}) \times C_{next}$$

Luego mezclamos las partes de adentro y de afuera,

$$C_{returned} = (\alpha_{in} \times C_{in} + \alpha_{out} \times C_{out}) / (\alpha_{in} + \alpha_{out})$$

El pseudocódigo de la rutina de packing es el siguiente:

```
Pack_Under_Mask(Fragment_ptr, SearchMask, Colour, Coverage)
si es el último fragmento de la lista
   retornamos su color y coverage
sino
   calculamos las máscaras de adentro y de afuera del fragmento
   si la máscara de afuera no está vacía
      calculamos el color y coverage del área de afuera
                  \* llamada recursiva con la máscara de afuera *\
   si el fragmento es transparente o se superpone en Z con el próximo fragmento
de la lista
      calculamos el color y coverage de lo que está atrás
                  \* llamada recursiva con la máscara de adentro *\
   si nada de lo oculto detrás del fragmento afecta su apariencia
      retornamos la mezcla del fragmento y el área de afuera
   sino
      si se superpone en Z con el próximo
         estimamos el radio de visibilidad
         estimamos el coverage del fragmento
         mezclamos el fragmento con lo de atrás a éste
         retornamos la mezcla de adentro y de afuera
      sino
         mezclamos el fragmento con lo de atrás a éste
         retornamos la mezcla de adentro y de afuera
end
```

6. Modelos de iluminación y sombreado

Hablamos del término modelo de sombreado cuando nos referimos al esquema general dentro del cual se encuentra el modelo de iluminación. El modelo de iluminación determina como sombrear superficies basándonos en la posición, las características de las superficies y las fuentes luminosas que la iluminan. Hay varios modelos de iluminación que determinan el color de una superficie en un punto específico. La función del modelo de sombreado es determinar cuando se aplica el modelo de iluminación y que argumentos recibe. Por ejemplo, algunos modelos de sombreado indican distintos modelos de iluminación para cada pixel de la imagen, mientras que otros invocan un modelo de iluminación sólo para ciertos pixeles y luego sombrean a los demás por interpolación.

La interacción entre luces y sombras es compleja. Muchas veces se han deducido las reglas que sirven de base a la óptica y a la radiación térmica, ya sea para simplificar los cálculos o porque no se conocían modelos más precisos. Por lo tanto muchos modelos que se han utilizado tradicionalmente en la computación gráfica incluyen muchos trucos y simplificaciones que no tienen bases teóricas firmes, pero que en la práctica funcionan bien.

6.1. Luz Ambiental

El modelo de iluminación más sencillo es aquel en el que cada objeto se presenta con una intensidad intrínseca. Este modelo no tiene una fuente de luz externa, podemos considerarlo como la descripción de un mundo ligeramente irreal de objetos no reflejantes y autoluminosos.

La ecuación de iluminación que expresa este sencillo modelo es:

$$I = k_i$$

donde I es la intensidad resultante y el coeficiente k_i es la intensidades intrínseca del objeto. Como esta ecuación de iluminación no contiene términos que dependan de la posición del punto que se sombrea, se puede evaluar una vez para cada objeto. Este proceso de evaluación se conoce como iluminación del objeto.

Imagine ahora que en lugar de autoluminosidad hay una fuente luminosa difusa, no direccional, producto de reflexiones múltiples de la luz en las superficies presentes en el ambiente. Esto se conoce como luz ambiental. Si suponemos que la luz ambiental afecta de la misma forma a todas las superficies desde todas las direcciones, nuestra ecuación de iluminación se convierte en

$$I = I_a k_a$$

I_a es la intensidad de la luz ambiental, que se supone constante para todos los objetos. La cantidad de luz ambiental reflejada por la superficie de un objeto está determinada por k_a, el coeficiente de reflexión ambiental, que varía entre 0 y 1. El coeficiente de reflexión ambiental es una propiedad material, se puede considerar como la caracterización del material que conforma la superficie. Este es una conveniencia empírica y no corresponde directamente a ninguna propiedad física de los materiales reales.

6.2. Reflexión difusa

Consideremos ahora la iluminación de un objeto a partir de una fuente luminosa puntual, cuyos rayos emanan uniformemente en todas las direcciones a partir de un solo punto.

La superficies opacas y mates, como la tiza exhiben reflexión difusa. Estas superficies aparecen con la misma brillantez desde todos los ángulos de observación porque reflejan la luz con igual intensidad en todas las direcciones. Para una superficie, la brillantez depende sólo del ángulo θ entre la dirección L a la fuente luminosa y el normal a la superficie N de la figura 6.1. Esto ocurre por dos motivos: en primer lugar porque un haz que intercepta una superficie cubre un área cuyo tamaño es inversamente proporcional al coseno del ángulo θ entre el haz y N.

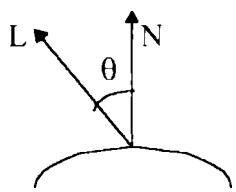


Figura 6.1: Angulo entre la dirección de la fuente luminosa y el normal

En segundo lugar debemos considerar la luz que ve el observador. Las superficies que tienen una reflexión difusa o Lambertiana tienen la propiedad de que la cantidad de luz que reflejan de un área diferencial *dA* hacia el observador es directamente proporcional al coseno del ángulo entre la dirección del observador y N (ver figura 6.2).

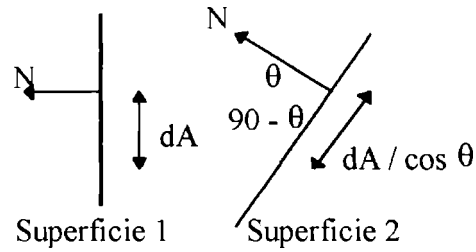


Figura 6.2: un haz (sombreada) de área de sección transversal *dA* en un ángulo de incidencia θ, intercepta un área de *dA* /cos θ

La ecuación de iluminación difusa es

$$I = I_p k_d \cos\theta \qquad (1)$$

I_p es la intensidad de la fuente luminosa puntual; el coeficiente de reflexión difusa k_d del material es una constante entre 0 y 1 que varía de una material a otro. El ángulo θ debe estar entre 0° y 90° para que la luz tenga efecto directo en el punto que se sombrea.

Suponiendo que los vectores **N** y L se normalizaron, podemos reescribir la ecuación (1) usando el producto el escalar:

$$I = I_p k_d (\mathbf{N} \bullet L) \qquad (2)$$

Los objetos iluminados usando la ecuación (2) se ven duros, como al iluminar con una linterna un objeto en una habitación a oscuras. Por lo tanto es común añadir un término de ambiente para obtener una ecuación de iluminación más realista:

$$I = I_a k_a + I_p k_d (\mathbf{N} \bullet L). \qquad (3)$$

Si las proyecciones de dos superficies paralelas de idéntico material, iluminadas desde el ojo, se sobreponen en una imagen, la ecuación (3) no distinguirá donde termina una superficie y comienza la otra sin importar cuan diferente sea su distancia a la fuente luminosa. Para ello, se introduce un factor de atenuación de fuente luminosa, f_{att}, para obtener:

$$I = I_a k_a + f_{att} I_p k_d (\mathbf{N} \bullet L). \qquad (4)$$

Una elección obvia de f_{att} tiene en cuenta el hecho de que la energía de una fuente luminosa puntual que llega a una parte de una superficie disminuye en proporción inversa al cuadrado de la distancia que viaja la luz de la fuente puntual a la superficie.

Sin embargo, en la práctica esto no siempre funciona muy bien. Si la luz está muy lejos, f_{att} no varía mucho; si está muy cerca varía considerablemente ocasionando sombras muy variables en superficies con el mismo ángulo θ

entre N y L. Aunque este comportamiento es correcto para una fuente luminosa puntual, los objetos que vemos en la vida real generalmente no están iluminados por fuentes puntuales.

Hasta ahora hemos descripto luces y superficies monocromáticas. Las luces y superficies coloreadas se tratan en forma común escribiendo ecuaciones distintas para cada componente del modelo de color. El color difuso de un objeto se representa con un valor de O_d para cada componente del sistema de colores RGB. En este caso, los tres componentes primarios de la luz iluminadora, I_{pR}, I_{pG} e I_{pB} se reflejan en proporción a K_dO_{dR}, K_dO_{dG} y K_dO_{dB}, respectivamente. Por lo tanto para el componente rojo:

$$I_R = I_{aR}\, k_a\, O_{dR} + f_{att}\, I_{pR}\, k_d\, O_{dR}\, (\mathbf{N}\bullet\mathbf{L}). \quad (5)$$

Se emplean ecuaciones similares para I_G e I_B.

Aquí hacemos una suposición de simplificación: que un modelo de colores de tres componentes puede modelar totalmente la interacción de la luz con los objetos. Esta suposición es errónea, pero resulta fácil de implantar y muchas veces produce imágenes aceptables. En teoría, la ecuación de iluminación debe evaluarse en forma continua en toda la gama espectral que se modela; en la práctica, se evalúa para varias muestras espectrales discretas. En lugar de limitarnos a un modelo de color en particular, indicamos explícitamente los términos en una ecuación de iluminación que dependen de la longitud de onda, colocando un subíndice λ. De esta manera, la ecuación (5) se convierte en:

$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + f_{att}\, I_{p\lambda}\, [k_d\, O_{d\lambda}\, (\mathbf{N}\bullet\mathbf{L})]. \quad (6)$$

6.3. Reflexión especular

La reflexión especular se puede observar en cualquier superficie brillante. Por ejemplo en una manzana iluminada con una luz blanca brillante se observa un punto de máximo brillo (highligth) ocasionado por la reflexión especular, mientras que la luz reflejada del resto de la manzana es el resultado de la reflexión difusa. En el punto de máximo brillo la manzana no aparece roja, sino blanca. Los objetos como las manzanas enceradas o los plásticos brillantes tienen superficie transparente; los plásticos, por ejemplo, generalmente están compuestos por partículas de pigmento incorporadas en un material transparente. La luz que se refleja especularmente de la superficie incolora tiene un color muy similar a la fuente luminosa.

Si se mueve el punto de visión también se mueve el punto de máximo brillo. Esto sucede porque la superficie brillante refleja la luz en forma desigual en distintas direcciones: en una superficie perfectamente brillante como un espejo perfecto, la luz se refleja sólo en la dirección de reflexión R, que es L invertida con respecto a N. De esta manera, el observador puede ver la luz

reflejada especularmente en un espejo sólo si el ángulo α entre el vector de reflexión R y la dirección V al punto de observación es cero (ver figura 6.3).

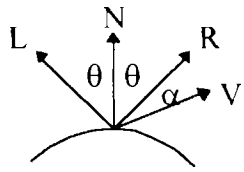


Figura 6.3: reflexión especular

6.4. Modelo de iluminación de Phong

Phong Bui-Thong desarrolló un popular modelo de iluminación para reflectores imperfectos, como la manzana [Foley96]. El modelo supone que la máxima reflexión ocurre cuando α es cero y decrece rápidamente conforme aumenta α. Esta caída es aproximada con $\cos^n\alpha$, donde n es el componente de la reflexión especular del material. Los valores típicos de n varían entre 1 y varios cientos, dependiendo del material de superficie que se simule. Un valor de 1 simula una caída amplia, suave, mientras que los valores más elevados simulan un punto de máximo brillo definido y enfocado (ver figura 6.4). Para un reflector perfecto n sería infinito. El modelo de iluminación de Phong se basa en el trabajo anterior de investigadores como Warnock, quienes usaron el término $\cos^n\alpha$ para modelar la reflexión especular con la luz en el punto de observación. Sin embargo, Phong fue el primero en tener en cuenta observadores y luces en posiciones arbitrarias.

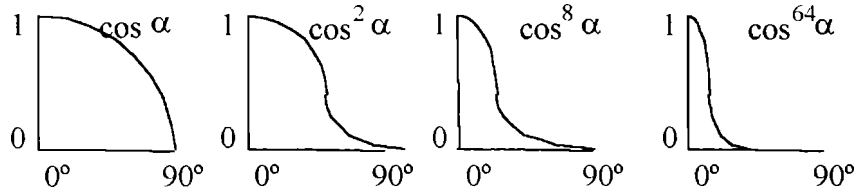


Figura 6.4: Valores diferentes de $\cos^n\alpha$ usados en el modelo de Iluminación de Phong

La cantidad de luz incidente que se refleja especularmente depende del ángulo de incidencia θ (entre la fuente luminosa y el normal a la superficie). Si $W(\theta)$ es la fracción de luz reflejada especularmente, el modelo de Phong es:

$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + f_{att}\, I_{p\lambda}\, [k_d\, O_{d\lambda} \cos\theta + W(\theta) \cos^n\alpha]. \qquad (7)$$

Si se normalizan la dirección de reflexión R y la dirección al punto de observación V, entonces el $\cos\alpha = R \bullet V$. Además $W(\theta)$ suele asignarse igual a una constante k_s, el coeficiente de reflexión especular del material, que varía entre 0 y 1. El valor de k_s se selecciona experimentalmente para producir resultados atractivos desde el punto de vista estético. Así la ecuación (7) se puede reescribir como:

$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + f_{att}\, I_{p\lambda}\, [k_d\, O_{d\lambda}\, (N \bullet L) + k_s\, (R \bullet V)^n]. \qquad (8)$$

El color en el componente especular del modelo de iluminación de Phong no depende de ninguna propiedad material: por lo tanto este modelo produce buenos resultados al modelar reflexiones especulares de superficies plásticas. Pero en realidad la reflexión especular es afectada por las propiedades de la superficie y, por lo general puede tener un color distinto al de la reflexión difusa cuando la superficie está compuesta por varios materiales. Podemos tener en cuenta esta efecto con una primera aproximación modificando la ecuación (8) para obtener:

$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + f_{att}\, I_{p\lambda}\, [k_d\, O_{d\lambda}\, (N \bullet L) + k_s\, O_{s\lambda}\, (R \bullet V)^n]. \qquad (9)$$

donde $O_{s\lambda}$ es el color especular del objeto.

6.5. Modelos de iluminación físicos

El modelo de iluminación analizado en la sección anterior es en gran parte el resultado de un enfoque práctico, de sentido común con respecto a los gráficos. Ahora veremos los modelos de iluminación con una base física [Foley96], basándonos en parte en el trabajo de Cook y Torrance.

Hasta ahora hemos usado la palabra intensidad sin definirla, refiriéndonos informalmente a la intensidad de una fuente luminosa, de un punto en una superficie o de un pixel. Es hora de formalizar nuestros términos. Comenzaremos por el **flujo**, que es la tasa con la cual se emite energía luminosa, y se mide en watts (W). Para referirnos a la cantidad de flujo recibida o emitida en una dirección determinada, se requiere el concepto de **ángulo sólido**, que es el ángulo en el ápice de un cono. El ángulo sólido se mide en función del área sobre una esfera interceptada por un cono cuyo ápice está en el centro de una esfera. Un **estereorradián** (sr) es el ángulo sólido de uno de estos conos que intercepta un área igual al cuadrado del radio r de la esfera. Si un punto está en una superficie nos interesa el hemisferio sobre él. Como el área de una esfera es $4\pi r^2$, hay $4\pi r^2 / 2r^2 = 2\pi$ sr en un hemisferio. Imagine la proyección de la forma de un objeto a un hemisferio centrado alrededor de un punto en la superficie que sirve como centro de proyección. El ángulo ω subtendido por el objeto es el área en el hemisferio ocupada por la proyección, dividida entre el cuadrado del radio del hemisferio (la división elimina la dependencia del tamaño del hemisferio).

La **intensidad radiante** es el flujo que se radia a un ángulo sólido unidad en una dirección específica y se mide en W / sr. Cuando hemos usado la palabra intensidad haciendo referencia a una fuente puntual, nos hemos referido a su intensidad radiante.

La **radiancia** es la intensidad radiante por área unidad de superficie escorzada frontalmente y se mide en W / ($sr \cdot m^2$). El **área de superficie recortada frontalmente**, también conocida como **área de superficie**

proyectada, se refiere a la proyección de la superficie sobre el plano perpendicular a la dirección de la radiación. El área de superficie escorzada frontalmente se determina multiplicando al área de la superficie por el cos θ_r, donde θ_r es el ángulo de la luz radiada con respecto a la normal a la superficie. Un ángulo sólido pequeño $d\omega$ se puede aproximar como el área de superficie escorzada frontalmente, dividida entre el cuadrado de la distancia del objeto al punto donde se calcula el ángulo sólido. Cuando empleamos la palabra intensidad haciendo referencia a una superficie nos estamos refiriendo a su radiancia. Por último la **irradiancia**, también conocida como **densidad de flujo**, es el flujo incidente por unidad de área de superficie (recortada frontalmente) y se mide en W / m^2.

En los gráficos nos interesa la relación entre la luz que incide sobre una superficie y la luz que se refleja y transmite en ella. Considere la figura 6.5 . La irradiancia de luz incidente es:

$$E_i = I_i\,(N \bullet L)\,d\omega_i,$$

donde I_i es la radiancia de la luz incidente y N•L es $\cos\theta_i$. Como la irradiancia se expresa por unidad de área, mientras que la radiancia se expresa por unidad de área escorzada frontalmente, se multiplica por N•L para convertirla al equivalente por unidad de área no recortada frontalmente.

$$E_i = I_i \cos\theta_i\, d\omega_i$$

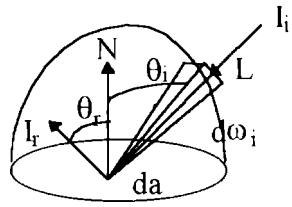


Figura 6.5: Radiancia reflejada e irradiancia incidente

No es suficiente considerar sólo I_i (la radiancia incidente) al determinar i_r (la radiancia reflejada); hay que considerar en su lugar E_i (la irradiancia incidente). Por ejemplo un haz incidente que tiene la misma intensidad radiante (W/sr) que otro haz, pero con mayor ángulo sólido, tiene una E_i proporcionalmente mayor y hace que la superficie aparezca proporcionalmente más brillante. La razón entre la radiancia reflejada (intensidad) en una dirección y la irradiancia incidente (densidad de flujo) responsable de ella en otra dirección, se conoce como **reflectividad bidirecional**, ρ, que es una función de las direcciones de incidencia y reflexión,

$$\rho = I_r/E_i.$$

Como hemos visto en la graficación por computador es convencional considerar la reflectividad bidireccional como compuesta por componentes difusos y especulares. Por lo tanto,

$$\rho = k_d\rho_d + k_s\rho_s,$$

donde ρ_d y ρ_s, son, respectivamente, los coeficientes de reflexión difusa y especular.

El modelo de superficie de Torrance-Sparrow, desarrollado por físicos, es un modelo físico de superficie reflectante. Blinn fue el primero en adaptar el modelo de Torrance-Sparrow a la graficación por computador, proporcionando detalles matemáticos y comparándolo con el modelo de Phong, Cook y Torrance fueron los primeros en aproximar la composición espectral de la luz reflejada en una implantación del modelo.

En el modelo de Torrance-Sparrow, se supone que la superficie es una colección isotrópica de facetas microscópicas planas, cada una de ellas un reflector perfectamente liso. La geometría y distribución de estas **microfacetas** y la dirección de la luz (que se supone emana de una fuente infinitamente distante, de manera que todos los rayos son paralelos) determinan la intensidad y la dirección de la reflexión especular como función de I_p (la intensidad de la fuente luminosa puntual), N, L y V. Las mediciones experimentales indican que hay muy buena correspondencia entre la reflexión real y la reflexión pronosticada por este modelo.

Para el componente especular de la reflectividad bidireccional, Cook y Torrance usan

$$\rho_s = (F_i/\pi) \bullet (DG/((N*V)(N*L))),$$

donde D es una función de distribución de las orientaciones de las microfacetas, G es el **factor de atenuación geométrica**, que representa los efectos de enmascaramiento y sombreado de las microfacetas entre sí y F_λ es el término de Fresnel calculado usando la ecuación de Fresnel, la cual, para la reflexión especular, relaciona la luz incidente con la luz reflejada para la superficie lisa de cada microfaceta. El término π supuestamente debe considerar la aspereza de la superficie. El término N•V hace la ecuación proporcional al área de la superficie (y por ende del número de microfacetas) que ve el observador en una porción unidad de área de superficie recortada puntualmente, mientras que el término N•L hace que la ecuación sea proporcional al área de superficie que la luz ve en una porción unidad de área de superficie recortada frontalmente.

6.6. Modelo de sombreado de Gouraud

El sombreado de Gouraud [Foley96], también llamado sombreado de intensidad o de interpolación de color, elimina la discontinuidad de intensidad, si bien no elimina por completo las tiras brillantes ocasionadas por cambios rápidos en la curva de intensidad.

El sombreado interpolado toma la información de sombreado de un triángulo a partir de sus vértices y luego la interpola. Este método es muy fácil de usar en un algoritmo de línea de barrido que interpola el valor z en un tramo a partir de valores z interpolados que se calculan para los puntos extremos del tramo.

El sombreado de Goraud extiende el concepto de sombreado interpolado aplicado a polígonos individuales, interpolando los valores de iluminación de los vértices de polígonos que toman en cuenta la superficie que se aproxima. El proceso de sombreado de Gouraud requiere que se conozca el normal a cada vértice de la malla poligonal. Gouraud pudo calcular estos normales a los vértices directamente de la descripción analítica de la superficie. Alternativamente si los normales a los vértices no se almacenan con la malla y no se pueden determinar directamente de la superficie real, Gouraud propone que se aproximen promediando los normales a la superficie de todas las caras poligonales que comparten cada vértice (ver figura 6.6).

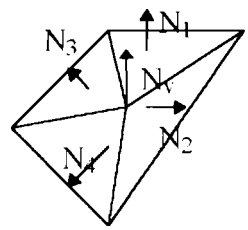


Figura 6.6: Los normales a la superficie de un polígono normalizado se pueden promediar para obtener el normal a los vértices

El siguiente paso en el modelo de Gouraud es encontrar intensidades de vértices usando los normales a los vértices con cualquier modelo de iluminación que se desee. Finalmente cada polígono se sombrea por interpolación lineal de las intensidades de los vértices a lo largo de cada arista y luego entre las aristas de cada línea de barrido.

La interpolación a lo largo de aristas se puede integrar fácilmente con el algoritmo de línea de barrido para superficies visibles. Para cada arista se almacena la intensidad inicial y el cambio de intensidad de cada componente de color por cada cambio de unidad en y. Un tramo visible en una línea de rastreo se rellena interpolando los valores de intensidad de las dos aristas que acotan el tramo.

6.7. Modelo de sombreado de Phong

El sombreado de Phong [Foley96], también conocido como sombreado de interpolación del vector normal, interpola el vector normal a la superficie, N, en lugar de la intensidad. La interpolación ocurre en un tramo del polígono sobre una línea de rastreo entre el normal inicial y final del tramo. A su vez estos normales se interpolan en las aristas de los polígonos a partir de normales a vértices que se calculan, de ser necesario como en el sombreado

de Gouraud. La interpolación a lo largo de aristas se puede efectuar con cálculos incrementales, aumentando las tres componentes del vector normal de una línea de rastreo a otra. En cada pixel de una línea de rastreo el normal interpolado se normaliza y se establece la correspondencia de regreso al sistema de coordenadas del mundo, donde se calcula una nueva intensidad usando cualquier modelo de iluminación. En la figura 6.7 se muestran dos normales a las aristas y los normales interpolados a partir de ellos, antes y después de la normalización.

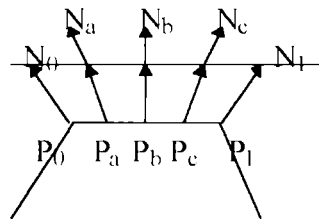


Figura 6.7: Interpolación de vectores normales

6.8. Sombreado de Phong vs. sombreado de Gouraud

El sombreado de Phong produce mejoras notables con respecto al sombreado de Gouraud, ya que los puntos de máximo brillo se producen de manera más fiel. Consideremos que pasa si la n en el término de iluminación $\cos^n\alpha$ de Phong es grande y uno de los vértices tiene α muy pequeña, pero sus vértices adyacentes tienen α muy grande. La intensidad asociada al vértice con α pequeña será apropiada a un punto de máximo brillo, mientras que los otros vértices no tendrán intensidades de punto de máximo brillo. Si se emplea el sombreado de Gouraud, la intensidad del polígono se interpola linealmente entre el punto de máximo brillo y las intensidades menores de los vértices adyacentes, distribuyendo el punto de máximo brillo por el polígono. Cuando se usan normales interpolados linealmente para calcular el término $\cos^n\alpha$ en cada pixel hay una rápida caída en la intensidad del punto de máximo brillo. Además si un punto de máximo brillo no cae sobre un vértice es posible que el sombreado de Gouraud lo omita, ya que ningún punto interior puede ser más brillante que el vértice de mayor brillo a partir del cual se interpola. En cambio el sombreado de Phong permite localizar realces en el interior de un polígono.

6.9. Problemas con el modelo interpolado

Todos estos modelos interpolados tienen algunos problemas comunes, algunos de los cuales describiremos a continuación:

Silueta poligonal: no importa cuan buena sea la aproximación de un modelo de sombreado interpolado con respecto al sombreado real de una superficie curva, la orilla de la silueta de la malla es poligonal.

Distorsión de perspectiva: se introducen anomalías porque la interpolación se lleva a cabo después de la transformación de perspectiva en el sistema de coordenadas de pantalla, en lugar del sistema de coordenadas del mundo.

Dependencia de la orientación: los resultados de los modelos de sombreado por interpolación no son independientes de la orientación del polígono proyectado. Como los valores se interpolan entre vértices y a través de líneas de rastreo horizontales, los resultados pueden variar al rotar el polígono.

Problemas en vértices compartidos: pueden ocurrir discontinuidades en el sombreado cuando dos polígonos adyacentes no comparten un vértice que está sobre su arista común.

Normales a los vértices no representativos: los normales a vértices adecuadas no siempre representan de manera adecuada la geometría de la superficie. Este problema se resolverá subdividiendo los polígonos antes de calcular los normales a los vértices.

7. Formulación de reflexiones

Para obtener imágenes que simulen mayor realismo también se utilizan modelos de reflexión. Estos modelos pueden simular los reflejos bien definidos de un espejo perfecto que refleja otro objeto o las reflexiones difusas de una superficie menos pulida. Para modelar la reflexión se requieren otras superficies además de la que se está dibujando. Los modelos más complejos incluyen refracción. La transparencia refractiva es un efecto que requiere que los objetos se modelen como sólidos. Tenemos que conocer algo acerca de los materiales por los cuales pasa un rayo luminoso y la distancia que viaja para poder modelar su refracción.

Tanto la reflexión como la refracción surgen de la interacción entre los objetos y las luces que es bastante compleja, por lo que los modelos utilizados han deducido muchas veces las reglas que sirven de base a la óptica y la radiación térmica ya sea porque los investigadores de la comunidad gráfica no conocían modelos más precisos o para simplificar los cálculos.

Para el modelado de reflexiones y refracciones se requieren cálculos adicionales similares a los cálculos para eliminación de superficies ocultas. Estos efectos ocurren porque algunas superficies ocultas en realidad no lo están: se ven a través de otros objetos, son reflejadas en objetos espejados o se refractan o traslucen a través de objetos transparentes.

Nos vemos obligados a realizar una serie de aproximaciones con respecto al comportamiento real de la luz, viéndonos obligados a despreciar otros efectos. Por ejemplo, al utilizar la óptica geométrica para computar reflexiones y refracciones, estamos despreciando los efectos dispersivos y/o interferenciales que puedan ocurrir. Pero esta aproximación está plenamente justificada si consideramos que las longitudes de onda involucradas en las simulaciones de Computación Gráfica se encuentran en el rango del espectro visible (que va desde los $7.8*10^{-7}$ a los $3.8*10^{-7}$ metros), y, por lo tanto, son mucho menores que los objetos usados comúnmente para modelar cualquier escena.

7.1. Reflexión especular

Utilizando la óptica geométrica en una superficie perfectamente brillante como un espejo perfecto, la luz se refleja sólo en la dirección de reflexión R, que es L (la dirección a la fuente luminosa) invertida con respecto a N (el normal a la superficie).

Ahora pensemos en un objeto espejado al que estamos viendo desde un punto de vista particular. Este objeto probablemente refleje sobre su superficie otros objetos de la escena. Para obtener el punto reflejado sobre la superficie debemos tener en cuenta la dirección del rayo de luz incidente y la posición

del vector normal a la superficie, a partir de los cuales se puede calcular la dirección del rayo reflejado R. Es fácil ver que el vector de reflexión R, la posición sobre la superficie del objeto espejado W y el punto reflejado Σ pertenecen al mismo plano, que llamaremos plano de reflexión (ver figura 7.1).

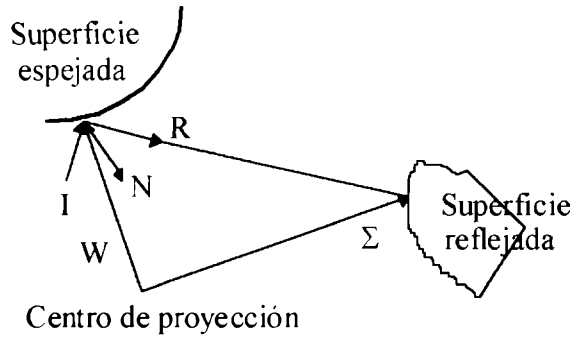


Figura 7.1: Plano de reflexión

7.2. Refracción

Si nos referimos a los objetos transparentes, podemos decir que hay dos tipos de transparencias: la transparencia refractiva es mucho más difícil de modelar que la no refractiva, ya que las líneas de visión geométrica y óptica son diferentes. Si se considera la refracción en la figura 7.2, el objeto A es visible a través del objeto transparente en la línea de visión refractada óptica; si se ignora la refracción, el objeto visible es el B. La relación entre el ángulo de incidencia θ_i y el ángulo de refracción θ_t está indicada por la ley de Snell [Foley96]:

$$\text{sen } \theta_i / \text{sen } \theta_t = \eta_{tz} / \eta_{iz}$$

donde η_{tz} y η_{iz} son los índices de refracción de los materiales por los cuales pasa la luz. El índice de refracción de un material es la relación entre la velocidad de la luz en el vacío y la velocidad de la luz en el material. Este índice varía de acuerdo con la longitud de onda de la luz e incluso con la temperatura. Un vacío tiene índice de refracción 1.0, como la atmósfera con una aproximación cercana; todos los materiales tienen valores más altos. La dependencia de la longitud de onda que tiene el índice de refracción se presenta en muchos casos de refracción como dispersión: el familiar, pero difícil de modelar, fenómeno de la luz refractada que se distribuye en su espectro.

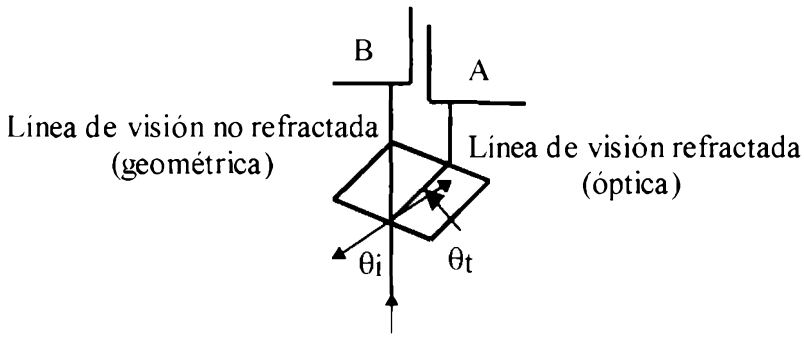


Figura 7.2: Refracción

7.3. Mapa de Ambiente: Modelo de Blinn y Newell

Llamaremos reflexiones interobjeto a aquellas reflexiones que ocurren cuando un objeto refleja otras superficies en su ambiente. Estos efectos varían desde las reflexiones de forma más o menos especular que cambian de acuerdo a la posición del punto de visión, hasta reflexiones difusas que son insensibles al punto de visión.

Blinn y Newell desarrollaron en 1976 un modelo de reflexión especular interobjeto llamado Mapa de Ambiente [Blinn76]. En este modelo se elige un centro de proyección desde el cual se mapea el ambiente que será reflejado. Este mapeo se realiza sobre una esfera que rodea a cada uno de los objetos que va a ser rendereado. Alternativamente se puede utilizar un cubo en lugar de una esfera, y hacer seis proyecciones del ambiente que será reflejado sobre las seis caras del cubo que rodea a cada objeto a ser rendereado (ver figura 7.3). El mapa de ambiente puede ser tratado entonces como una textura 2D. Es este caso del mapa de entorno cúbico al que nos referiremos de acá en más.

El mapeo de reflexiones introducido por Blinn y Newell hace que la reflexión espejo del ambiente sobre un punto de la superficie sea tomada de un punto en la proyección del mundo que corresponde a la dirección de un rayo reflejado por la superficie y mirado desde el centro del mundo. Consecuentemente, las reflexiones son geométricamente exactas solamente si el punto de la superficie está en el centro del mundo o si el objeto reflejado está muy distante. La distorsión geométrica de las reflexiones aumenta cuando la distancia del punto de la superficie al centro del mundo aumenta y la distancia desde el objeto reflejado al centro del mundo decrece.

Cuando aplicamos el mapeo de reflexiones a un objeto particular, el resultado satisfactorio es usualmente obtenido centrando la proyección del mundo en el centro del objeto.

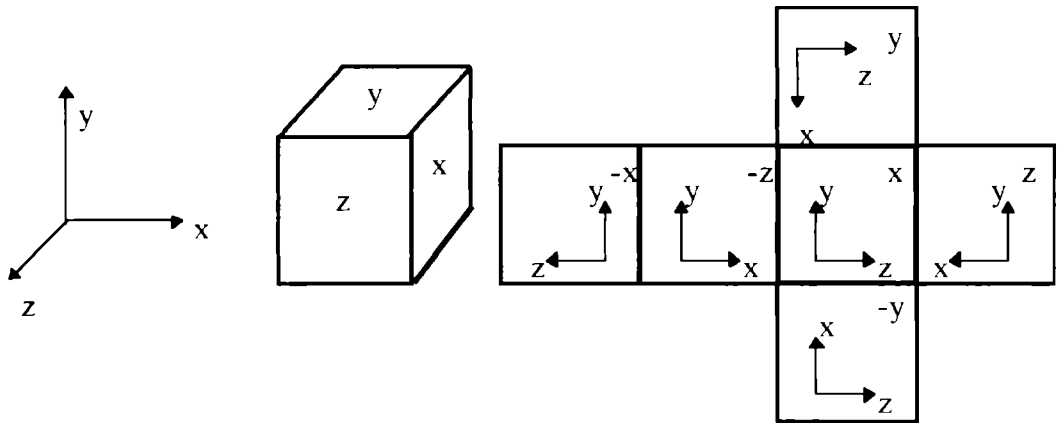


Figura 7.3: Mapa de Entorno Cúbico

7.4. Reflexiones de Ambiente a través de un mapa de ambiente Z_Buffer

Nos referimos en esta parte del trabajo a un algoritmo para calcular reflexiones que guarda el Z_Buffer ya calculado para obtener al mapa de ambiente que se usa para encontrar al mejor aproximador al punto real reflejado, desarrollado por Gustavo Patow, en 1995 [Patow95]. Este implementa una búsqueda en el plano definido por el vector reflejado y la posición de la superficie reflectiva. Para cada vector en este plano el algoritmo calcula cuanto se aproxima al vector de reflexión real y guarda solamente el que se aproxima mejor a la posición reflejada.

Como ya mencionamos anteriormente el vector reflejado R se calcula reflejando el vector incidente I sobre el vector normal a la superficie N. Recordemos que R, la posición sobre la superficie espejada (W) y el punto reflejado Σ pertenecen al mismo plano al que llamamos plano de reflexión. Notemos que todos los orígenes son ubicados en el centro de proyección del mapa de ambiente. De este modo podemos enunciar el siguiente teorema:

La verdadera dirección unitaria reflejada desde el centro de proyección ($P = [[\Sigma]]$, donde $[[A]] = A / ||A||$, para $||A|| <> 0$, significa la normalización a la unidad), podría ser escrito como una combinación lineal de los vectores R y W:

$$P = [\,[\,(1 - t)\,R + t\,W\,]\,] \qquad (1)$$

con t en el rango [0,1] (Esto implica que, cuando $t = 0$, $P = R$; y cuando $t = 1$, $P = [[W]]$.

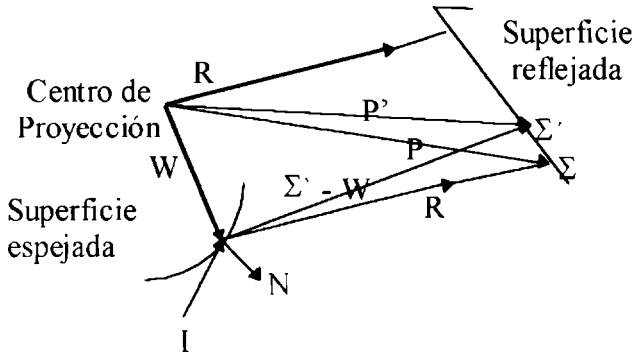


Figura 7.4: Plano geométrico de reflexión

Podemos demostrar este teorema usando el hecho de que Σ es, por su definición $W + \lambda R$ (con λ un real positivo constante apropiado). Entonces, $\Sigma = W + \lambda R = \eta ((1-t)R + tW)$, donde η es otro valor real. De lo anterior se deduce fácilmente que $t = (\lambda + 1)^{-1}$ y $\eta = (\lambda + 1)$; que presenta el comportamiento asimptótico apropiado para $\lambda \rightarrow 0$ y $\lambda \rightarrow \infty$.

Podemos decir que el vector P cae entre los vectores R y W, sobre el mismo plano (ver figura 7.4). En otras palabras lo que el teorema significa es que el punto reflejado desde el centro de proyección del mapa de ambiente (Σ) se encuentra entre la superficie reflectiva W y R. El ángulo entre estos dos últimos está entre $[0,\pi)$.

Por supuesto que cuando hablamos de λ, η o t no los conocemos, así que debemos encontrar una forma alternativa de computarlos. Es aquí cuando la información del Z_Buffer se vuelve útil. Para cada pixel en el mapa de reflexión, se almacena el valor de la distancia desde el centro de proyección al punto reflejado en la superficie (α_i, donde i es cada punto en el mapa de reflexión). De esta forma αP (α es el valor de Z del mapa de ambiente indexado en la dirección de P) representa la posición desde el centro de proyección del punto en la superficie reflectora, ($\Sigma = \alpha P$). Estamos asumiendo un mapa de reflexión idealizado aquí, con resolución infinita, y negando la diferencia entre el vector P y el vector original usado para computar la reflexión.

Como ya hemos mencionado, Σ podría ser escrito como $W + \lambda R$, donde λ es un número real. Así que el problema que resta es minimizar la expresión:

Angulo_entre (R,(Σ' - W)) (2)

para todo Σ' con la restricción que los puntos $\Sigma' = \alpha' P'$ caigan sobre el plano de reflexión, entre R y W. Los valores para α' están almacenados en el Z-Buffer, indexados por P'. Cuando P' es igual a P la ecuación (2) es cero, y toma su valor teórico mínimo.

7.4.1. Algoritmo

El problema puede ser presentado ahora de la siguiente manera: encontrar el vector P´ que mejor aproxima el punto reflejado. Dicho vector cae en el plano de reflexión entre R y W. Esto significa que sólo es necesaria una búsqueda lineal.

Así que lo que nosotros necesitamos es inspeccionar cada vector posible P´, iterando con la ecuación (1). Usar un pequeño valor en t para cada paso, podría resultar en que muchos pixeles del mapa de reflexión podrían ser examinados más de una vez, o si en caso contrario, t fuera muy grande, algunos pixeles no serían examinados. La mejor solución es iterar en la resolución del espacio del mapa de ambiente, recuperando P´ para cada pixel en el plano de reflexión.

Pero, ¿cuáles son los pixeles que se deben inspeccionar? Son aquellos que caen sobre la intersección entre el plano de reflexión (computado por los dos vectores de reflexión) y el mapa de ambiente. Las intersecciones del plano con las caras del cubo (el mapa de ambiente) deben ser calculadas y para esto es necesario usar un algoritmo que dibuje líneas para indexar el mapa.

Como establecimos anteriormente, el mapa de ambiente utilizado en el algoritmo es de naturaleza discreta, por lo que no es verdad que el vector utilizado originalmente para computar el mapa caiga exactamente sobre el plano de reflexión, y no es verdad que el pixel en el mapa fue originado desde el vector de reflexión que estamos buscando. Para resolver parcialmente este problema, cuando se itera en el espacio de la imagen, la posición verdadera del pixel en el mapa es calculada, y el vector original es entonces recuperado. La expresión (2) no dará una relación exacta (el mínimo podría no ser cero). Nosotros deberíamos tratar de encontrar su mínimo para los pixeles del mapa de ambiente que caen sobre el plano de reflexión, entre los dos vectores, y guardar el valor final del pixel solamente si la relación (2) está por debajo de un valor dado (para evitar reflexiones erróneas).

A continuación presentamos el pseudocódigo para el cálculo de reflexiones con una estructura Z_Buffer:

```
Con el normal N y el incidente I encontrar el vector reflejado R
Si R • [[ W ]] = +-1
    computar reflexión en la dirección de R
si no
   para cada pixel en (plano de reflexión n mapa de reflexión)
       encontrar P' desde la posición del pixel
       Tomar a´ del mapa de reflexión y calcular s´
       Si el pixel indexado corrientemente tiene un ángulo menor que el anterior
           guardar el nuevo pixel
   Si el ángulo del mejor aproximador es mayor a un límite tolerable
       refleja el infinito (color del background)
```

7.4.2. Limitaciones del modelo

Como el algoritmo implementa una búsqueda lineal para el mejor aproximador, situaciones como aquella presentada en la figura 7.5 podrían ocurrir: hay algunos puntos como Σ_b que no son visibles desde el punto central (Σ_b no aparece en el mapa de profundidades porque éste está escondido por Σ_a para el centro de reflexión del mapa de ambiente), pero son visibles desde otro punto de la escena. Es por esto que el algoritmo puede fallar en la búsqueda, retornando el color del fondo de la escena. Nosotros llamamos a esto "el problema del punto escondido".

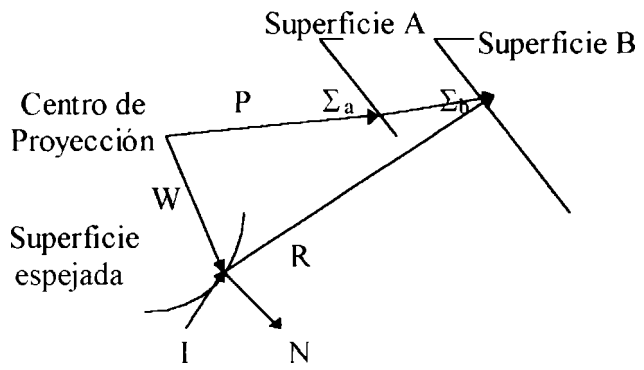


Figura 7.5: Geometría del problema del punto escondido

La solución obvia es almacenar más de un valor de Z para cada pixel, guardándolos en una lista ordenada de profundidades. De esta forma el primer ítem en la lista podría ser el tradicional valor del Z_Buffer para cada pixel. Nos estamos refiriendo en este caso a un mapa de ambiente A_Buffereado.

7.5. Reflexiones desde un mapa de ambiente único A_Buffereado

Este algoritmo (que presentamos en este trabajo), se basa en la utilización de un único mapa de entorno cúbico A_Buffereado, centrado en la cámara de modo tal que una de sus caras coincida con la imagen a ser rendereada finalmente. El algoritmo propuesto trabaja en dos pasadas: primero se computan las seis imágenes que forman el mapa de entorno, y luego se renderea la imagen final.

Una vez relevada toda la información geométrica, los valores finales para cada pixel pueden ser hallados calculando cuáles entradas del A_Buffer contribuyen. Dichas contribuciones se obtienen realizando búsquedas lineales en el mapa de entorno para encontrar los objetos reflejados. Esta búsqueda se realiza entre la posición de la superficie reflectante en el mapa de entorno, W y la posición señalada por el vector reflejado, R (calculado a partir del normal a la superficie N y el vector incidente I). Para poder solucionar el problema mostrado en la figura 6.6, es necesario que el A_Buffer no sólo contenga la

información sobre las caras que miran al observador, sino que tiene que guardar todas las caras porque el punto reflejado en un pixel visible puede pertenecer a la cara trasera de otro objeto. Se puede entender mejor el funcionamiento del algoritmo utilizando la figura 7.6, donde se remarcan los pixeles del mapa de entorno recorridos en la búsqueda lineal realizadas para encontrar el punto reflejado.

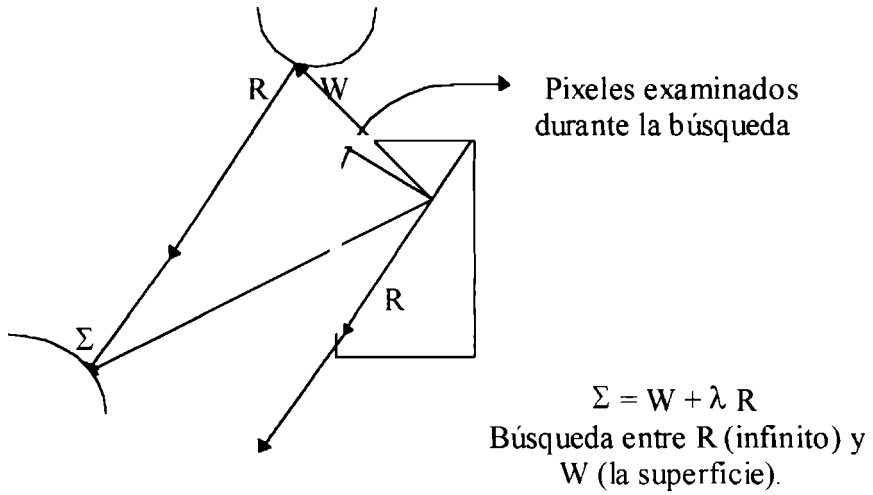


Figura 7.6: Geometría del cálculo de reflexiones

Como puede verse el algoritmo funciona en dos pasadas: primeramente se calcula el mapa de entorno, cargando todos sus A_Buffers con la información correspondiente, incluso con la información de las caras traseras de los objetos para poder obtener la información de las interreflexiones. Recordemos que para obtener el mapa de ambiente hay que renderear las seis caras del cubo que lo forman ubicando el centro de proyección en el lugar donde está la cámara, que se hace girar hacia arriba, abajo y a cada uno de los lados para renderear en las seis caras correspondientes. Luego en la segunda pasada, se renderea sólo la imagen final correspondiente a la cámara seleccionada, que coincide con unas de las caras del mapa de entorno.

Todos estos cálculos son posibles gracias a que la proyección de un segmento del espacio (definido por dos vectores que marcan su inicio y su final) sobre el mapa de entorno forma otro segmento sobre la cara del mapa cúbico. Y como la luz viaja a todos los fines prácticos como una recta (despreciando todo tipo de efectos ondulatorios como difracción e interferencia), podemos encontrar las reflexiones siguiendo la proyección de dichas rectas sin mayor inconveniente, y la recta estaría formada por Objeto Reflectante-Objeto Reflejado. Nuevamente como en el caso del mapa de ambiente Z_Buffereado podemos escribir:

PosiciónDeLaReflexión = [[(1-t)R + tW]] con [[ ]] significando normalización a la unidad.

Como ya hemos visto el parámetro t es un real que pertenece al intervalo [0..1]. Si t = 1 el objeto refleja a otro en contacto con él. Si t = 0, el objeto refleja el infinito (background). Pero si t ≠ 0,1, encontraremos que el objeto refleja a otro indicado por el vector resultante.

En todo momento el algoritmo trabaja en el espacio de precisión de la imagen.

Recordemos que para cada pixel tenemos guardada una lista de superficies ordenada por profundidad para resolver el problema de la reflexión de un "objeto escondido". El algoritmo deberá hacer una búsqueda lineal solamente para los primeros valores de las listas asociadas a cada pixel; en cuyo caso procederá como si estuviera trabajando con una estructura Z_Buffer. Sólo si esta búsqueda falla (no se encuentra el mejor predictor), entonces el segundo valor en cada lista de profundidades de cada pixel es examinado. Aquí es donde se utiliza la estructura A_Buffer. En la implementación todas las búsquedas se harán a la vez; es decir se recuperará el mejor predictor para cada capa de pixeles sobre otra lista, y luego se elegirá entre los elementos de la lista cuál es el mejor predictor. En esta búsqueda podemos descartar totalmente una lista perteneciente a un pixel si la posición del rayo para ese pixel es mayor que la mayor distancia guardada o menor que la menor distancia guardada.

El pseudocódigo asociado al A_Buffer, que coincide con el código implementado en nuestro trabajo es descripto a continuación:

```
Encontrar en el mapa de entorno el mejor aproximador al punto reflejado
                                                    (entre W y R)
Si el aproximador no es suficientemente bueno
   usar valor de fondo
si no
   calcular la contribución según máscara
```

El pseudocódigo es similar al utilizado en el caso del Z_Buffer (ver pág. 69). La diferencia principal reside en cómo calcula la contribución a un pixel Reflectivo. El A_Buffer utiliza una máscara y puede obtener superficies reflejadas que no están visibles desde el punto de observación (ver pág. 46)

7.6. Reflexión sobre proyecciones no isotrópicas

Con respecto a la proyección del mapa de entorno, es necesario aclarar que para el caso de un ángulo de visión de 90°, el mapa de entorno coincide con un cubo, por lo que la proyección en perspectiva de todas las caras introduce la misma deformación en las coordenadas paramétricas ortogonales que definen dichas caras. Este es el caso del algoritmo que implementamos en nuestro trabajo.

En cambio si este ángulo varía, y deseamos que el mapa siga centrado en el observador, debemos diseñar una proyección en perspectiva no uniforme respecto de los ejes X e Y que definen las pantallas de observación para cada una de las caras del mapa, como se muestra en la figura 7.7. La solución más obvia es tratar ambos ejes en forma independiente, aplicando diferentes proyecciones en perspectiva para cada uno.

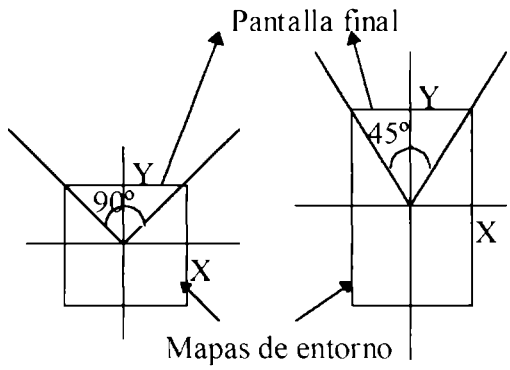


Figura 7.7: Diferentes mapas de entorno según el ángulo de visión

7.7. Otra técnica para el cálculo de reflexiones: Traza de rayos

Un punto crítico en la formación de imágenes es la determinación del color correcto de cada pixel, y una forma de encontrarlo es promediar los colores de los rayos de luz que alcanzan ese pixel [Glassner91].

Nosotros podemos pensar en el rayo de luz como un camino recto seguido por una partícula de luz (llamada fotón) que viaja a través del espacio. En el mundo físico un fotón acarrea energía, y cuando esta energía entra en el ojo es transferida desde el fotón mismo a las celdas receptoras sobre nuestra retina. El color que recibimos está relacionado con esta energía.

Consideremos un pixel particular en el plano de la imagen. ¿Cuáles son los fotones en una escena tridimensional que contribuyen actualmente a ese pixel?

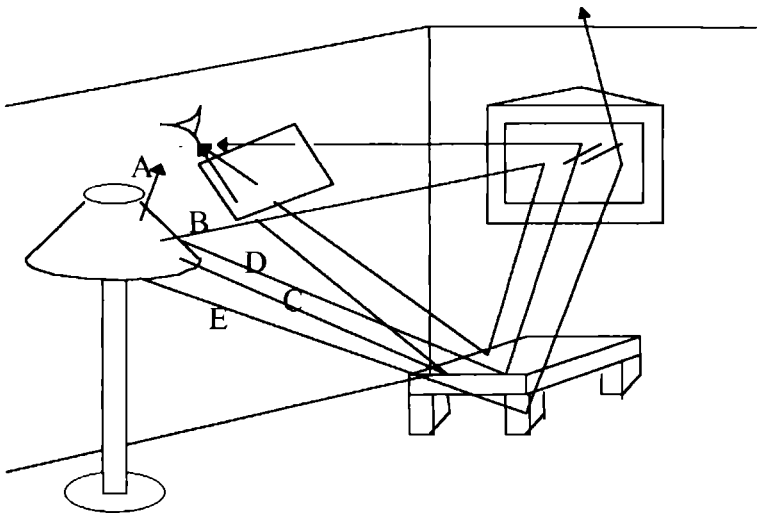


Figura 7.8: Algunos rayos nunca alcanzan el plano de la imagen. Otros siguen rutas simples o complicadas

Los fotones deben comenzar en una fuente de luz. Después de todo, exponiendo un trozo de film en una habitación completamente oscura nada sucede, ninguna luz lo alcanza. Si la lámpara en el cuadro 7.8 está apagada, entonces la habitación está completamente oscura y el cuadro se verá totalmente negro. Así que imaginemos la luz encendida. La lámpara tiene un único foco. El trabajo del foco es crear fotones de todas las frecuencias posibles y enviarlos en todas las direcciones. Para ver como los fotones contribuyen eventualmente a la fotografía sigamos unos pocos fotones en particular.

No consideraremos toda la complejidad de los rebotes de la luz en una escena tridimensional, pero tomaremos los conceptos más importantes.

El fotón A deja la fuente de luz en la dirección de la pared, y choca con ésta. Por ahora supondremos que la luz choca la pared y es absorbida. De este modo el fotón para aquí y no contribuye al cuadro.

El fotón C deja la luz y alcanza la mesa. En la mesa es absorbido en parte antes de ser reflejado. Aún así el fotón (aunque un poco más débil) deja la mesa y pasa a través de la pantalla y entra en el ojo. Este es el motivo por el cual nosotros vemos la mesa: la luz desde la fuente de luz alcanza la mesa y ésta es reflejada a nuestro ojo a través de la pantalla.

La reflexión puede volverse más compleja. El fotón B es reflejado desde el espejo antes de que alcance la mesa. Su camino es la luz, el espejo, la mesa y la pantalla. Alternativamente el fotón D deja la fuente de luz, alcanza la mesa, luego el espejo, para por último ser reflejado en el film. El fotón E sigue un camino similar pero nunca alcanza el film. Otros fotones podrían seguir caminos mucho más complicados en su viaje.

Así, en general, los fotones dejan la fuente de luz y rebotan a lo largo de la escena. Usualmente, la luz se mitiga un poco en cada rebote, así que después de un número de rebotes la luz se mitiga tanto que ya no se puede

ver más. Solamente los fotones que eventualmente alcanzan la pantalla y entran en el ojo (cuando aún están brillantes) contribuyen a la imagen. Notemos que si miramos alrededor nuestro a través de un espejo es probable que veamos objetos que no podemos ver directamente. Los fotones dejan la fuente de luz, alcanzan dicho objeto, luego alcanzan el espejo y por último arriban a nuestro ojo. Esto que hemos visto es el trazado de rayos. Nosotros seguimos (o trazamos) el camino de un fotón (o del rayo de luz) como rebotando a través de la escena. Más específicamente hemos hecho un trazado de rayos hacia adelante, es decir, siguiendo los fotones desde su origen, trazando su camino hacia adelante, tal como los fotones mismos deberían viajar.

7.7.1. Trazado de rayos hacia adelante y hacia atrás

La técnica de trazado de rayos hacia adelante es una primera aproximación al modo en que trabaja el mundo real [Glassner91]. Deberíamos pensar que la simulación de este proceso sería una buena forma de hacer cuadros. Pero hay un problema con esta simulación directa, y es la cantidad de tiempo que llevaría producir una imagen. Consideremos que cada fuente de luz en la escena genera, posiblemente, millones de fotones cada segundo, donde cada fotón está vibrando a una frecuencia levemente diferente, yendo en direcciones levemente diferentes. Muchos de estos fotones alcanzan objetos que nunca veremos, ni siquiera indirectamente. Si tratáramos de crear un cuadro siguiendo fotones desde la fuente, encontraríamos un número despreciablemente pequeños de ellos que alcanzarían la pantalla con una intensidad considerable. Podría tomar años armar un cuadro.

El problema esencial no es que el trazado de rayos hacia adelante no sea bueno, sino que muchos de los fotones que salen de la fuente de luz no juegan un rol importante en la imagen dada. Computacionalmente es muy caro seguir fotones inútiles.

La clave para la eficiencia computacional es revertir el problema, siguiendo los fotones hacia atrás en vez de hacia adelante. Comenzaremos preguntándonos que fotones contribuyen realmente a la imagen. La respuesta es aquellos fotones que alcanzan el plano de la imagen y atraviesan el ojo. Cada uno de estos fotones viajan a lo largo de algún rayo antes de alcanzar la pantalla; algunos pueden venir directamente de la fuente de luz, pero más probablemente rebotarán en otros objetos antes.

Consideremos un punto particular sobre el plano de la imagen. Podemos encontrar fácilmente el camino seguido por un fotón que alcanza el punto sobre la pantalla: es la línea que alcanza nuestro ojo y el punto sobre el film, como lo muestra la figura 7.9. Nosotros sabemos que el camino del fotón es una línea limitada en uno de los finales, pero el fotón podría haber comenzado en cualquier lugar a lo largo de la línea. El término formal para una línea que tiene un punto final fijo es ***rayo***.

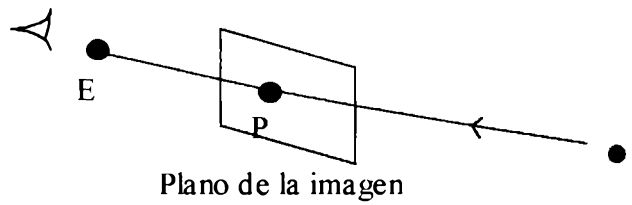


Figura 7.9: Un fotón de luz brillante arriba al ojo pasando a través del punto P en el plano de la imagen. El camino del fotón es a través de una línea recta uniendo E y P.

Entonces, un fotón que contribuye a nuestra visión de la imagen en aquel punto, viene a lo largo de un rayo que une nuestro ojo y el punto sobre el film. Pero ¿qué objeto trae ese fotón? Si extendemos el rayo dentro del mundo, podemos ver al objeto más cercano a lo largo del camino del rayo. El fotón debe haber venido desde ese objeto.

Consideremos el rayo en la figura 7.10, que muestra un rayo de luz uniendo una esfera y el ojo, pasando a través del plano de la imagen. Este es el posible camino del fotón. Nosotros no sabemos si algún fotón alcanza el camino. Pero si algún fotón alcanza aquel trozo de pantalla y entonces el ojo, este debe venir a lo largo de la línea desde la esfera hacia el ojo. Así que la nueva pregunta que debemos hacernos es si algún fotón arriba a lo largo del camino.

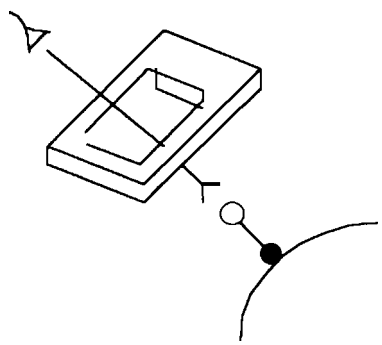


Figura 7.10: un fotón dejando la esfera podría volar a través de un pixel y dentro del ojo

En esta aproximación estamos siguiendo los rayos, no hacia adelante, sino hacia atrás, desde el ojo a los objetos y luego a la fuente de luz. Esta es una observación crítica porque permite restringir nuestra atención a los rayos, que sabemos, serán útiles en nuestra imagen.

Una vez que hemos encontrado el objeto, un fotón debe haberlo dejado para alcanzar nuestro ojo. Debemos encontrar si un fotón realmente viajó por ese camino y si es así ver que color es.

Debido a que el trazado de rayos hacia adelante es demasiado caro, el término "trazado de rayo" se refiere casi exclusivamente al trazado de rayos hacia atrás en computación gráfica.

Recordemos que seguimos un rayo hacia atrás para encontrar donde ha comenzado. Sin embargo, frecuentemente encaramos la búsqueda siguiendo el camino del rayo de luz hacia atrás, imaginándonos a nosotros mismos rodando a lo largo de un camino tomados de un fotón, buscando el primer objeto a lo largo del camino. Este es el objeto en el cual el rayo comienza. Es así que a veces hablamos de buscar el "primer objeto alcanzado por el rayo" o el "primer objeto en el camino del rayo". A lo que nos estamos refiriendo es al objeto que debería haber radiado el fotón que eventualmente viajó a lo largo del rayo.

7.7.2. Traza de Rayos recursiva

El modelo de iluminación desarrollado por Whitted [WHIT80] y Kay [KAY79a] extiende la traza de rayos para incluir la reflexión especular y la transparencia refractiva [Foley96]. La idea general es que la iluminación que cae en una superficie y es enviada, entonces, en nuestra dirección de interés, o bien rebota como una pelota de basketball en un piso duro (reflexión), o bien pasa a través de la superficie arribando al otro lado como un auto conducido a través de un túnel (transparencia).

El algoritmo recursivo de traza de rayos de Whitted genera condicionalmente rayos de reflexión y rayos de refracción desde el punto de intersección como se muestra en la figura 7.11. Los rayos de reflexión y refracción, junto con los rayos de sombra (aquellos trazados desde una superficie hasta la fuente de luz), también se conocen como rayos secundarios para distinguirlos de los rayos primarios que parten del ojo. Si el rayo es especularmente reflectante, un rayo de reflexión se refleja por el normal a la superficie en la dirección de R, que puede calcularse en la manera ya descripta para el modelo de mapa de ambiente(ver pág. 67). Si el modelo es transparente y no ocurre la reflexión interna total, se envía un rayo de refracción al objeto a lo largo de T con un ángulo determinado por la ley de Snell, descripta en la sección anterior (ver pág. 64).

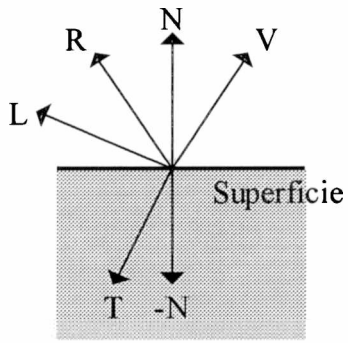


Figura 7.11: Los rayos de reflexión y refracción se generan desde un punto de intersección

A su vez cada uno de estos rayos de refracción y reflexión puede generar recursivamente rayos de refracción, reflexión o sombras, como se presenta en la figura 7.12. Estos rayos forman un árbol de rayos como el de la figura 7.13. En el algoritmo de Whitted [Foley96], una rama termina si los rayos refractados y reflejados no intersecan un objeto, si se ha alcanzado una profundidad máxima establecida por el usuario o si el sistema agota su espacio de almacenamiento.

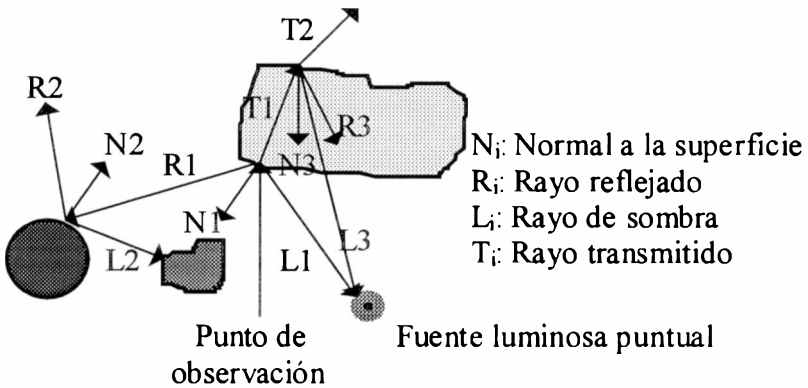


Figura 7.12: Rayos que generan recursivamente otros rayos

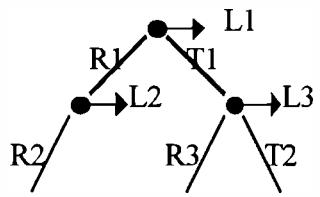


Figura 7.13: Arbol de rayos para la figura 7.12

En la figura 7.12 se presenta un problema básico con la forma en que la traza de rayos modela la refracción: el rayo de sombra L3 no se refracta en su trayectoria hacia la luz. De hecho si simplemente refractáramos L3 de su dirección actual al punto donde sale el objeto grande, no terminaría en la fuente luminosa. Además, al terminar las trayectorias de los rayos refractados se utiliza un sólo índice de refracción para cada rayo.

La traza de rayos es muy susceptible de presentar problemas ocasionados por la limitada precisión numérica. Estos problemas se presentan al calcular objetos que intersecan rayos secundarios. Después de haber calculado las coordenadas *x*, *y*, y *z* del punto de intersección de un objeto visible a un rayo ocular, éstas se usan para definir el punto de partida del rayo secundario cuyo parámetro t debemos determinar. Si el objeto que acabamos de intersecar intercepta el nuevo rayo, con frecuencia tendrá un t muy pequeño, distinto de cero. Si no consideramos esta situación, esta falsa intersección puede generar problemas visuales. Por ejemplo si el rayo fuera de sombra, se consideraría que el objeto bloquea la luz que recibe, produciendo manchas de superficie incorrectamente "autosombreada". Una forma sencilla de resolver este problema es tratar como caso especial el objeto del cual se genera un rayo secundario, para que las pruebas de intersección no se apliquen a él. Es obvio que, esto no funcionará si se utilizan objetos que en realidad podrían oscurecerse a sí mismos o si los rayos transmitidos tiene que pasar por el objeto y reflejarse en su interior. Una solución más general es calcular abs(t) para la intersección, comparar el resultado con un pequeño valor de tolerancia e ignorarlo si está por debajo de la tolerancia.

7.8. Comparación entre Mapa de Ambiente y Trazado de Rayos

El mapa de ambiente es frecuentemente pensado como una forma de obtener efectos de reflexiones aproximados con una fracción de costo computacional. Mientras el trazado de rayos es una técnica incuestionablemente más versátil y comprehensiva, ya que maneja sombras y múltiples niveles de reflexiones y refracciones, el mapa de ambiente es superior en algunos aspectos, por ejemplo en su enorme ventaja en velocidad. El mapa de ambiente convenientemente manejado encuentra iluminaciones difusas, operaciones de antialiasing de iluminación especular que son problemáticas con el trazado de rayos en puntos muestreados en un ambiente.

Teóricamente encontrar la iluminación difusa y especular impregnada sobre una región de la superficie requiere integración sobre parte del ambiente 3D e inherentemente procedimientos complejos. Refinamientos al trazado de rayos propuesto para aproximar esta integración incluyen trazado de rayos distribuido y trazado de rayos con conos. El trazado de rayos distribuido y el de conos se encuentran en el capítulo 16 de [Foley90].

El mapeo de ambiente simplifica el problema tratando el ambiente como una proyección 2D, que permite la integración implementada con filtrado de textura. Especialmente la iluminación de superficies difusas y especulares puede ser encontrada filtrando regiones de un mapa de ambiente. Sin embargo la proyección sobre un ambiente 2D más que el ambiente 3D es una simplificación grosera que sacrifica la calidad de la información local, y consecuentemente aspectos del formado de la imagen que dependen del ambiente local, tales como sombras que no pueden ser implementadas de un

modo confiable. Por otro lado el mapa de ambiente transporta exactamente la iluminación global, y en situaciones donde el ambiente local no afecta substancialmente la superficie mostrada, la calidad subjetiva producida por el mapa de ambiente puede ser superior que aquella producida por el trazado de rayos.

Hay una limitación inherente al mapeo de ambiente. El objeto reflectante no está normalmente en el mapa de ambiente, el objeto no puede reflejar partes de si mismo (por ejemplo las patas no son reflejadas en el cuerpo). Actualmente esta limitación puede ser parcialmente superada utilizando mapas de entorno diferentes para partes del objeto. Con los mapas de entorno A_Buffereados, al guardar información de toda la escena, se resuelven esta limitación porque permiten hacer una implementación recursiva de las reflexiones.

Pero a pesar de todo el mapa de entorno se implementa bien en una escena. Los objetos lejanos son reflejados exactamente. Las reflexiones más próximas son menos exactas, pero la curvatura de la superficie reflectora hace que esto sea difícil de reconocer. Las reflexiones no necesitan ser exactas para verse bien, la atención debería ser puesta en la composición de la escena. Las superficies reflectoras planares, por ejemplo, podrían causar problemas, dado que su distorsión en las reflexiones puede ser bastante notables.

8. Sombras

Podemos aumentar el realismo reproduciendo las sombras proyectadas por un objeto sobre otro. Este es otro caso en que la apariencia de un objeto se ve afectada por otros objetos. Las sombras aumentan el realismo y ofrecen mayor indicación de la profundidad: si el objeto A proyecta una sombra sobre la superficie B, entonces sabemos que A se encuentra entre B y una fuente de luz directa o reflejada.

Los algoritmos de superficies visibles determinan cuales son las superficies que pueden verse desde el punto de observación; los algoritmos de sombras determinan cuales son las superficies que se pueden "ver" desde la fuente luminosa. Por lo tanto ambos algoritmos son en esencia los mismos. Las superficies que son visibles desde la fuente luminosa no están bajo sombra; los que no son visibles desde la fuente están bajo sombra. Cuando hay varias fuentes luminosas, hay que clasificar una superficie con respecto a cada una de ellas.

Consideraremos aquí las fuentes luminosas puntuales, dejando de lado las extendidas. La visibilidad desde una fuente luminosa puntual es, como la visibilidad desde el punto de observación, un aspecto absoluto: todo o nada. Cuando un punto en la superficie no se puede ver desde una fuente luminosa, el cálculo de iluminación debe ajustarse para considerarlo. La adición de sombras a la ecuación de iluminación (ver pág. 57) genera:

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + \Sigma_{1\leq i\leq n}\, S_i f_{atti} I_{p\lambda i}[k_dO_{d\lambda}(N\bullet L) + k_sO_{s\lambda}(R_i\bullet V)^n]$$

donde

$$S_i \begin{cases} 0, \text{ si la luz i está bloqueada en ese punto;} \\ 1, \text{ si la luz i no está bloqueada en ese punto} \end{cases}$$

Las áreas en sombra de todas las fuentes puntuales continúan iluminadas por la luz ambiental.

8.1. Traza de rayos de sombras

Imagínese a usted mismo sobre la superficie de un objeto, como el punto P en la figura 8.1, por ejemplo. ¿Hay alguna luz que llegue a usted desde las fuentes de luz? Una forma de responder a esta pregunta es, simplemente, mirar a cada luz. Si usted puede ver la fuente de luz, entonces hay un camino entre usted y la luz, y como mínimo algunos fotones pueden viajar por ese camino. Si hay algunos objetos opacos en ese tramo, entonces

ninguna luz está viniendo directamente desde la fuente de luz hasta hacia su ojo, y usted está en sombra respecto de esa luz.

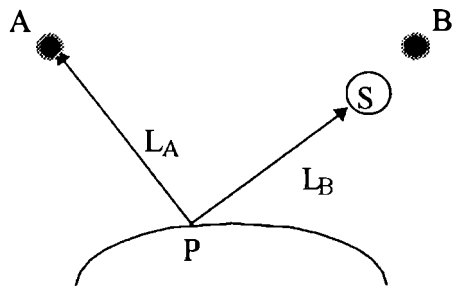


Figura 8.1: determinación de la iluminación de un objeto P por dos fuentes de luz, A y B.

Podemos simular esta operación de pararnos sobre un objeto y mirar hacia la fuente de luz con un rayo de luz llamado rayo de sombras. En la práctica, un rayo de sombra es como cualquier otro rayo, excepto que lo usamos para mirar alrededor buscando sombras. Básicamente comenzamos el trayecto del rayo en el objeto y lo enviamos a la fuente de luz (recuerde que estamos siguiendo el camino de los fotones hacia atrás). Si este rayo retrógrado alcanza la fuente de luz sin chocar ningún objeto en su camino, entonces, ciertamente algunos fotones viajarán hacia adelante por el rayo de luz para iluminar el objeto. Pero si hay algún objeto opaco en el camino, entonces la luz no puede llegar a través de este objeto hacia nosotros; nosotros estaríamos en sombra relativa a esa fuente de luz. La figura 8.1 muestra dos rayos de sombra dejando la superficie, el rayo L_A yendo a la fuente de luz A, y el rayo L_B yendo a la fuente de luz B. El rayo L_A llega a su fuente de luz sin interrupción, pero el rayo L_B choca con un objeto opaco en su camino. De esto deducimos que la luz llegará desde la luz A pero no desde la luz B.

Cuando un rayo de sombra alcanza una fuente de luz sin interrupción, dejamos de pensar en él como tal, y comenzamos a pensarlo como un rayo de luz, que acarrea luz desde la fuente de luz.

En resumen, la primer clase de rayos de luz que contribuyen al color de la luz son los rayos de luz que vienen directamente de la fuente de luz, iluminando al objeto. Para determinar cuando hay algunos fotones viniendo desde una fuente de luz enviamos un rayo de sombra a cada fuente de luz. Si el rayo no encuentra un objeto opaco en su camino hacia la luz, es signo de que los fotones arribarán desde la luz hasta el objeto. En caso contrario el objeto está en sombra relativa a esa fuente de luz.

En este caso solo discutimos que pasa cuando chocamos con objetos opacos; el caso de los objetos Reflectivos es discutido en otra sección del trabajo (ver pág. 77)

8.2. Algoritmo Shadow Buffer

Williams en 1980, desarrolló un método de generación de sombras basado en un algoritmo de dos pasadas a través de un Z_Buffer, una centrada en el centro de proyección y la otra centrada en la fuente de luz. Este algoritmo determina cuando una superficie está sombreada usando cálculos de precisión de imagen. Calcula y almacena, tal como el Z_Buffer para la imagen, desde el punto de vista de la luz. Cuando se determina que un pixel es visible, sus coordenadas de precisión de objeto en el espacio de la pantalla (x_0, y_0, z_0) son trasformadas en coordenadas dentro del espacio de la fuente de luz (x'_0, y'_0, z'_0). Las coordenadas transformadas x'_0 e y'_0 son usadas para seleccionar el valor z_L en el Z_Buffer de la fuente de luz para ser comparado con el valor transformado z'_0. Si z_L es mas cercano a la luz que z'_0, entonces hay algo bloqueando a la luz desde el punto, y el pixel es mostrado como estando en sombra; de otro modo el punto es visible desde la luz y es mostrado como luminoso. En analogía con el mapa de textura, podemos pensar al Z_Buffer de la luz como un mapa de sombra (Shadow Map). Múltiples fuentes de luz pueden ser acomodadas por el uso de mapas de sombra separados para cada fuente de luz.

Este algoritmo requiere que cada pixel rendereado sea procesado. Esto significa que los cálculos de sombras deben ser implementados para el pixel, aún si este es finalmente sobre pintado por objetos mas cercanos.

8.3. Modelo de sombras desde un mapa de ambiente único A_Buffereado.

Este modelo nos permite calcular sombras sin necesidad de usar algoritmos de sombras de Precisión de Objeto, Shadow Buffer o incluso Volúmenes de Sombras. Para conocer detalles de los algoritmos precisión de objetos y volúmenes de sombras consulte el cap. 16 de [Foley90]. Este es el modelo usado en nuestro algoritmo.

El modelo que aquí se propone es el mismo que se describió en la sección donde se habla del cálculo de reflexiones usando un A_Buffer (ver pág. 70). Recordemos que el mismo utiliza un único mapa de entorno A_Buffereado centrado en el observador. El cálculo de sombras puede incorporarse fácilmente al algoritmo descripto para reflexiones si notamos que, dada una superficie visible, esta será iluminada si y sólo si ningún objeto se interpone entre ésta y la luz. Pero toda la información geométrica concerniente a la escena está acumulada en el mapa de entorno, por lo que lo único que debe hacerse es otra búsqueda lineal entre la posición indicada por la proyección de la luz y la posición de la proyección en el mapa (pantalla final) del punto en la superficie a ser sombreado. Esto está resumido en la figura 8.2. Como puede verse, en el caso de las sombras lo único que nos interesa es una función booleana que nos indique si el punto que estamos analizando

está iluminado o no. Es por esto que una vez encontrado un objeto obstructor podemos abandonar la búsqueda con un resultado negativo: el objeto está en sombras.

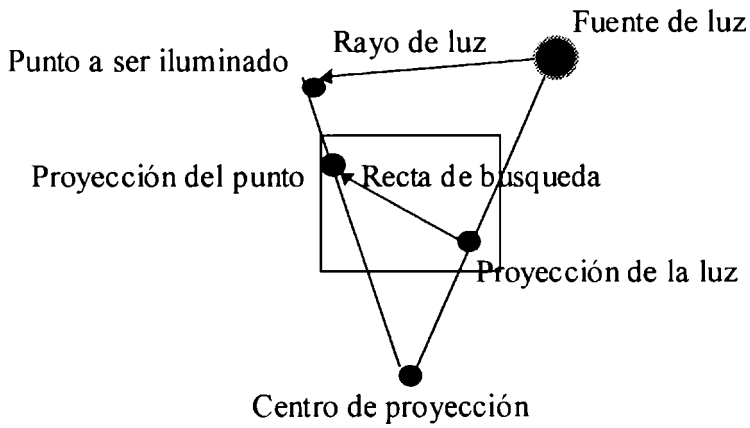


Figura 8.2: Recta de búsqueda para cálculos de iluminación y de sombras

Recordamos que todos estos cálculos son posibles gracias al simple hecho que la proyección de un segmento cualquiera del espacio sobre el mapa de entorno forma otro/s segmento/s sobre la/s cara/s del mapa. Y como la luz viaja a todos los fines prácticos como una recta (despreciando todo tipo de efectos ondulatorios como difracción e interferencia), podemos encontrar las sombras siguiendo la proyección de dichas rectas sin mayor inconveniente, dado que la recta está formada por la luz y el objeto iluminado. Por ello podemos decir una vez más que:

PosiciónDelObjetoInterceptorDeLaLuz = [[(1-t) ***PosiciónDeLaLuz*** + tW]]

El parámetro t es un real que pertenece al intervalo [0..1]. Si t = 1 o si t = 0 el objeto será completamente iluminado, si $t \neq 0, 1$ el objeto ve a la luz bloqueada por otro objeto interpuesto.

9. Análisis del error en Modelos de Reflexión

En esta sección se analizan los errores que se comenten al utilizar los diferentes modelos existentes en Computación Gráfica para calcular reflexiones. Para ello, dos diferentes formas de acotación del error cometido en las respectivas aproximaciones son introducidas, siendo cada una adecuada para determinar el error cometido como una función de la información geométrica disponible, la que estará determinada, obviamente, por el modelo utilizado. En este contexto, los modelos de reflexión de Trazado de Rayos, Mapas de Entorno, Textura y los Mapas de Entorno Z_Bufferados son analizados.

9.1. Introducción

El análisis de los errores cometidos en las aproximaciones realizadas en las diferentes ramas de investigación en Computación Gráfica ha cobrado recientemente una significativa importancia. Esta importancia surge del hecho de que, en esta rama de la ciencia, se intenta realizar una simulación basada en principios físicos del comportamiento de la luz. El objetivo de dicha simulación es modelar la relación existente entre los objetos presentes en una escena dada y la cantidad de luz que alcanza el ojo del observador [Arvo93]. Pero no es necesario modelar hasta el ínfimo detalle el comportamiento de la luz, dado que sólo buscamos predecir con un cierto límite de precisión la apariencia de una escena. Por ello, al imitar la física del comportamiento de la luz, esperamos aproximar el estímulo visual de observar la escena.

Pero al realizar una simulación, nos vemos obligados a realizar una serie de aproximaciones con respecto al comportamiento real de la luz, viéndonos obligados a despreciar otros efectos.

Por ello, el cálculo del error cometido en las aproximaciones realizadas se vuelve sumamente importante para determinar la cantidad de información que se pierde al realizarlas. Una fuerte tendencia en este sentido se ha observado recientemente [Lischinski94], [Arvo94] debido al mayor grado de realismo alcanzado por los actuales algoritmos de iluminación y rendering.

El objetivo de esta sección es realizar el análisis del error en los modelos existentes para el cálculo de reflexiones.

9.1.1. Física de las Reflexiones

Si operamos dentro de los límites de la Optica Geométrica, donde la aproximación por rayos para longitudes de onda muy cortas es válida (siendo

los apartamientos de esta aproximación tan sutiles que, para observarlos, es necesario realizar experimentos cuidadosamente planeados), la formulación de las leyes de reflexión y refracción pueden formularse en términos extremadamente sencillos como hemos visto en una de las secciones previas (ver pág. 64). De hecho, el nombre de esta rama de la Óptica deriva del hecho de que, al tomar el caso límite Longitud_De_Onda $\rightarrow 0$ (longitudes de onda despreciables), se puede formular la teoría en términos puramente geométricos [Born64] Recordemos las siguientes leyes experimentales :

Las direcciones de Incidencia, **I**, Refracción, **T**, y Reflexión, **R**, están en un mismo plano, que es normal a la superficie de separación y por lo tanto contiene al normal **n** a la superficie.

El ángulo de incidencia es igual al ángulo de reflexión, es decir, $\theta_I = \theta_R$.

Estas leyes son válidas aún cuando la superficie de la onda y la superficie de separación no sean planas, porque en cada punto hay una sección limitada por ambas superficies que se puede considerar plana y los rayos en ese punto se comportan en conformidad a las leyes hasta aquí presentadas.

9.1.2. Relaciones Geométricas de la Reflexión

En lo que sigue de nuestro análisis supondremos que los vectores **n**, **I** y **R** están normalizados a la unidad (o sea: $\| \mathbf{n} \| = \| \mathbf{I} \| = \| \mathbf{R} \| = 1$). Esto no representa ninguna pérdida de generalidad en nuestros cálculos debido a que sólo nos interesan sus direcciones y no sus magnitudes.

Si llamamos **W** al vector que señala la superficie de separación (reflectora), calculado desde el origen de coordenadas, podemos escribir (ver pág. 67)

$$\Sigma_{\text{reflexión}} = \mathbf{W} + \lambda \mathbf{R} \qquad (1)$$

donde $\Sigma_{\text{reflexión}}$ es el vector que señala la superficie reflejada por el rayo **R** a partir del punto señalado por el vector **W**, medido desde el origen; y $\lambda \in \Re$, $\lambda \geq 0$ (que, obviamente, desconocemos a priori).

De esta forma, podemos ver que, por construcción, los vectores **W**, **R** y $\Sigma_{\text{reflexión}}$ pertenecen al mismo plano (Dos vectores --y el origen-- definen un plano al que pertenecen, y cualquier combinación lineal de ellos también pertenece a dicho plano). Notar que este plano no necesariamente contiene a los vectores **I** y **n**.

Recordemos también el teorema 1 (ver pág. 67)

$$[[\Sigma_{\text{reflexión}}]] = [[\ (1-t_0)\ \mathbf{R} + t_0\ \mathbf{W}\ ]] \qquad (2)$$

con $t_0 \in [0,1]$.

La interpretación geométrica de estos resultados está de acuerdo con la intuición: cuando $t_0 = 1$ (λ=0), $[[\Sigma_{reflexión}\ (t_0=1)]] = [[\ \mathbf{W}\ ]]$ y el objeto reflejado por nuestro espejo se encontrará en contacto con la superficie reflectora, mientras que si $t_0 = 0$ ($\lambda \rightarrow \infty$), $[[\Sigma_{reflexión}\ (t_0=0)]] = \mathbf{R}$ y podemos decir que el objeto reflejado se encuentra en el infinito.

9.2. Análisis del error desde el centro de proyección

En esta sección presentaremos el primero de nuestras cotas del error, que nos permitirá analizar los modelos más utilizados en Computación Gráfica para calcular reflexiones, abriéndonos las puertas para desarrollar un nuevo modelo con mayor precisión.

Por definición, el error se obtiene de comparar el vector buscado ($\Sigma_{reflexión}$) con el vector unitario P´ utilizado para calcular las reflexiones en los diferentes modelos vistos. Una expresión posible para el error es utilizar el coseno del ángulo existente entre el vector utilizado (medido desde el Centro de Proyección, CP) y $\Sigma_{reflexión}$:

$$Error_{CP} = (1 - \mathbf{P}' \bullet [[\ \Sigma_{reflexión}\ ]]) / 2 \qquad (3)$$

dónde hemos supuesto que el vector utilizado para estimar la reflexión, P´, está normalizado a la unidad y '•' representa el producto escalar entre dos vectores. El nombre de este error proviene de que utilizamos al Centro de Proyección (CP) como origen de los vectores utilizados.

Con esta definición del error, podemos ver que éste puede tomar valores entre 0 y 1, significando 0 que no se comete error alguno ($P' \bullet [[\ \Sigma_{reflexión}\ ]] = 1$, es decir que P´ y $[[\ \Sigma_{reflexión}\ ]]$ son paralelos y, por lo tanto, $P' = [[\ \Sigma_{reflexión}\ ]]$) y 1 que nuestra aproximación está dándonos la dirección opuesta a la buscada ($P' = -[[\ \Sigma_{reflexión}\ ]]$).

Como es inmediatamente obvio, este error sólo tiene en cuenta el error en la elección del ángulo entre ambos vectores, ignorando completamente la posibilidad de error en la distancia al Centro de Proyección. Sin embargo, es el único error que podemos definir si carecemos de esta información, como ocurre con los modelos basados en el uso de mapas de entorno tradicionales [Greene86], como más adelante veremos al analizarlos en detalle.

En lo que resta de esta sección analizaremos algunos modelos de reflexión comúnmente usados, buscando luego, al final de la misma, un nuevo modelo que nos asegure la mayor precisión posible, es decir, aquella que nos de una cota mínima para el error.

9.2.1. Trazado de Rayos (Ray Tracing) recursivo

Como ya hemos mencionado (ver pág. 77) el modelo de Trazado de Rayos (Ray Tracing) consiste en simular el comportamiento de la luz utilizando las leyes de la Optica Geométrica. El costo temporal de computar qué color corresponde a cada pixel de una imagen utilizando un algoritmo de trazado de rayos desde las fuentes (Light Ray Tracing [Arvo86]) sería excesivamente costoso [Glassner91]. La idea, entonces, es invertir el problema, preguntándonos cuáles son los fotones que con seguridad contribuyen a la imagen. Este algoritmo se denomina Trazado de Rayos Hacia Atrás (Backwards Ray Tracing) o, simplemente, Trazado de Rayos.

Cada vez un que un rayo intercepta con una superficie (W), el rayo reflejado (R) es calculado a partir del incidente (I) y una llamada recursiva repite el proceso con este nuevo rayo y con W como origen, hasta que se alcanza un cierto nivel límite dentro de la recursión o hasta que una superficie difusa (sin componente especular) es alcanzada, lo que nos da el valor exacto para λ al hallar $\Sigma_{\text{reflexión}}$ en forma exacta.

Este modelo realiza un seguimiento de los rayos que parten del ojo hacia toda la escena, simulando de esta manera, para cada uno, en el caso de las reflexiones, el desarrollo exacto de la Ecuación (1). Por lo tanto, cualquier análisis del error aplicado a este modelo nos conducirá al error nulo, es decir, que el modelo de trazado de rayos calcula las reflexiones exactas.

A pesar de ello, la utilización de este modelo para calcular imágenes de gran realismo es comúnmente evitado por tener un altísimo costo computacional. El proceso de calcular la intersección de un rayo con un objeto requiere calcular su intersección con todos los objetos presentes en la escena y retener sólo el más cercano. Incluso con las técnicas de optimización más eficientes un gran número de computaciones innecesarias son realizadas.

9.2.2. Modelos Basados en el uso de Mapas de Entorno

La utilización de Mapas de Entorno (Environment Maps) para calcular reflexiones fue introducida por Blinn y Newell en el año 1976 [Blinn76]. Su utilidad es especialmente significativa en escenas complejas, donde utilizar el Trazado de Rayos es prohibitivamente costoso, aunque el costo ahora se encuentra en la precisión de las aproximaciones hechas.

Generalmente, estos modelos se utilizan dentro del marco de los algoritmos Scan-Line (ver pág. 36) en los que el proceso es nuevamente invertido, pero sólo para un primer nivel (primer llamado recursivo en un algoritmo de ray tracing) y sólo para aquellos objetos que se proyecten a la pantalla. El eje del método es que la mayor parte de los cálculos son descompuestos en problemas de interpolación más simples: los algoritmos operan sobre un polígono por vez, dividiéndolo en segmentos horizontales respecto de la pantalla (scan lines), los que a su vez son divididos según los

pixeles que cubren, calculándose para cada uno los respectivos valores de iluminación (modelos de Gouraud, Phong, etc.).

La técnica de uso de un Mapa de Entorno se basa en la suposición crucial de que el objeto reflector es pequeño comparado con las dimensiones de la escena que lo rodea, lo que permite modelar dicho entorno como una proyección bidimensional rodeando el objeto que nos interesa. Podemos, entonces, rodear al objeto reflector por una superficie tridimensional cerrada, generalmente una esfera o un cubo, sobre la cual es proyectado el entorno del objeto (respecto del Centro de Proyección, CP, mencionado anteriormente). Podemos decir que el mapa de entorno no es más que la imagen de la escena vista desde el CP, que generalmente es elegido en el centro geométrico del objeto a ser mapeado. En términos más formales, podemos decir que un mapa de entorno es un conjunto de pares {(V, RGB)}, dónde V es un vector unitario desde el centro de proyección (independientemente de que realización se halla elegido para el mapa), y RGB es el valor observado (intensidad lumínica, en general, en tres frecuencias características: Rojo, Verde y Azul) desde allí en dirección V. En el proceso final de generación de la imagen, cuando se observa el objeto espejado, las reflexiones son calculadas, obteniéndose el vector R a partir de I y n. En referencia a la Ecuación (1), podemos decir que conocemos los valores de R y de W, pero no de λ, por lo que calcular la reflexión exacta utilizando sólo el mapa de entorno es virtualmente imposible (ver Figura 9.1). Entonces, la aproximación consistirá en elegir, a partir de la información que poseemos, el mejor V posible, es decir, la mejor forma de indexar el mapa, sin calcular el valor exacto de λ ni de $\Sigma_{reflexión}$

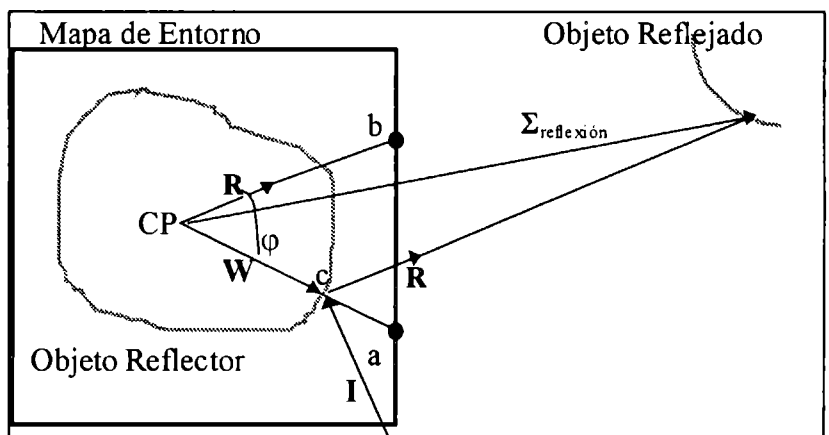


Figura 9.1: utilización del mapa de entorno

9.2.2.1. Uso del mapa como textura.

Una posible elección para elegir el vector que indexará el mapa es tomar $t_0 = 1$ en la Ecuación (2), es decir, usar $[[\ \Sigma_{reflexión}\ ]] \approx [[\ \mathbf{W}\ ]]$, obteniendo como resultado el valor correspondiente al punto 'a' en la figura 9.1. Como se comprende fácilmente, esta elección del indexador no tiene en cuenta al vector reflejado, **R**, dependiendo sólo de la posición en el objeto donde ocurre la reflexión. Por lo tanto, indexar el mapa de esta forma es totalmente equivalente a utilizarlo como una textura, siendo éste totalmente independiente del

observador. Las reflexiones logradas de esta forma no guardan ningún realismo, pero el "modelo" ha sido incluido aquí solamente con fines ilustrativos.

La expresión (3) para el error en esta elección particular de $\mathbf{P}' \equiv [[\ \mathbf{W}\ ]]$, es

$$Error_{CP} = (1 - [[\ \mathbf{W}\ ]] \bullet [[\ \Sigma_{reflexión}\ ]]\ ) / 2$$

Si reemplazamos en esta expresión el valor de $[[\ \Sigma_{reflexión}\ ]]$ dado por la Ecuación (2), notamos que en realidad $Error_{CP}$ es un función siempre positiva con parámetro t_0. Como tal, estamos interesados en hallar su valor máximo, es decir, el valor de t_0 que maximiza la expresión anterior (para poder determinar cual es el peor error que podemos cometer utilizando este modelo). Derivando respecto de t_0 e igualando a cero obtenemos que el valor de t_0 que extremiza la función es $t_0 = 1$, es decir que $[[\ \Sigma_{reflexión}\ (t_0=1)\ ]] = [[\ \mathbf{W}\ ]]$ y $Error_{CP} = 0$, por lo que hemos hallado un mínimo absoluto. Por lo tanto, el valor máximo del error es alcanzado en el otro extremo del intervalo [0,1], para $t_0 = 0$ ($[[\ \Sigma_{reflexión}\ (t_0=1)]] = \mathbf{R}$) y el error será

$$Error_{CP}^{Max} = (1 - \mathbf{R} \bullet [[\mathbf{W}]]) / 2 = (1 - Cos\ (\varphi)\ ) / 2$$

dónde φ es el ángulo entre **W** y **R**, como se ve en la figura 9.1. En la figura 9.2 se encuentra el gráfico de $Error_{CP}$ para todos los posibles valores de t_0 ($\in [0,1]$) y de φ ($\in [0,\pi]$). Como puede observarse en el gráfico, el máximo valor para el parámetro t_0 se alcanza en 0. De esta expresión, se comprende que el error máximo cometido al hacer esta aproximación depende del valor del ángulo φ, que puede determinarse fácilmente una vez conocidos los vectores involucrados en la reflexión. Vemos un par de ejemplos: como puede verse fácilmente de la última expresión, si $\varphi = 0$ tendremos que $Error_{CP}^{Max} = 0$, que es un resultado que concuerda con la intuición porque ambos vectores apuntan en la misma dirección, por lo que la dirección real no puede ser otra que alguna de las dos. En cambio, si $\varphi = \pi$, $Error_{CP}^{Max} = 1$, lo que significa que en realidad no podemos tener ninguna confianza en el resultado obtenido: **R** y **W** son vectores que apuntan en direcciones opuestas, y el vector real puede encontrarse apuntando en cualquier dirección con un ángulo respecto de **W** entre 0 y π.

A priori, sin más información que esta, no podemos hacer una mejor acotación del error.

9.2.2.2. El modelo de Blinn y Newell

Cuando introdujeron su trabajo original, Blinn y Newell propusieron indexar el mapa solamente usando el vector **R**, lo que implica tomar $t_0 = 0$ en la Ecuación (2) y, por lo tanto, suponer que $[[\ \Sigma_{reflexión}\ ]] \approx \mathbf{R}$. Esta aproximación implica tomar $\lambda \rightarrow \infty$ en (1), es decir que suponemos las dimensiones del objeto despreciables con respecto a las de su entorno [Foley90]. El punto del

mapa elegido de esta manera está señalado con 'b' en la figura 9.1, siendo ésta la manera de indexar al mapa más utilizada por la vasta mayoría de los sistemas de rendering existentes hoy en día [Upstill92], [Voorhies94], principalmente debido a su sencillez y gran realismo visual.

Reemplazando **P´** por **R** en (3) obtenemos:

$$Error_{CP} = (1 - [[\mathbf{R}]] \bullet [[\Sigma_{reflexión}]]) / 2$$

Que tiene un mínimo absoluto para $t_0 = 0$ cuyo valor es $Error_{CP} = 0$ (que se obtiene derivando esta expresión respecto de t_0) y alcanza su máximo valor en el otro extremo del intervalo, $t_0 = 1$, siendo entonces

$$Error_{CP}^{Max} = (1 - \mathbf{R} \bullet [[\mathbf{W}]]) / 2 = (1 - \cos(\varphi)) / 2$$

Que es exactamente la misma cota del error hallada para la aproximación anterior. $Error_{CP}$ se encuentra graficada, para los diferentes valores del ángulo y del parámetro t_0, en la Figura 9.3

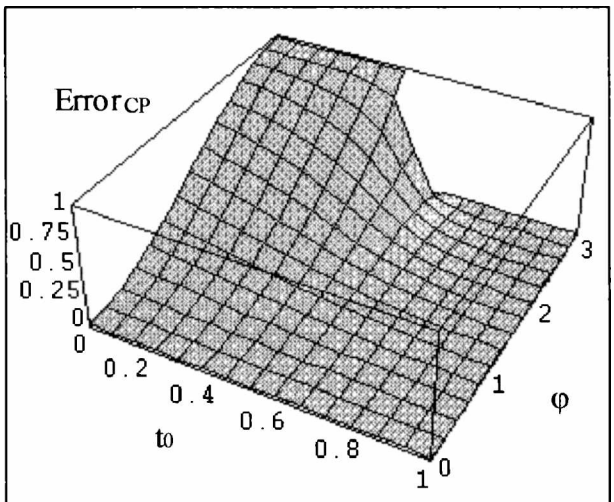


Figura 9.2: Gráfico de $Error_{CP}$ (t_0, t=1, φ)

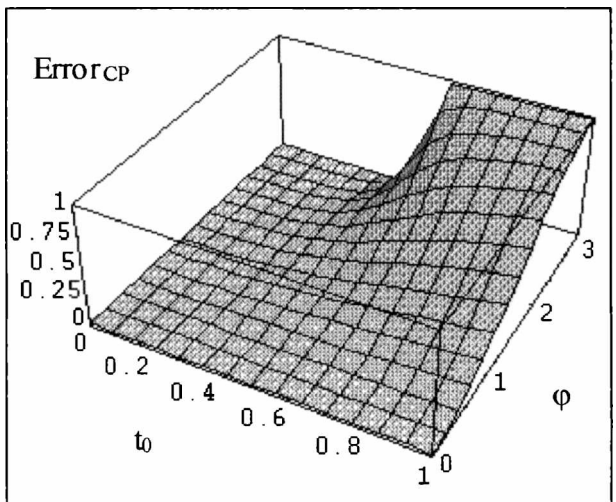


Figura: 9.3: Gráfico de $Error_{CP}$ (t_0, t = 0, φ)

Hasta este momento hemos discutido dos aproximaciones para la reflexión, siendo una de ellas la elegida por prácticamente la universalidad de los paquetes de rendering. Pero... ¿Es ésta la mejor aproximación posible? En la siguiente sección buscaremos el vector que nos asegure el menor error posible, respondiendo de esta manera a la pregunta formulada.

9.2.3. Búsqueda del mejor aproximador para modelos basados en Mapas de Entorno.

Hasta ahora sólo hemos propuesto un par de valores posibles para el aproximador **P'**, habiendo analizado en detalle cada uno. Pero aún queda pendiente la cuestión de si existe un mejor aproximador y, de ser así, cuál es.

Para ello, recordemos que, a partir de la expresión (2), el aproximador también puede ser escrito como

$$\mathbf{P'} = [[\ (1-t)\ \mathbf{R} + t\ \mathbf{W}\ ]] \qquad (4)$$

con $t \in [0,1]$. Reemplazando los posibles valores para $[[\ \Sigma_{\text{reflexión}}\ ]]$ y para **P'** en la definición del error desde el Centro de Proyección, tenemos

$$\text{Error}_{CP} = (\ 1 - [[\ (1-t)\ \mathbf{R} + t\ \mathbf{W}\ ]] \bullet [[\ (1-t_0)\ \mathbf{R} + t_0\ \mathbf{W}\ ]]\ ) / 2 \qquad (5)$$

Como vemos, la expresión más general para el Error_{CP} depende de tres parámetros: t, t_0 y φ, el ángulo entre **R** y **W**.

En lo que sigue, con el fin de simplificar los cálculos, supondremos que $||\ \mathbf{W}\ || = W = 1$, dejando para el final de esta sección la explicación de cómo generalizar los resultados obtenidos para el caso más general.

Armados con estas expresiones, estamos en condiciones de formular el siguiente teorema:

Teorema 2:

El máximo error posible para cualquier elección del parámetro t en (5) es mayor que el máximo error posible cuando elegimos t = ½. Más formalmente:

$$\forall t \in [0,1] \quad \underset{t_0 \in [0,1]}{Supremo}\left\{Error_{CP}(t_0,t)\right\} \geq \underset{t_0 \in [0,1]}{Supremo}\left\{Error_{CP}(t_0,1/2)\right\}$$

Que se puede demostrar como sigue:

La expresión (5) puede ser diferenciada respecto de t_0 e igualada a 0, hallándose, luego de un largo pero sencillo cálculo algebraico, un valor que representa un mínimo (absoluto) del error. En efecto, el valor del error para esta elección particular de t_0 es 0, lo que nos indica que el máximo con respecto de t_0 de esta función será alcanzado en los bordes. Esto significa

que, para un t dado, el máximo error posible será el mayor valor entre los valores que tome en los extremos del intervalo [0,1]. Más formalmente:

$$\underset{t_0 \in [0,1]}{Supremo}\left\{Error_{CP}(t_0,t)\right\} = Max\left\{Error_{CP}(0,t), Error_{CP}(1,t)\right\}$$

Obviamente, para cualquier valor que tomemos de t, el error en $t_0 = 0,1$ será mayor que su mínimo, para todo t, evaluando también t_0 en 0 ó 1:

$$Max\left\{Error_{CP}(0,t), Error_{CP}(1,t)\right\} \geq \underset{t \in [0,1]}{Infimo}\left\{Max\left\{Error_{CP}(0,t), Error_{CP}(1,t)\right\}\right\}$$

Podemos analizar esta expresión analíticamente, pero aquí tomaremos un camino gráfico mucho más sencillo y de mayor claridad conceptual. Los gráficos de las Figuras 4 y 5 son, respectivamente, los gráficos del error para las elecciones de $t_0 = 1,0$. Si graficamos la $Max\{Error_{CP}(0,t), Error_{CP}(1,t)\}$, obtendremos la figura 9.4, de dónde hallamos fácilmente que el ínfimo se logra para t = ½, es decir

$$\underset{t \in [0,1]}{Infimo}\left\{Max\left\{Error_{CP}(0,t), Error_{CP}(1,t)\right\}\right\} = Max\left\{Error_{CP}(0,t), Error_{CP}(1,t)\right\}$$
$$= \underset{t_0 \in [0,1]}{Supremo}\left\{Error(t_0, 1/2)\right\}$$

Con lo que el teorema queda demostrado.

El valor que toma el vector **P´** para t = ½ es (reemplazando en (4)):

$$\mathbf{P'} = [[\mathbf{R} + \mathbf{W}]]$$

Como vemos, el vector que nos garantiza cometer el error menos grosero es aquel que representa el medio geométrico entre los vectores **R** y **W** (este resultado es válido en general, como enseguida veremos). El gráfico para el error cometido usando este aproximador se encuentra en la figura 9.5, donde podemos notar que el máximo valor que toma la función en el dominio es ½.

Si buscamos generalizar el teorema para $||\mathbf{W}|| = W \neq 1$, sólo tenemos que repetir los cálculos, pero esta vez el ínfimo se encontrará en t = 1/(1+W). Este valor nos conduce a la expresión más general posible para el vector **P´** que nos asegura cometer el menor error posible dentro de los límites de las aproximaciones usadas.

Este valor de **P´** es:

$$\mathbf{P'} = [[\mathbf{R} + [[\mathbf{W}]]]] \qquad (6)$$

Por supuesto que, para el caso $||\mathbf{R} + [[\mathbf{W}]]|| = 0$ en que la expresión anterior no se puede evaluar porque $\mathbf{R} = -[[\mathbf{W}]]$, debemos tomar $\mathbf{P'} = \mathbf{R}$. De la observación de la expresión (6) vemos que **P´** es exactamente el vector que

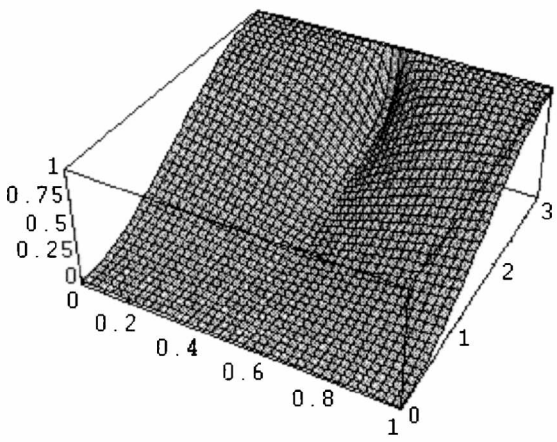


Figura 9.4: gráfico de Max{Error $_{CP}$ (0,t), Error$_{CP}$ (1,t)}

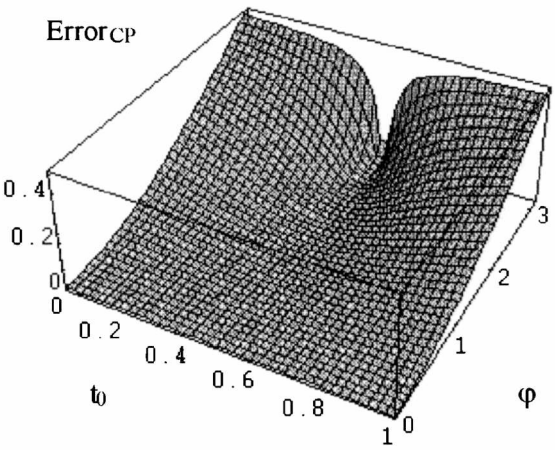


Figura 9.5: Gráfico de Error $_{CP}$ (t_0, φ) para el **P**′ óptimo (cuando usamos t = ½).

representa el medio geométrico entre las direcciones indicadas por **R** y **W**, como anteriormente señalamos.

9.3. El Error de Paralaje

En la sección precedente presentamos el error medido desde el Centro de Proyección. Este es el error utilizado si no poseemos otra información que la distribución angular por medio de, como ya mencionamos, los pares (V, RGB). En esta sección haremos el análisis para el caso en que dispongamos de más información, en particular si disponemos de ternas de la forma (V, α_v , RGB), donde αv es la distancia a la superficie más próxima señalada por V, medida desde el Centro de Proyección, (como ejemplo podemos escribir $\Sigma_{reflexión} = \alpha_{reflexión}$ [[$\Sigma_{reflexión}$]], de dónde, para este caso particular, $\alpha_{reflexión} \equiv$ || $\Sigma_{reflexión}$ ||).

Para ello, es más conveniente introducir una nueva estimación para el error para las reflexiones, que consiste en acotar la desviación que cometemos con respecto al rayo reflejado, pero medido desde la superficie reflectora. Definimos:

$$Error_{Paralaje} = (1 - [[\mathbf{P}'']] \bullet [[\ \Sigma_{reflexión}']]) / 2 \qquad (7)$$

donde son utilizados ahora los vectores primados indicando que son medidos a partir de **W**, es decir, que realizamos el cambio de coordenadas **X'** = **X** – **W**. De la figura 9.1, vemos que, ante este cambio de coordenadas que nos lleva el origen al punto rotulado como 'c', tenemos [[$\Sigma_{reflexión}$']] ≡ **R**. Una vez más, el error fue definido de forma tal que nos de 0 para los valores exactos y 1 para el error máximo alcanzable, cuando no se tiene ninguna seguridad con respecto a la aproximación hecha.

En esta sección revisaremos las aproximaciones ya analizadas en la sección anterior, pero utilizando el nuevo error introducido. Para ello supondremos que disponemos de la información de distancias (α). Finalmente, analizaremos los Mapas de Entorno Z_Buffereados [Patow95], determinando una cota superior para el error cometido y demostrando que, efectivamente, representan una notable mejora en la precisión alcanzada.

9.3.1. Error de Paralaje para el uso del mapa como textura

Para poder calcular el $Error_{Paralaje}$ para cuando aproximamos $[[\Sigma_{reflexión}]] \approx [[\mathbf{W}]]$, debemos primeramente hallar el valor de $[[\mathbf{W'}]]$:

$$[[\mathbf{W'}]] = [[\alpha_W \mathbf{W} - \mathbf{W}]] = [[(\alpha_W - 1) \mathbf{W}]] = \begin{cases} -[[\mathbf{W}]] & 0 \le \alpha_W < 1 \\ +[[\mathbf{W}]] & \alpha_W \ge 1 \end{cases}$$

por lo que, para $Error_{Paralaje}$ obtendremos

$$Error_{Paralaje}(\varphi, W, \alpha_W) = \begin{cases} \dfrac{1+Cos(\varphi)}{2} & 0 \le \alpha_W < 1 \\ \dfrac{1-Cos(\varphi)}{2} & \alpha_W \ge 1 \end{cases}$$

que, como podemos ver, toma valores entre 0 y 1 en función del ángulo entre **R** y **W**, tanto para valores de $0 \le \alpha_W < 1$ como para valores $\alpha_W \ge 1$. Gracias a éste análisis con el error, vemos confirmada nuestra apreciación anterior sobre la incerteza que representa tomar esta aproximación en particular.

9.3.2. Error de Paralaje en el modelo de Blinn y Newell

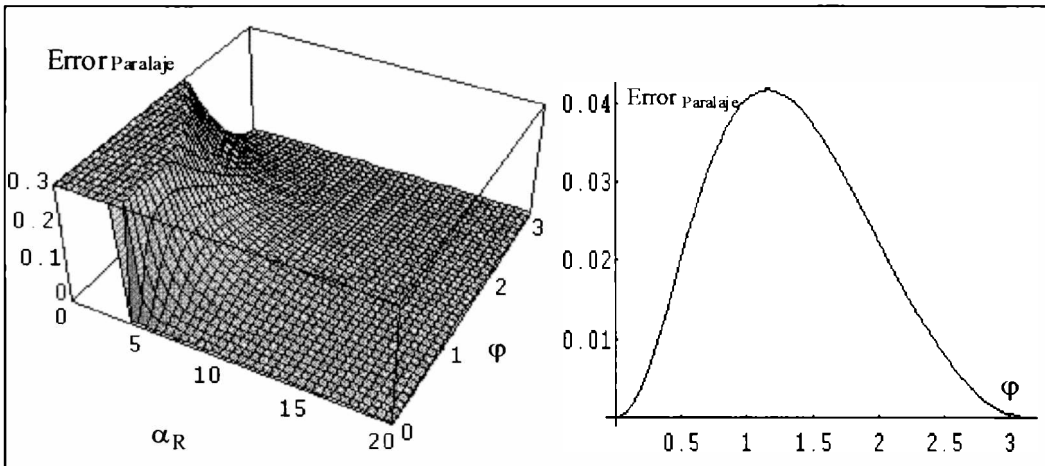
Si bien el análisis del primer modelo no aportó demasiado a nuestro conocimiento, el análisis del modelo original de Blinn y Newell mostrará que, a medida que se incrementa la distancia al Centro de Proyección, su precisión va aumentando continuamente o, lo que es lo mismo, su error decrece monótonamente.

Para ello, recordemos que $\mathbf{R'} = [[\alpha_R \mathbf{R} - \mathbf{W}]]$, por lo que el error puede ser escrito, luego de ser completamente desarrollado, como

$$Error_{Paralaje}(\alpha_R, W, \varphi) = \frac{1 - \dfrac{\alpha_R - W\cos(\varphi)}{(\alpha_R - 2\alpha_R W\cos(\varphi) + W^2)^{1/2}}}{2}$$

El gráfico de esta función (para una elección particular del parámetro W) se encuentra en la figura 9.6a). Como puede demostrarse fácilmente, para $\alpha_R \to \infty$ tenemos que $Error_{Paralaje} \to 0$, lo cual verifica nuevamente que ésta es

una aproximación excelente para objetos infinitamente alejados, o de dimensiones pequeñas respecto del reflector. Asimismo, como se desprende de la figura, la función de error presenta una notoria discontinuidad para $\varphi = 0$, y, para todos los valores de $\alpha_R < W$, vemos que el error es muy cercano a 1. En realidad, no debemos preocuparnos demasiado por este fenómeno, dado que, si elegimos el Centro de Proyección en el centro geométrico del objeto reflector, difícilmente nos encontremos en esta situación, que implicaría que el objeto reflejado estaría entre la superficie del reflector y su centro geométrico (podemos decir que en este caso, el objeto reflejado estará *dentro* del reflector), que representa una situación físicamente poco probable. Lo que sí debería llamar nuestra atención, es que el error para distancias cercanos o intermedios con respecto a la superficie reflectora tiene un error no despreciable que depende fuertemente del ángulo entre **R** y **W**, como se ve fácilmente en la figura 9.6b).



a) Comportamiento general de la función

b) Gráfico de $Error_{Paralaje}$ (α_R =10, W=5, φ)

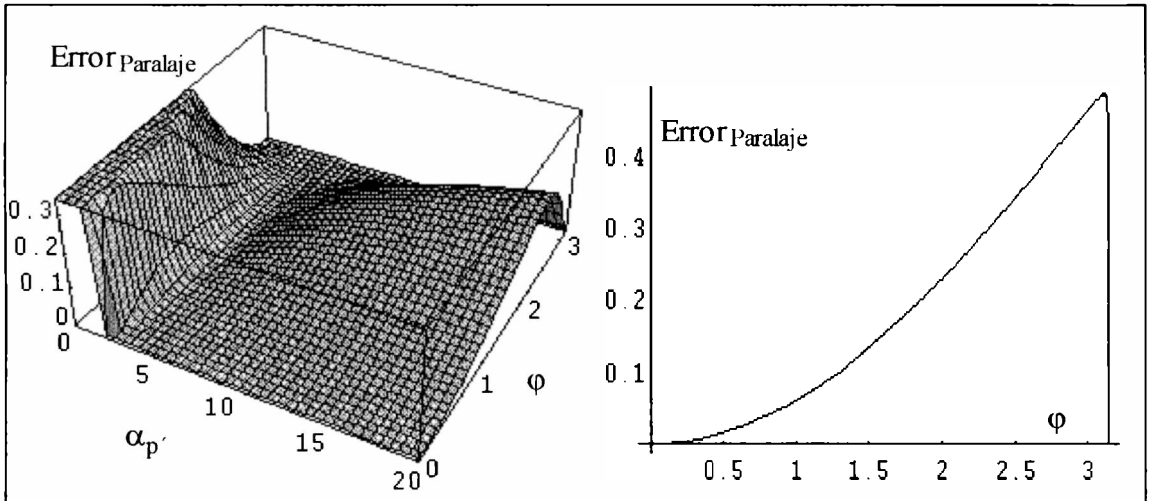
Figura 9.6: gráfico de $Error_{Paralaje}(\alpha_R, W{=}5, \varphi)$ para el Modelo de Blinn y Newell. En este gráfico, W fue elegido igual a 5 puramente con fines ilustrativos

9.3.3. Error de Paralaje del óptimo para el $Error_{CP}$

Podemos aplicar estos mismos análisis para el vector **P´** hallado en la Sección 9.2.3 (ver pág. 92) que representa la elección que nos optimiza $Error_{CP}$. Procediendo como antes, el vector **P´´** en el nuevo sistema de referencia será:

$$P'' = \frac{\alpha_{P'} R + (\alpha_{P'} - W)\,[[\,W\,]]}{(\alpha_{P'}^{\,2} + (\alpha_{P'} - W)^{\,2} + 2\alpha_{P'}(\alpha_{P'} - W)\cos(\varphi))^{1/2}}$$

siendo $\alpha_{P'}$ la distancia desde el CP a la superficie más cercana en la dirección **P´**. Esta expresión nos conducirá, para el error de paralaje, a:



a) Comportamiento General de la Función

b) Gráfico de $Error_{Paralaje}(\alpha_{P'}=10000, W=4, \varphi)$

Figura 9.7: gráfico de $Error_{Paralaje}(\alpha_{P'}, W=5, \varphi)$ para el indexador óptimo segúnel $Error_{P}$. En este gráfico, W fue elegido igual a 4 puramente con fines ilustrativos

$$Error_{Paralaje} = \frac{1 - \dfrac{\alpha_{P'} + (\alpha_{P'} - W)\cos(\varphi)}{(\alpha_{P'}^2 + (\alpha_{P'} - W)^2 + 2\alpha_{P'}(\alpha_{P'} - W)\cos(\varphi))^{1/2}}}{2}$$

El correspondiente gráfico se encuentra en la figura 9.7.a), siendo 9.7.b) el error para un valor grande arbitrariamente elegido de $\alpha_{P'}$ (nuevamente, con fines ilustrativos, se eligió $W = 4$ y el valor para $\alpha_{P'}$ fue elegido en 10000). Como puede verse fácilmente, el error ahora vale idénticamente cero para todos los valores de φ sólo cuando $\alpha_{P'} = W$, creciendo monótonamente para valores mayores de $\alpha_{P'}$ y está acotada en ½ para $\alpha_{P'} \to \infty$ (alcanzado en $\varphi = \pi$).

De todo este análisis concluimos que, si disponemos de la información de distancias al CP, dependiendo de los valores relativos de los diferentes α involucrados, puede llegar a ser más conveniente utilizar un aproximador u otro. Para $\alpha_{P'}$ grande, **R** es la mejor elección, pero si $\alpha_{P'} \approx W$, entonces elegir [[**R** + [[**W**]]]] es mucho mejor.

Pero, en el caso de que contemos con esa información, es mejor cambiar el modelo y utilizar un Mapa de Entorno Z_Buffereado, como discutiremos en la siguiente sección.

9.3.4.3.4 Error de Paralaje para los Mapas de Entorno Z_Buffereados

El Mapa de Entorno Z_Buffereado es una estructura de datos introducida para el cálculo más preciso de reflexiones. La misma consiste en

almacenar, para cada pixel del mapa de entorno, una lista ordenada por distancias con las superficies proyectadas a dicho pixel, de forma tal de tener por cada elemento de la lista un par (α, RGB), que junto con la locación del pixel (que nos da **V**), nos construye una terna como el anteriormente mencionado. El algoritmo consiste, básicamente, en ir recorriendo todos los pixeles del mapa indexados por la expresión (4), eligiendo $t \in [0,1]$, buscando el que satisfaga

$$\underset{t\varepsilon[0,1]}{\text{Mínimo}} \{\text{AnguloEntre}\,(R, \alpha_t\, V_t - W) \text{ con } V_t = [[\,(1-t)\,R + t\,W]]\}$$

Dado a que estamos trabajando con un conjunto discreto de direcciones debido a la naturaleza discreta del mapa de entorno, el resultado final deberá ser retenido sólo si es menor que un cierto valor umbral, que llamaremos θ_{max} (obviamente, medido en radianes).

Si llamamos $\mathbf{P'}_t = \alpha_t \mathbf{V}_t$, y hacemos $\mathbf{P''}_t = [[\,\mathbf{P'}_t - \mathbf{W}\,]]$, tendremos que

$$\text{Error}_{\text{Paralaje}} = (1 - P'' \bullet [[\,\Sigma_{\text{reflexión}}'\,]]) / 2 = (1 - [[\,\alpha_t \mathbf{V}_t - \mathbf{W}\,]] \bullet \mathbf{R}) / 2$$

$$= \{1 - \mathit{Cos}(\mathit{AnguloEntre}(R, \alpha_t \mathbf{V}_t - \mathbf{W}))\} / 2$$

Desarrollando hasta segundo orden la serie de Taylor para ángulos pequeños tenemos

$$\text{Error}_{\text{Paralaje}} \approx (1 - (1 - \mathit{AnguloEntre}(R, \alpha_t \mathbf{V}_t - \mathbf{W})^2 / 2)) / 2$$

$$\approx \mathit{AnguloEntre}(R, \alpha_t \mathbf{V}_t - \mathbf{W})^2 / 4$$

$$\leq \theta_{max}^2 / 4$$

Como podemos ver, el error cometido por esta aproximación tiene una cota constante para todo el dominio, es decir, independiente del ángulo φ y del valor de α.

Por ejemplo con $\theta_{max} = 2°$, tenemos que:

$$\text{Error}_{\text{Paralaje}} \leq 3 \cdot 10^{-4}$$

Esto muestra claramente la gran precisión alcanzable con este modelo. En realidad, aún podemos encontrar una mejor cota para el error: En efecto, si cada pixel fue muestreado exactamente en su centro, el mayo error que podemos cometer al recorrer todos los pixeles estará dado por el ángulo subtendido desde el CP por la mitad de la diagonal del pixel más cercano al CP, es decir, el que está ubicado sobre uno de los ejes en un mapa de entorno cúbico. La expresión para este ángulo, en este caso, es:

$$\theta = \mathit{Arctg}\,(\sqrt{2} / \mathbf{N})$$

dónde N es la resolución del mapa (es decir, cada uno de los 6 lados del cubo es una pantalla de N*N pixeles). Desarrollando esta expresión en serie hasta segundo orden, obtenemos que

$$\boxed{Error_{Paralaje} \leq 1/(2N^2)}$$

Vemos que, para un mapa de 100*100 pixeles, $Error_{Paralaje} \leq 5 \cdot 10^{-5}$, para uno de 200*200 pixeles, $Error_{Paralaje} \leq 1.25 \cdot 10^{-5}$ y para uno de 300*300 pixeles, $Error_{Paralaje} \leq 5.5 \cdot 10^{-6}$.

Como puede apreciarse, estos resultados nos ofrecen una excelente acotación para el error cometido al utilizar este modelo, permitiéndonos tener una gran confianza en el mismo, confianza sólo comparable a la ofrecida por las técnicas de Trazado de Rayos.

10. Técnicas de eliminación de artefactos de discretización (Antialiasing)

Sintetizar una imagen con una computadora digital es muy distinto de exponer un film de una escena real. Las diferencias son interminables, aunque mucha de la investigación en computación gráfica está dirigida a hacer estas diferencias tan pequeñas como sea posible. Pero hay un problema fundamental al que estamos prendidos: las computadoras digitales modernas: no pueden representar imágenes continuas.

En el caso del uso de polígonos para aproximar las imágenes reales, todos estos tienen un problema común: sus aristas son irregulares. Este indeseable efecto, conocido como **serrado** o **escalonamiento**, es el resultado del 'todo o nada' de la discretización, por la cual cada pixel se reemplaza con el color del polígono o permanece sin cambio. El serrado es un ejemplo de un fenómeno conocido como **artefacto de discretización** (aliasing). La aplicación de técnicas para reducir o eliminar este fenómeno se conoce como eliminación de artefacto de discretización (antialiasing).

Desde el punto de vista del procesamiento de señales, el modelado puede ser considerado como la definición del mundo que se ha pensado por datos discretos y digitales que son procesados después por la síntesis de la imagen. Como el modelo del mundo deseado, al igual que el mundo real, es continuo, el modelado siempre involucra una conversión analógica - digital a la representación interna de la computadora. Después en la síntesis de la imagen, el modelo digital es remuestreado y recuantificado para adaptarse a los requrimentos del hardware de salida, que es mucho más drástico que el muestreo del modelado, lo que hace a este paso responsable de la generación de artificios.

El muestreo de una distribución de color bidimensional, I(x,y), puede ser descripto matemáticamente como una multiplicación por una función que guarda los valores de la función muestreada en los puntos muestreados, pero los hace cero en cualquier otro lugar:

$$I_s(x,y) = I(x,y) * \sum_i \sum_j \delta(x-i * \Delta x, y-j * \Delta y) \qquad (1)$$

La transformación de Fourier de esta señal es:

$$I_s^*(\alpha,\beta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_s(x,y)\, e^{-jx\alpha}\, e^{-jy\beta}\, dx\, dy =$$

$$1/(\Delta x\, \Delta y) \sum_i \sum_j I^*(\alpha - 2\pi i/\Delta x,\ \beta - 2\pi j/\Delta y) \qquad (2)$$

El muestreo será correcto si los requerimentos del teorema de muestreo se cumplen. El teorema de muestreo establece que una señal continua puede ser restablecida exactamente desde su muestreo por un filtro ideal pasa bajos solamente si es de banda limitada a una frecuencia máxima menor que la mitad de la frecuencia de muestreo. Esto es casi obvio desde la ecuación (2), dado que este repite el espectro de la señal infinitamente por períodos $2\pi/\Delta x$ y $2\pi/\Delta y$, lo cual significa que el espectro de señales no limitadas en banda serán destruidas por sus copias repetidas.

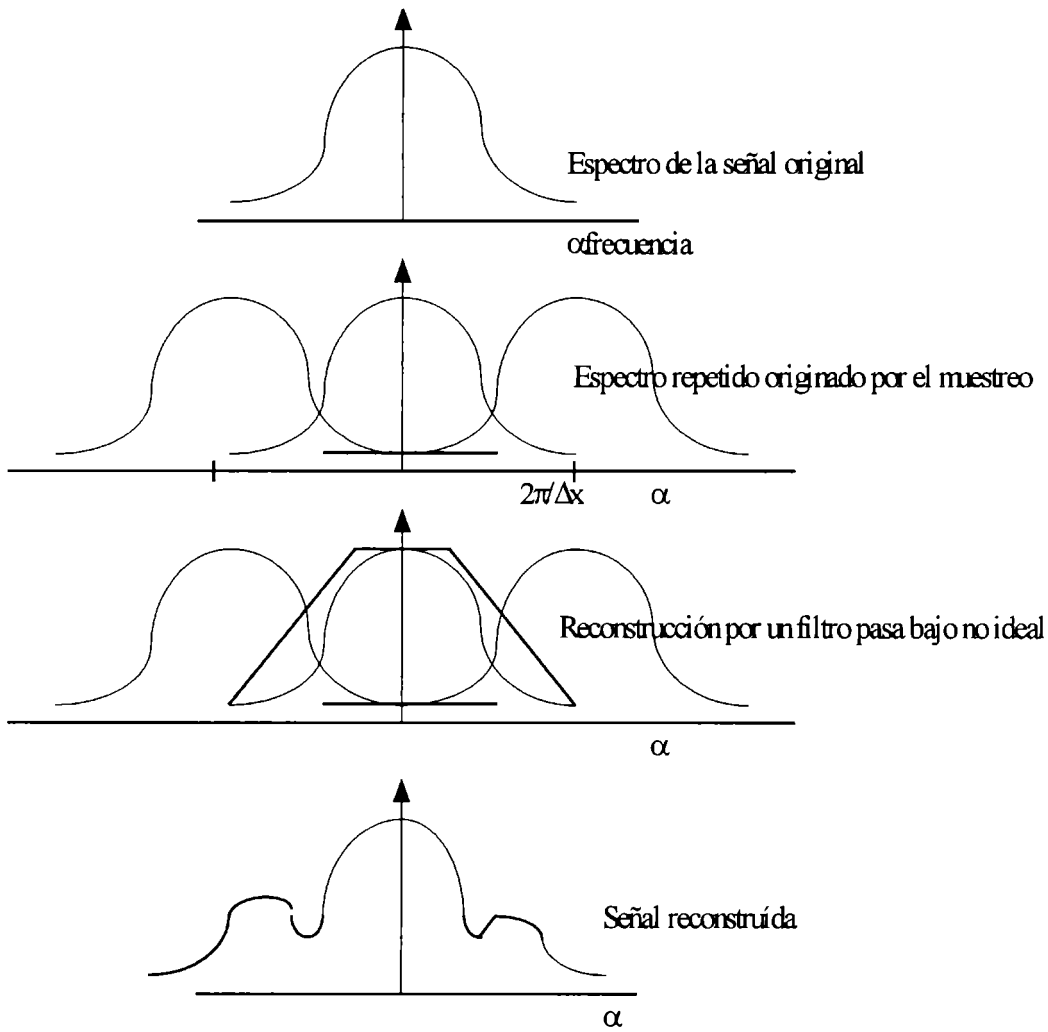


Figura 10.1: Análisis del espectro de un muestreo de la distribución del color

El mundo real no tiene una banda limitada, porque los objetos aparecen repentinamente cuando nos movemos a lo largo de un camino, introduciendo frecuencias infinitamente altas. Esto es, el teorema del muestreo no puede ser satisfecho nunca en computación gráfica, causando que el espectro repetido de la distribución del color se superponga, y destruya las componentes de baja frecuencia del espectro final. El fenómeno de la aparición de las componentes de alta frecuencia en rangos de baja frecuencia conduce un muestreo

incorrecto llamado aliasing (ver figura 10.1). La situación empeora por el método de reconstrucción de señales continuas

El resultado de un muestreo y reconstrucción insatisfactorios son conocidos en computación gráfica. Los lados de los polígonos tienen escalones que son alias de frecuencias altas no deseadas. Las esquinas de los escalones son provocadas por filtros pasa_bajos inadecuados que no suprimen aquellas componentes altas. La situación es aún peor para objetos pequeños de un tamaño comparable con pixeles, tales como pequeños caracteres o texturas, porque pueden desaparecer totalmente de la pantalla dependiendo de la grilla de muestreo actual. Para reducir los efectos irritantes del aliasing se pueden tomar distintas aproximaciones :

1.Aumento de la resolución de la muestra. Esta aproximación, además de restricciones tecnológicas claras, es ineficiente para la eliminación de efectos de aliasing, dado que el ojo humano es muy sensitivo a los patrones regulares que causan aliasing (Ver sección Aumento de resolución, pág.102)

2.Muestreo de área no ponderado. Esta aproximación utiliza el color de los polígonos que intersecan el pixel y el área que ocupa cada uno de los polígonos con relación al área total del pixel. (ver sección Muestreo de área no ponderada, pág. 103)

3.Muestreo de área ponderada. Esta aproximación se asemeja a la anterior, pero además tiene en cuenta la cercanía del polígono al centro del pixel. (ver sección Muestreo de área ponderada, pág 105)

10.1. Aumento de la resolución

Considere la utilización de un algoritmo para dibujar una línea negra de un pixel de grosor, con pendiente entre 0 y 1, sobre fondo blanco. El algoritmo establece el color del pixel más cercano a la línea en cada columna por la cual pasa. Cada vez que el algoritmo pasa de una columna a otra donde los pixeles más cercanos a la línea no están en la misma fila, hay un cambio abrupto en la línea dibujada, como se ilustra en la figura 10.2.

Suponga ahora que usamos un dispositivo de presentación con el doble de resolución horizontal y vertical. Como se ilustra en la figura 10.3, la línea pasa por dos veces más columnas y en consecuencia tiene el doble de escalones, pero el tamaño de cada escalón es la mitad en x y en y. Aunque la imagen resultante se ve mejor, esta mejora es a expensas de cuadruplicar el costo de la memoria, el ancho de banda de la memoria y el tiempo de discretización. El aumento de la resolución es una solución costosa que sólo reduce el problema del serrado, pero no elimina el problema.

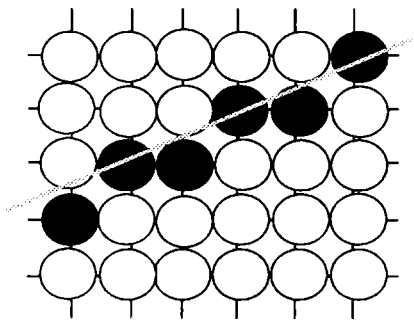


Figura 10.2: Línea estándar en una pantalla de dos niveles

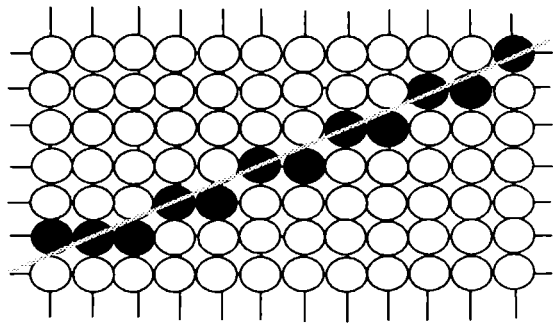


Figura 10.3: Misma línea en una pantalla con el doble de resolución

10.2. Muestreo de área no ponderada

El primer método para mejorar la calidad de las imágenes se puede desarrollar con base en la observación de que, aunque un polígono ideal (como la línea) tiene anchura de cero, el polígono que estamos dibujando tiene anchura distinta de cero. Un polígono discretizado ocupa un área finita en la pantalla; incluso la línea vertical u horizontal más delgada en la superficie de la pantalla tiene un pixel de grosor y las líneas en otros ángulos tienen anchura que varían en el polígono. Por ende, consideremos una línea como un rectángulo del grosor deseado que cubre una porción de la malla, como se ilustra en la figura 10.4. De esto se desprende que una línea no debe establecer a negro la intensidad de un solo pixel en una columna, sino brindar cierta cantidad de intensidad a cada pixel en las columnas que interseca (por supuesto, estas intensidades variables únicamente se pueden mostrar en pantallas que usan varios bits por pixel). Entonces, para las líneas de un pixel de grosor, sólo las líneas horizontales o verticales afectarían exactamente un pixel en su columna o fila. Para las líneas en otros ángulos, se asignaría más de un pixel en cada columna o fila, con la intensidad apropiada.

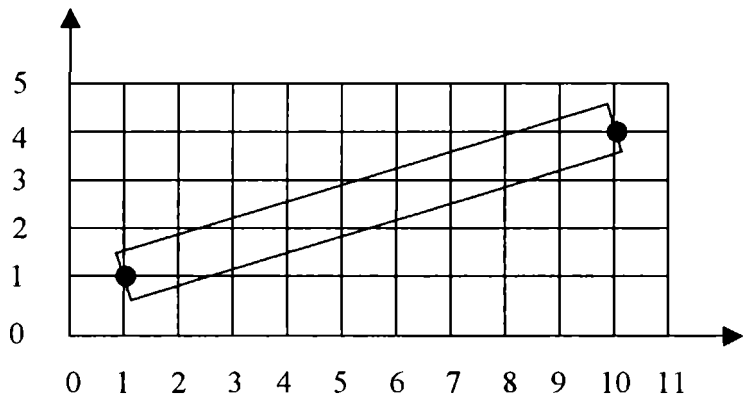


Figura 10.4: Línea con anchura distinta de cero que parte del punto (1, 1) al punto (10, 4)

Desde el punto de vista computacional, es bastante sencillo suponer que los pixeles forman un arreglo de azulejos cuadrados no superpuestos que cubren la pantalla, centradas en los puntos de la malla. Cuando nos referimos a un polígono que se sobrepone a un pixel o a una porción de éste, queremos decir que cubre (parte de) el azulejo; en ocasiones nos referimos al cuadrado como el área representada por el pixel. También supondremos que una línea contribuye a la intensidad de cada pixel con una cantidad proporcional al porcentaje de la baldosa del pixel que cubre. Un pixel completamente cubierto en una pantalla de negro sobre blanco tendrá color negro, mientras que un pixel parcialmente cubierto será de color gris, con una intensidad que dependerá de la cobertura del pixel. Esta técnica, aplicada a la línea de la figura 10.4, se presenta en la figura 10.5.

La asignación de la intensidad de un pixel en proporción a la cantidad de área cubierta por el polígono suaviza la brusca característica de encendido-apagado de la orilla del polígono y ofrece una transición más gradual entre el encendido total y el apagado. Este aspecto borroso hace que la línea se vea mejor a distancia, a pesar de que distribuye la transición de encendido-apagado por varios pixeles en una columna o fila.

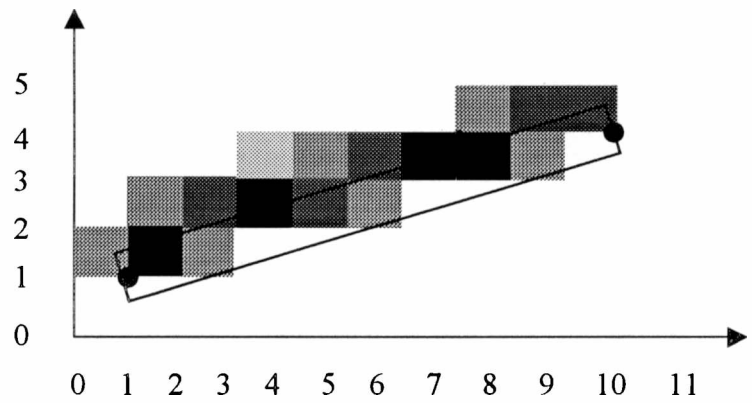


Figura 10.5: La intensidad de un pixel es proporcional al área cubierta por la línea

A la técnica de establecer la intensidad en forma proporcional al área cubierta la llamamos **muestreo de área no ponderada**. Esta técnica produce resultados notablemente mejores que al asignar los pixeles a su intensidad total o a intensidad de cero. El muestreo de área no ponderada tiene tres propiedades.

1. La intensidad de un pixel intersecado por una línea decrece conforme aumenta la distancia entre el centro del pixel y la orilla: cuanto más lejos esté la línea, menor será su influencia sobre la intensidad del pixel. Esta relación es obviamente verdarera porque la intensidad decrece al reducirse el área superpuesta, y esta área se reduce conforme la orilla de la línea se aleja del centro del pixel y se acerca a su frontera. Cuando la línea cubre por completo el pixel, el área de superposición y por ende la intensidad se encuentran al

máximo; cuando la orilla de la línea es tangente a la frontera, el área y la intensidad son cero.

2. Un polígono no puede influir en la intensidad del pixel si no lo interseca, es decir, si no interseca el azulejo cuadrado que el pixel representa.

3. Las áreas iguales contribuyen con igual intensidad, sin importar la distancia entre el centro del pixel y el área; lo único que importa es el área total superpuesta. Por lo tanto, una pequeña área en la esquina del pixel contribuye lo mismo que un área de igual tamaño cerca del centro.

10.3. Muestreo de área ponderada

En el muestreo de área ponderada se conserva las dos primeras propiedades del muestreo de área no ponderada (la intensidad decrece al reducirse el área superpuesta y las primitivas sólo contribuyen si se sobreponen al área representada por I pixel), pero se altera la tercera propiedad. Ahora las áreas iguales contribuyen en forma desigual: un área más pequeña cerca del centro del pixel tiene mayor influencia que una lejos del centro.

Para conservar la segunda propiedad tenemos que efectuar el siguiente cambio en la geometría del pixel. En el muestreo de área no ponderada, si la arista de un polígono se encuentra bastante cerca de la frontera del azulejo cuadrado que hemos usado hasta ahora para representar un pixel, pero en realidad no interseca esta frontera, no contribuirá a la intensidad del pixel. En nuestro nuevo método, el pixel representa un área circular mayor que el azulejo cuadrado; así, el polígono *sí* intersecará el área mayor y, por consiguiente, contribuirá a la intensidad del pixel. Observe que esto significa que las áreas relacionadas con pixeles adyacentes en realidad se sobreponen.

Para explicar el origen de los términos *no ponderada* y *ponderada*, definimos una **función de ponderación** que determina la influencia en la intensidad del pixel de una pequeña área dA de un polígono, como función de la distancia de dA al centro del pixel. Esta función es constante en el muestreo de área no ponderada y disminuye al aumentar la distancia en el caso del muestreo de área ponderada. Considere la función de ponderación como una función W(x, y) en el plano, cuya altura sobre el plano (x, y) indica la ponderación del área dA en (x, y). En el muestreo de área no ponderada, donde los pixeles se representan como azulejos cuadrados, el gráfico de W es una caja, como se ilustra en la figura 10.6.

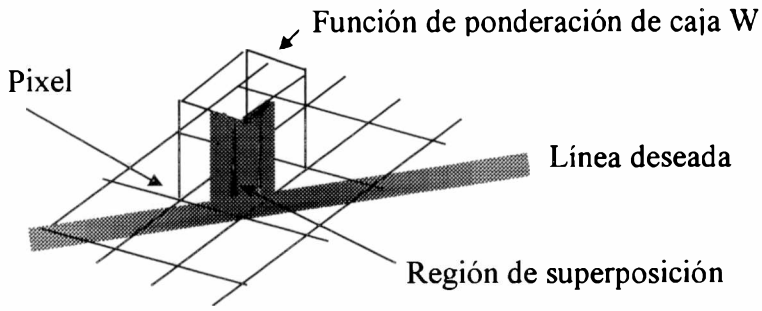


Figura 10.6: Filtro de caja para un pixel

En la figura 10.6 se muestran pixeles cuadrados, con centros indicados por las intersecciones de las líneas de la malla; la función de ponderación aparece como una caja cuya base es el pixel actual. La intensidad que brinda el área del pixel cubierto por la línea es el total de las contribuciones de intensidad de todas las pequeñas áreas de la región de superposición de la línea y el pixel. La intensidad que aporta cada pequeña área es proporcional al área multiplicada por la altura. Por lo tanto, la intensidad total es la integral de la función de ponderación en el área de superposición. El volumen representado por esta integral, Ws, siempre es una fracción entre 0 y 1, y la intensidad *I* del pixel es Imáx • Ws. En la figura 10.6, Ws es una rebanada de la caja. La función de ponderación también se conoce como **función de filtrado** y la caja se denomina así mismo **filtro de caja**. En el caso del muestreo de área no ponderada, la altura de la caja se normaliza en 1 par que el volumen de la caja sea 1 lo que ocasiona que una línea gruesa que cubra todo el pixel tenga intensidad *I* = *I*máx • 1 = *I*máx.

Construyamos ahora una función de ponderación para el muestreo de área ponderada; esta función debe asignar menor peso a las áreas pequeñas lejanas del centro del pixel que a las que se encuentran cerca. Elijamos una función de ponderación que sea la más sencilla de las funciones que decrecen con la distancia; por ejemplo, escogemos una función cuyo máximo corresponda al centro del pixel y que disminuya linalmente al aumentar la distancia con respecto al centro. Debido a la simetría rotacional, el gráfico de está función forma un cono circular. La base circular del cono (conocida con frecuencia como soporte del filtro) debe tener un radio mayor de lo que se esperaría; la teoría de filtrado nos indica que una buena elección para el radio es la distancia unitaria de la malla entera. De esta manera una línea que se encuentre lejos del centro de un pixel todavía puede influir en la intensidad de ese pixel; además, los soportes relacionados con los pixeles vecinos se sobreponen y por consiguiente un pequeño trozo del polígono puede contribuir en realidad a varios pixeles diferentes (figura 10.7). Esta superposición también asegura que no existan áreas de la malla que no estén cubiertas por un pixel, lo que sucedería si los pixeles circulares tuvieran radio igual a la mitad de la malla.

Como sucede con los filtros de caja, la suma de todas las contribuciones de intensidad del filtro de cono es el volumen bajo el cono y encima de la intersección de la base del cono y la línea; este volumen Ws es una sección

vertical del cono, como se muestra en la figura 10.7. Igual que en el filtro de caja, primero se normaliza la altura del cono para que el volumen debajo de todo el cono sea 1; esto permite que un pixel cuyo soporte está completamente cubierto por una línea se presente con la máxima intensidad. Aunque son bastante pequeñas las contribuciones de las áreas de las primitivas que están lejos del centro del pixel pero que sí intersecan el soporte del cono, un pixel cuyo centro esté suficientemente cerca de una línea recibe cierta contribución de intensidad de ella. El efecto neto del muestreo de área ponderada es reducir el contraste entre pixeles adyacentes para proporcionar transiciones más suaves. Específicamente, en el muestreo de área ponderada una línea horizontal o vertical con grosor unitario tiene más de un pixel intensificado en cada fila o columna, lo que no sucedería con el muestreo de área no ponderada.

El filtro cónico tiene dos útiles propiedades: simetría rotacional y reducción lineal de la función al aumentar la distancia radial. Preferimos la simetría rotacional porque no sólo hace que los cálculos de áreas sean independientes del ángulo de la línea, sino que además es teóricamente óptima. Observe, sin embargo, que la pendiente lineal del cono (y su radio) es sólo una aproximación de la función óptima, aunque el filtro de cono sigue siendo mejor que el filtro de caja, es sólo una aproximación de la función de filtrado óptima. Los filtros óptimos son computacionalmente muy costosos y los de caja son los más baratos; por lo tanto, los filtros de cono constituyen un razonable punto medio entre el costo y la calidad. Podríamos integrar sin muchas dificultades el filtro de cono a nuestros algoritmos de discretización.

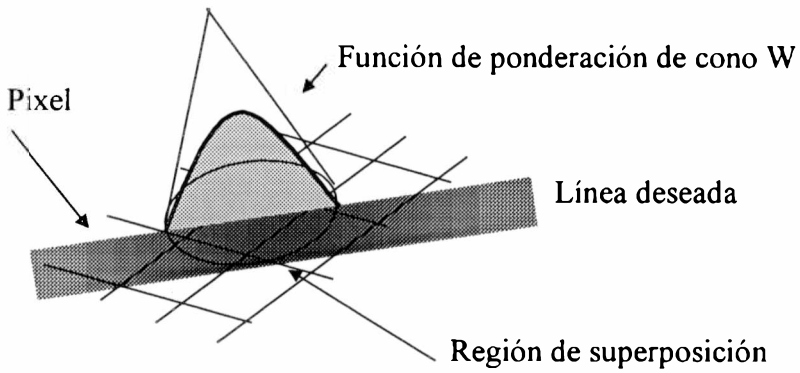


Figura 10.7: Filtro de cono para un pixel con diámetro de dos

11. Resultados y trabajos futuros

En este capítulo presentamos una comparación entre las imágenes obtenidas por el algoritmo presentado en este trabajo y el algoritmo que utiliza mapas de entornos Z_Buffereados desarrollado por Gustavo Patow [Patow95]. Elegimos este algoritmo para compararlo con el de mapas de entorno A_Buffereado, ya que éste plantea alternativas superadoras de algunas limitaciones que se observan en el primero.

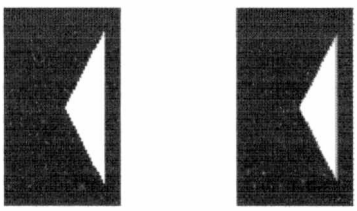
Al final del capítulo presentamos algunas mejoras que se pueden hacer al algoritmo aquí presentado en trabajos futuros.

11.1. Resultados

11.1.1. Resultados: Z_Buffer vs. A_Buffer

En las figuras 11.1 y 11.2 observamos dos triángulos dibujados por los algoritmos de mapa de entornos Z_Buffereados y A_Buffereado respectivamente. Las consideraciones referidas a iluminación y reflexión las veremos más adelante.

Los dos triángulos dibujados aquí son idénticos. Recordemos que nuestro algoritmo trabaja aproximando los objetos con triángulos. Es probable que si dibujamos uno o más objetos de mayor complejidad, estos estén compuestos por muchos triángulos más pequeños. Nosotros elegimos mostrarlos con esta dimensión con el objetivo de visualizar más claramente los resultados obtenidos.



a) Imagen muestreada usando Z_Buffer

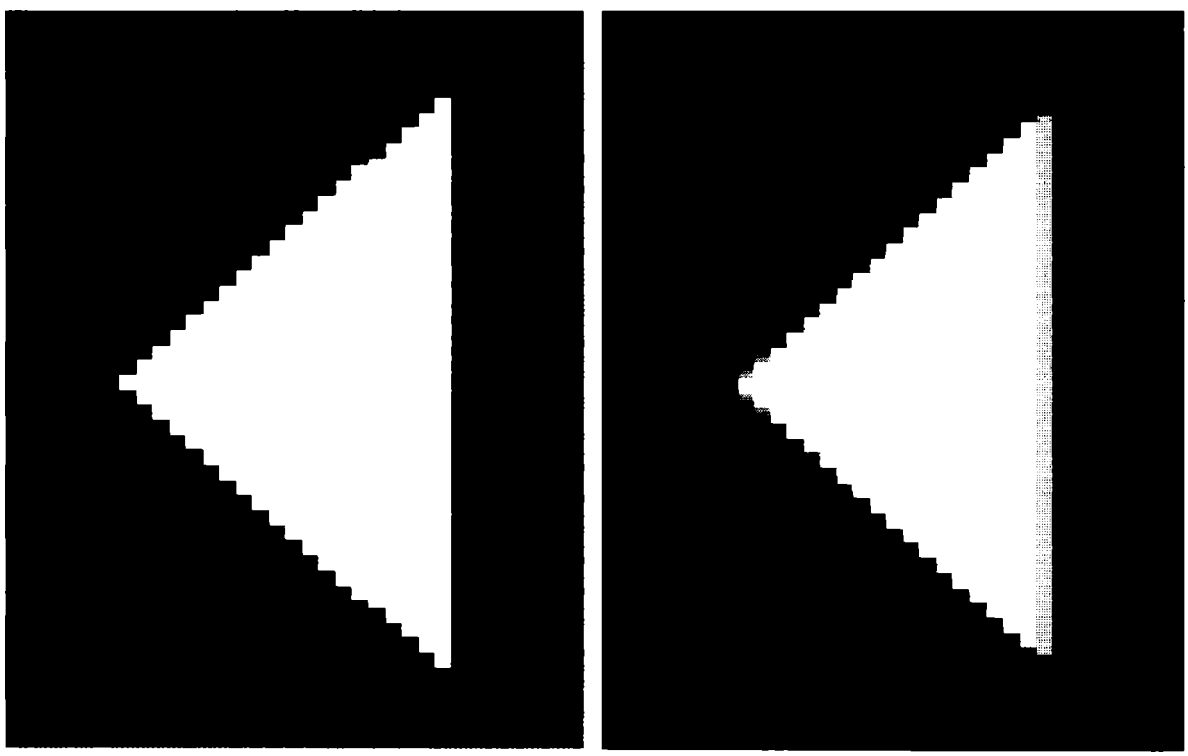
b) Imagen muestreada usando A_Buffer

Figura 11.1: imágenes dibujadas por los algoritmos que utilizan Z_buffer y A_Buffer en el tamaño original en que fueron muestreadas

Como se ve en la figura 11.1 el triángulo muestreado usando el Z_Buffer tiene un efecto escalonado más visible en sus lados que el A_Buffer. Esto se debe a que el Z_Buffer utiliza la información referida al color del objeto en ese

pixel, y lo cubre con ese color. En el caso de nuestro ejemplo considera que el pixel es del color del triángulo (blanco) o del background (negro). El A_Buffer, en cambio, divide el pixel en 32 subpixeles (máscara de 4x8) y calcula el color final a partir de la contribución de cada uno de ellos. La diferencia se hace visible en los lados, debido a que este es el lugar donde el objeto no cubre todo el pixel. En este caso el Z_Buffer cubre el pixel con el color del objeto; el A_Buffer calcula el color en base a la contribución de sus subpixeles, alguno de ellos del color del objeto y otros del color del background. Para visualizar esto más claramente ampliamos las imágenes de la figura 11.1 y las mostramos en la figura 11.2.

Los efectos de antialiasing visualizados en la imagen de la figura 11.1 b) y 11.2 b) se deben a la utilización de la técnica de muestreo de área no ponderada.



a) Imagen muestreda con el Z_Buffer b) Imagen muestreda con el A_Buffer

Figura 11.2: Ampliación de las imágenes de la figura 11.1

11.1.2. Reflexiones

En esta sección presentamos los resultados relacionados con el procesamiento de reflexiones, comparando una imagen resultante al utilizar un Z_Buffer y una que usa el A_Buffer. La figura 11.3 muestra la imagen utilizada antes de procesar las reflexiones. El triángulo que se ve en color blanco es el objeto reflectivo. El triángulo rojo se encuentra delante del triángulo verde

mirando desde el centro de proyección, por lo que este último se encuentra tapado en parte.

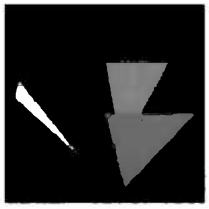
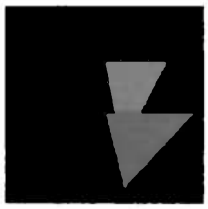


Figura 11.3: Escena analizada

En la figura 11.4 a) presentamos la imagen de la figura 11.3 después de realizar el cálculo de reflexiones utilizando el mapa de ambiente Z_Buffereado, mientras que en la figura 11.4 b) se realizó el cálculo de reflexiones usando el mapa de ambiente A_Buffereado. Como se puede observar el resultado obtenido no es el mismo; esto se debe al problema del objeto escondido (Ver sección 7.4.2). Usando el mapa de ambiente Z_Buffereado si un objeto no es visible desde el centro de proyección del mapa este no se verá reflejado, aunque sea visible indirectamente a través de su reflexión en otro objeto. Por ejemplo la parte del triángulo verde que está oculta desde el centro de proyección no se refleja en el triángulo reflectivo. Como se aprecia en la figura 11.4 b) esto no sucede cuando usamos el mapa de entorno A_Buffereado. Aún la parte del triángulo verde que no es visible desde el punto de vista del observador, se ve reflejada en el triángulo reflectivo.

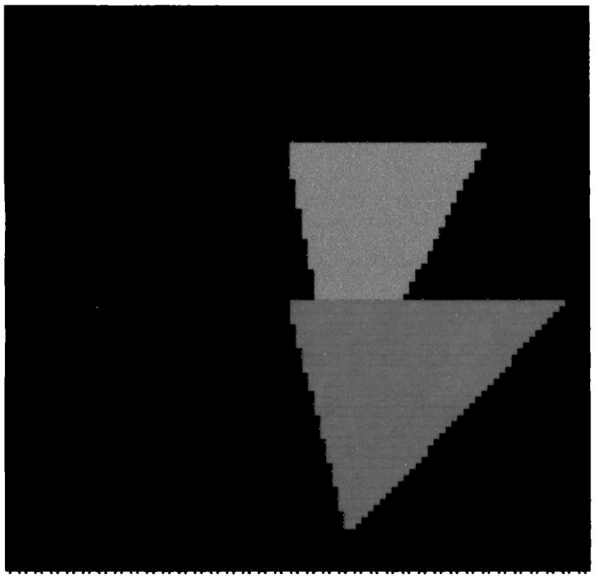
a)Imagen resultante de aplicar reflexiones con un mapade entorno Z_Buffereado



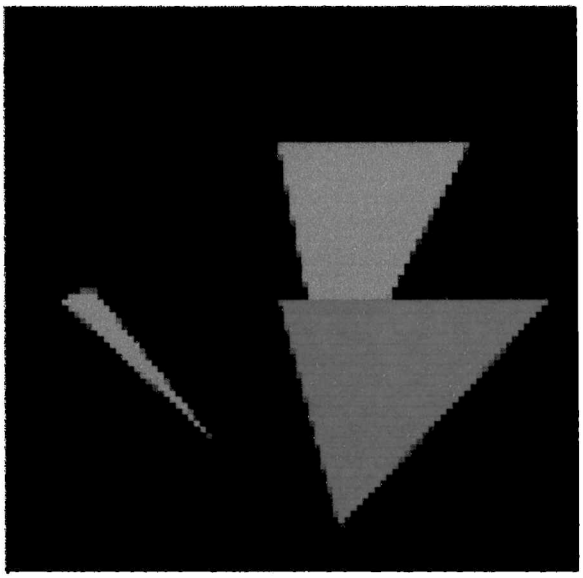
b)Imagen resultante de aplicar reflexiones con un mapade entorno A_Buffereado

Figura 11.4: imágenes resultantes después de aplicar reflexiones a la imagen mostrada en la figura 11.3 utilizando mapa de entorno Z_Buffereado y A_Buffereado respectivamente en su tamaño original

En las figuras 11.5 a) y 11.5 b) vemos las imágenes de las figuras 11.4 a) y b) ampliadas.



a)Imagen rendereada usando mapas de entorno Z_Buffereados



b)Imagen rendereada usando mapas de entorno A_Buffereados

Figura 11.5: imágenes mostradas en la figura 11.4 ampliadas

11.1.3. Orden de ejecución

Cabe aclarar que el código que implementamos en este trabajo realiza dos pasadas; una para calcular el mapa de ambiente único A_Buffereado y otra para calcular la imagen final. Es decir el algoritmo es de orden O(n), donde n es el número de objetos en la escena. Si quisiéramos solucionar el problema del objeto escondido de la sección anterior usando un Z_Buffer, debería generarse un mapa de ambiente por cada objeto reflectivo centrado en el objeto mismo.

11.2. Optimización y Trabajos Futuros

11.2.1. Uso del mapa de entorno global para el cálculo de la imagen final

Con esto nos referimos al hecho de que cuando generamos al mapa de ambiente, una de las seis caras (BOTTOM), tiene la misma información que se usará después en el render final pero no calcula el color (reflexiones, refracciones y sombras).

La optimización consistiría en reusar la información, ya generada, de esta cara del mapa de ambiente y en la generación de la imagen final sólo calcular el color. Para esto sería necesario guardar, además de la información geométrica, información acerca del material, el normal, y todo lo necesario

para calcular el color. La misma debe ser guardada en la estructura de los fragmentos.

Concretamente en el caso de nuestro código significaría que sólo la función ArmarEnvMapGlobal (que arma el mapa de ambiente global) llamaría a ScanLineTriangles que actualmente también es llamada por GenerarImagenFinal y esta última función sólo llamaría a la función PackUnderMask que calcula el color final.

Esta optimización no fue implementada en nuestro trabajo para permitir variar libremente la resolución del mapa de ambiente global en forma independiente de la imagen final, para poder buscar la relación óptima.

11.2.2. Reflexiones y Refracciones recursivas a la Ray Tracing

La estructura del mapa de entorno A_Buffereado permite calcular reflecciones y refracciones múltiples, es decir, el caso en que un objeto reflectivo/refractivo, refleje/refracte a otro objeto reflectivo/refractivo.

Esto se podría realizar operando de la misma manera que lo hace el trazado de rayos recursivo (Ver sección Trazado de Rayos Recursivo), pero con la diferencia de que la búsqueda del pixel reflejado o refractado se haría sobre el mapa de entorno A_Buffereado, que tiene información de toda la escena y no sobre la escena misma (3D) como lo hace el trazado de rayos. Recordemos que el mapa de entorno es una proyección 2D de la escena que resulta en una aproximación lo suficientemente buena como para no tener que implementar el trazado de rayos que es más costoso.

11.2.3. Iluminación global

Un modelo de iluminación calcula el color en un punto en función de la luz emitida en forma directa por las fuentes luminosas, y de la luz que llega al punto después de la reflexión y la transmisión en su superficies y en otras. Esta luz transmitida y reflejada en forma indirecta se llama iluminación global. En el algoritmo presentado en este trabajo hemos modelado la iluminación global con un término de iluminación ambiental que se ha mantenido constante para todos los puntos en los objetos, independientemente de las posiciones del objeto y del observador, así como de la presencia o ausencia de objetos cercanos que podrían bloquear la luz ambiental.

Se han estudiado algunos algoritmos para generar imágenes que subrayen las contribuciones de la iluminación global y que pueden contribuir a completar este trabajo. El algoritmo de traza de rayos recursivo combina la determinación de superficies visibles y el sombreado para presentar sombras, reflexión y refracción. Así, la transmisión y la reflexión especular complementan la iluminación local. En cambio, los métodos de radiosidad separan totalmente el sombreado y la determinación de superficies visibles,

modelando todas las interacciones del ambiente con las fuentes luminosas, primero en una etapa independiente de la vista, para calcular después imágenes para los puntos de observación deseados usando algoritmos convencionales de superficies visibles e interpolación.

Los algoritmos dependientes de la vista son adecuados para manejar fenómenos especulares altamente dependientes de la posición del observador, pero pueden requerir trabajo adicional para modelar fenómenos difusos que cambian poco en grandes áreas. En cambio los algoritmos independientes de la vista modelan con eficiencia fenómenos difusos pero requieren grandes cantidades de almacenamiento para disponer de información suficiente acerca de los fenómenos especulares.

Todas estas estrategias tratan de resolver lo que Kajiya llamó ecuación de generación, que expresa la luz transferida de un punto a otro en función de la intensidad de la luz emitida del primer punto al segundo y la intensidad de luz emitida desde todos los demás puntos que llega al primero y es reflejada del primero al segundo. La luz transferida de cada uno de estos puntos al primero se expresa, recursivamente por la ecuación de generación. Kajiya presenta la ecuación de generación como:

$$I(x,x') = g(x,x') [\varepsilon(x,x') + \int_S \rho(x,x',x'') I(x',x'')dx'']$$

donde x, x' y x'' son puntos en el ambiente; I(x,x') se relaciona con la intensidad que pasa de x' a x; g(x,x') es un término geométrico que es cero cuando x y x' están ocultos uno con respecto al otro y $1/r^2$ cuando son visibles, donde r es la distancia entre ellos; y $\varepsilon(x,x')$ se relaciona con la intensidad de la luz que es emitida de x' a x. La evaluación inicial de $g(x,x')\ \varepsilon(x,x')$ para x en el punto de observación logra la determinación de superficies visibles en la esfera alrededor de x. La integral abarca todos los puntos en todas las superficies S. p(x,x',x'') se relaciona con la intensidad de la luz reflejada (incluyendo la reflexión tanto especular como difusa) de x'' a x desde la superficie en x'. Así, la ecuación de generación establece que la luz de x' que llega a x consiste en la luz emitida por la propia x' y la luz dispersa por x' a todas las demás superficies, las cuales emiten luz y dispersan recursivamente luz de otras superficies.

11.2.4. Corrección de los errores de Aliasing usando los Exact A_Buffers

Se utiliza para la corrección de los errores de aliasing que aparecen por la deformación introducida por la proyección en perspectiva.

El algoritmo EXACT (Cálculo exacto de área de cobertura) resuelve el problema de Eliminación de Superficies Escondidas sobre un nivel de subpixel.

El uso de máscaras de subpixeles para antialiasing causan algunos problemas con el algoritmo de superficies escondidas sobre un nivel de pixel, difíciles de superar. Las aproximaciones del bien conocido algoritmo de

A_Buffer son reemplazadas por una solución exacta que evita pixeles erráticos a lo largo de superficies intersectantes o que se tocan entre sí.

Con EXACT el problema del algoritmo de superficies escondidas sobre un nivel de subpixel es resuelto con la ayuda de p-máscaras. Las P-máscaras (máscaras de prioridad) son máscaras de subpixel que indican para cada subpixel cuál de dos planos dados es más cercanos a quién mira.

El rasterizado produce efectos de aliasing. Si un filtro cuadrado es usado para implementar antialiasing, el brillo y el color de los pixeles de los bordes son funciones del área del pixel cubierta por el objeto, así como de los colores del objeto. La intensidad ideal debería ser descripta por la fórmula $I = 1/A \Sigma_i I_i A_i$, donde A_i e I_i son las áreas e intensidades de las superficies visibles dentro del pixel y A es el área total del pixel. Las máscaras de subpixel pueden ser usadas para calcular la fracción del área del pixel cubierta por un objeto. Si el punto muestreado está fuera del polígono, su valor de Z es útil para un algoritmo de superficies escondidas correcto. Una correcta eliminación de superficies para el área del pixel es necesaria.

Apéndice

A continuación se presentan los conceptos matemáticos más importantes utilizados en nuestro trabajo, en especial aquellos relacionados con vectores y matrices. Las transformaciones de traslación, escalamiento y rotación que analizamos aquí son indispensables en muchas operaciones gráficas. Un programa de aplicación para la planificación de ciudades usaría la traslación para colocar símbolos de árboles y edificios en los lugares apropiados, la rotación para orientar los objetos y el escalamiento para alterar el tamaño de los símbolo. En términos generales, muchas aplicaciones emplean transformaciones geométricas para cambiar la posición, la orientación y el tamaño de los objetos en un dibujo.

1. Puntos en el espacio

Obviamente, la representación del mundo que nos rodea es de una complejidad enorme, pero podemos comenzar por definir, simplemente, a un punto en el espacio.

Un punto en el plano puede representarse mediante un sistema de coordenadas como el de la figura 1, en el que el punto P tiene asociado las coordenadas (xp, yp) que dan su ubicación de forma única con respecto al origen O.

Es muy fácil extender esta idea al mundo tridimensional que nos rodea, simplemente agregando una tercera componente que indica la altura con respecto al plano (x, y), que tomamos desde altura O, ver figura 2. Quedando entonces que P tiene asociada las coordenadas (xp, yp, zp), lo que nos da su ubicación completa, esta vez con respecto al origen del espacio de tres dimensiones.

Un punto, de esta forma definido, es lo que en matemática se denomina un **vector**, siendo xp, yp y zp sus coordenadas. Podemos, sin problema, imaginar al vector como una flecha, con su final en el origen y la punta en el punto P.

Qué pasa si tenemos un vector P que nos da la posición de un punto con respecto al origen y tenemos otro vector Q que nos indica lo mismo, pero respecto de P? Cuál es la posición del punto final? Pues, simplemente, la respuesta es que el punto estará señalado por un vector R dado por:

R = P + Q = (xr, yr, zr) que como se puede ver en la figura 3, sus componentes serán:

xr = xp + xq

yr = yp + yq

$zr = zp + zq$

Qué pasa si queremos saber cuál es la distancia que hay entre un punto y el origen? Para esto consideremos la figura 4. La distancia D es, por el Teorema de Pitágoras, $D = (x^2 + y^2)^{1/2}$, o sea que, $D^2 = x \cdot x + y \cdot y = |P|$, donde $|P|$ es el módulo del vector P. Esta última expresión puede verse como si multiplicásemos el vector, componente a componente, por sí mismo. Entonces podemos escribirlo, finalmente, así:

$D^2 = P \bullet P$, donde el punto significa $x \cdot x + y \cdot y$, que es un producto entre vectores llamado **producto escalar**. En tres dimensiones, esto mismo se escribe:

$P \bullet P = px \cdot px + py \cdot py + pz \cdot pz$

Este producto se llama escalar porque para dos vectores, nos da en número escalar.

Así,

$P \bullet Q = px \cdot qx + py \cdot qy + pz \cdot qz$

representa el producto escalar entre el vector P y el vector Q. Pero en este caso, qué significa este número? Bueno, para ello existe una propiedad del producto escalar que dice que:

$P \bullet Q = |P| \, |Q| \cos(\beta)$

Donde β es el ángulo entre los dos vectores, ver figura 5. Qué pasa si $\beta = 0°$? Entonces, el producto escalar será sólo el producto entre los módulos de los vectores, que siempre será mayor que cero. Pero si el ángulo es de 90°? Entonces $P \bullet Q = 0$. Y si el ángulo está entre 90° y 180°? Pues, $P \bullet Q < 0$, por lo tanto, con solo sumar los productos $Pi \cdot Qi$, con $i = x, y, z$, sabremos la relación entre los dos vectores.

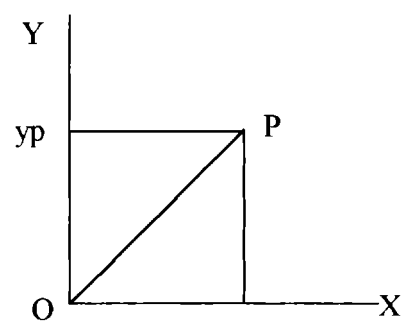


Figura 1: un punto en el plano

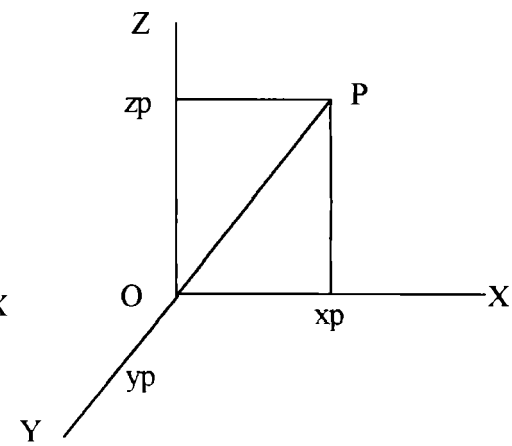


Figura 2: un punto en el espacio

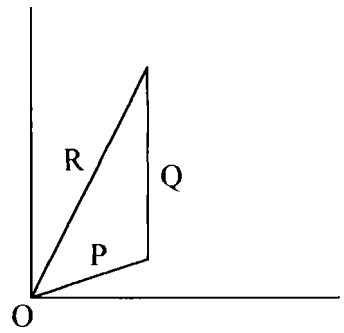


Figura 3: suma de vectores

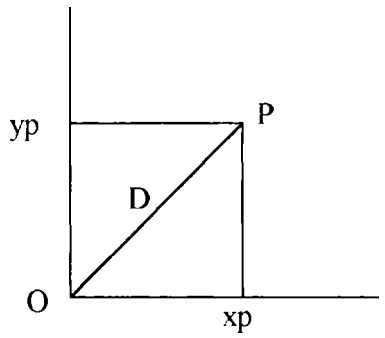


Figura 4: distancia

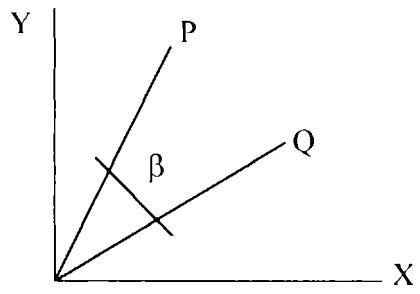


Figura 5: producto escalar

2. Producto Vectorial

El producto entre dos vectores, llamado **producto vectorial**, nos da como resultado otro vector, perpendicular a los dos originales, como más adelante veremos.

Este producto se define así:

A = B * C

donde * representa el símbolo de producto vectorial, obviamente, A, B y C son vectores. Las componentes de A están dadas por:

Ax = By · Cz - Bz · Cy

Ay = Bz · Cx - Bx · Cz

Az = Bx · Cy - By · Cx

Gráficamente, estos tres vectores se encuentran representados según la figura 6, dónde los vectores B y C forman con el origen O un plano, y el vector A es perpendicular a los otros dos.

Existe una interesante propiedad del producto vectorial, la no-conmutabilidad, es decir, si cambiamos el orden de la multiplicación, obtendremos otro vector, llamado A', que es:

A' = C * B = - A, es decir que tiene el sentido opuesto al de A. Por lo tanto vemos que:

C * B <> B * C, de lo que obtenemos que el producto vectorial no es conmutativo.

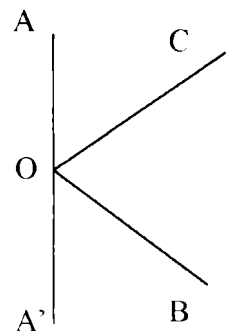


Figura 6: producto vectorial

3. Rotaciones

Hagamos ahora una rotación alrededor del origen en el plano como muestra la figura 7, de tal forma que transforme al vector P en el vector Q al rotarlo en un ángulo α.

Con medios trigonométricos encontramos que :

$Qx = R \cos(\alpha + \beta)$

$Qy = R \operatorname{sen}(\alpha + \beta)$

$Px = R \cos(\beta)$

$Py = R \operatorname{sen}(\beta)$

como las distancias al origen deben conservarse, por lo que R es el mismo para los dos vectores.

Utilizando propiedades trigonométricas del seno y del coseno, hacemos:

$Qx = R \cos(\alpha + \beta) = R \{\cos(\alpha) \cos(\beta) - \operatorname{sen}(\alpha) \operatorname{sen}(\beta)\}$

$= Px \cos(\alpha) - Py \operatorname{sen}(\alpha)$

$Qy = R \operatorname{sen}(\alpha + \beta) = R \{\operatorname{sen}(\alpha) \cos(\beta) + \cos(\alpha) \operatorname{sen}(\beta)\}$

$= Px \text{ sen } (\alpha) + Py \cos (\alpha)$

En síntesis, hemos hallado las componentes de Q en función de las componentes de P:

$Qx = Px \cos (\alpha) - Py \text{ sen } (\alpha)$

$Qy = Px \text{ sen } (\alpha) + Py \cos (\alpha)$

Exactamente esta misma expresión puede reescribirse, utilizando una notación matricial, de la siguiente forma:

$$\begin{vmatrix} Qx \\ Qy \end{vmatrix} = \begin{vmatrix} \cos(\alpha) & -\text{Sen}(\alpha) \\ \text{Sen}(\alpha) & \cos(\alpha) \end{vmatrix} \cdot \begin{vmatrix} Px \\ Py \end{vmatrix} = R \cdot P$$

dónde R es la matriz de senos y cosenos (matriz de rotación). Esto puede leerse así: multiplicamos escalarmente la primera fila de la matriz con el vector P y esa operación nos da Qx, de la misma manera con la segunda fila nos da Qy.

Esta forma de escribir la rotación puede parecer complicada en un principio, pero es extremadamente útil, como más adelante veremos.

Considere como vectores las dos filas de la matriz R. se puede demostrar que los vectores tienen tres propiedades:

1. Cada uno es un vector unidad.

2. Cada uno es perpendicular al otro (su producto escalar es cero).

3. El primer y segundo vectores se rotarán por R(a) para que caigan sobre los ejes x y y positivos, respectivamente (en presencia de las condiciones 1 y 2, esta propiedad equivale a que la matriz tenga determinante de 1).

Las dos primeras propiedades también son verdaderas para las columnas de la matriz. Estas propiedades sugieren dos maneras útiles de obtener una matriz de rotación cuando se conoce el efecto deseado de la rotación. Una matriz que tiene estas propiedades se denomina **ortogonal especial**.

Una matriz de rotación ortogonal, conserva los ángulos y las longitudes. Estas rotaciones se denominan de **cuerpo rígido**, ya que el cuerpo u objeto que se transforma no se distorsiona de ninguna manera.

Extendamos ahora la situación para el caso tridimensional, ver figura 8. Entonces una rotación alrededor del eje z es:

$Qx = Px \cos(\alpha) - Py \operatorname{sen}(\alpha)$

$Qy = Px \operatorname{sen}(\alpha) + Py \cos(\alpha)$

$Qz = Pz$ (no cambia)

o, en forma matricial (usando esta vez matrices de 3 x 3)

$$Q = \begin{vmatrix} Qx \\ Qy \\ Qz \end{vmatrix} = \begin{vmatrix} \cos\alpha & -\operatorname{sen}\alpha & 0 \\ \operatorname{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} Px \\ Py \\ Pz \end{vmatrix}$$

Para una rotación alrededor del eje x tendremos:

$Qx = Px$

$Qy = Py \cos(\alpha) - Pz \operatorname{sen}(\alpha)$

$Qz = Px \operatorname{sen}(\alpha) + Pz \cos(\alpha)$

o, en notación matricial:

$$Q = \begin{vmatrix} Qx \\ Qy \\ Qz \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\operatorname{sen}\alpha \\ 0 & \operatorname{sen}\alpha & \cos\alpha \end{vmatrix} \cdot \begin{vmatrix} Px \\ Py \\ Pz \end{vmatrix}$$

y para una rotación alrededor del eje y:

$Qx = Px \cos(\alpha) + Pz \operatorname{sen}(\alpha)$

$Qy = Py$

$Qz = Px \operatorname{sen}(\alpha) + Pz \cos(\alpha)$

o, en notación matricial:

$$Q = \begin{vmatrix} Qx \\ Qy \\ Qz \end{vmatrix} = \begin{vmatrix} \cos\alpha & 0 & -\operatorname{sen}\alpha \\ 0 & 1 & 0 \\ \operatorname{sen}\alpha & 0 & \cos\alpha \end{vmatrix} \cdot \begin{vmatrix} Px \\ Py \\ Pz \end{vmatrix}$$

Existe una matriz particularmente interesante, llamada la matriz Identidad, que tiene la siguiente forma:

$$I = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

y que, al rotar a un vector, no realiza ningún cambio, esto es $Q = I \cdot P = P$. Rota un vector en 0° (nada).

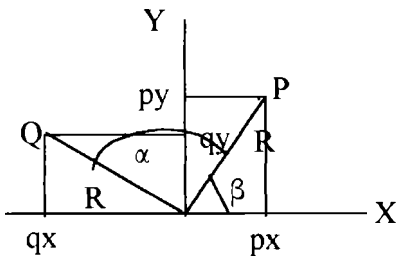


Figura 7: Rotaciones en el plano

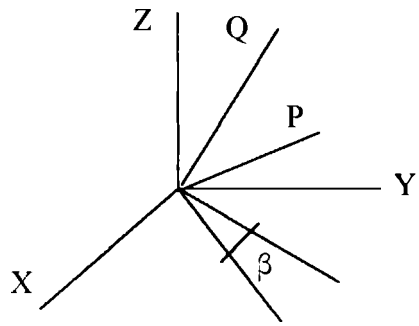


Figura 8: Rotaciones en el espacio alrede alrededor del eje z.

Pero existe un pequeño problema con respecto a las rotaciones. Al igual que el producto vectorial, tampoco conmutan, es decir, que dadas dos matrices A y B, al multiplicarlas hallamos que $A \cdot B$ es distinto de $B \cdot A$. Por lo tanto, al aplicar rotaciones sucesivas, el resultado depende fuertemente del orden en que se lo haga.

Por ejemplo, si primero se aplica al vector P una rotación representada por la matriz R1 y luego se aplica otra rotación R2, podemos describir la situación como:

$$P2 = R1 \cdot P$$

P3 = R2 · P2, donde P2 es un vector intermedio, que podemos reemplazar, obteniendo, finalmente:

$$P3 = R2 \cdot (R1 \cdot P)$$

que es lo mismo que hacer una única rotación

$$Q = R2 \cdot R1 \cdot P$$

en ese orden estricto.

4. Traslaciones

Podemos trasladar puntos en el plano (x, y) a nuevas posiciones si añadimos valores de traslación a las coordenadas de los puntos. Para cada punto P que se moverá dx unidades en forma paralela al eje x y dy unidades en paralelo al eje y, para llegar al nuevo punto P', podemos escribir

$$P'x = Px + dx$$

$$P'y = Py + dy \qquad (1)$$

es decir,

P' = P + T, donde T es la matriz de traslación

En forma matricial de 3 x 3, tenemos las traslaciones como:

$$\begin{vmatrix} P'x \\ P'y \\ P'z \end{vmatrix} = \begin{vmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & dz \end{vmatrix} \cdot \begin{vmatrix} Px \\ Py \\ Pz \end{vmatrix}$$

¿Qué sucede si el punto P es trasladado por T1 a P' y luego por T2 a P''? El resultado que esperamos intuitivamente es una traslación neta T, podemos escribir la situación como:

$$P' = T1 \cdot P$$

$$P'' = T2 \cdot P'$$

ahora si reemplazamos, obtenemos finalmente:

$$P'' = T2 \cdot (T1 \cdot P) = (T2 \cdot T1) \cdot P$$

donde , T = T2 · T1. El producto de matrices se conoce como concatenación o composición de T1 y T2.

Podemos trasladar un objeto aplicando la ecuación (1) a cada uno de sus puntos. Sin embargo, en un objeto cada línea está formada por un número infinito de puntos, por lo que este proceso requeriría un tiempo infinito. Por fortuna, podemos trasladar todos los puntos de una línea con sólo trasladar los puntos extremos y dibujar una línea entre estos puntos trasladados.

5. Escalamientos

Los puntos se pueden escalar (estirar) por sx sobre el eje x y por sy en el eje y para obtener nuevos puntos si se aplican las multiplicaciones

$P'x = sx \cdot Px$

$P'y = sy \cdot Py$

es decir,

$P' = S \cdot P$, donde S es la matriz de escalamiento.

El escalamiento se presenta en forma matricial, usando matrices de 3 x 3, como:

$$\begin{vmatrix} P'x \\ P'y \\ P'z \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & sz \end{vmatrix} \cdot \begin{vmatrix} Px \\ Py \\ Pz \end{vmatrix}$$

Así como las traslaciones sucesivas son aditivas, podemos esperar que los escalamientos sean multiplicativos. Dado

$P' = S1 \cdot P$

$P'' = S2 \cdot P'$,

se obtiene

$P'' = S2 \cdot (S1 \cdot P) = (S2 \cdot S1) \cdot P$

donde,

$S = S2 \cdot S1$

Por lo tanto, los escalamientos son multiplicativos.

Las proporciones de un objeto después del escalamiento no son afectadas si se usa escalamiento uniforme, donde sx = sy. En cambio, si se verán afectadas si se aplica un escalamiento diferencial, donde sx <> sy.

6. Coordenadas homogéneas

Si los puntos se expresan con coordenadas homogéneas, las tres transformaciones se pueden tratar como multiplicaciones. En las coordenadas homogéneas, se añade una tercera coordenada a un punto; nos referimos al caso de transformaciones bidimensionales. Entonces, en lugar de representar un punto con un par de números (x, y), se representa con tres, (x, y, W). Al mismo tiempo, se dice que dos conjuntos de coordenadas homogéneas (x, y, W) y (x', y', W') representan el mismo punto si y sólo si uno es múltiplo del otro. Es decir, cada punto tiene varias representaciones con coordenadas

homogéneas. Además, al menos una de las coordenadas homogéneas debe ser distinta de cero. Si la coordenada W es diferente de cero, es posible usarla como divisor: (x, y, W) representa el mismo punto que (x/W, y/W, 1). Los números x/W y y/W se denominan coordenadas cartesianas del punto homogéneo. Los puntos con W = 0 se conocen como puntos en el infinito.

Los triples de coordenadas usualmente representan puntos en el espacio tridimensional, pero aquí los usamos para representar puntos en el espacio bidimensional. La relación es la siguiente: si tomamos los triples que representan un mismo punto, es decir, todos los triples (tx, ty, tW), con t <> 0, obtenemos una línea en el espacio tridimensional. Si homogeneizamos el punto (lo dividimos entre W), obtenemos un punto de la forma (x, y, 1). Por consiguiente, los puntos homogeneizados constituyen el plano definido por la ecuación W = 1 en el espacio (x, y, W).

Así como las transformaciones bidimensionales se pueden representar en matrices de 3 x 3 usando coordenadas homogéneas, las transformaciones tridimensionales se pueden representar con matrices de 4 x 4, siempre y cuando usemos representaciones de coordenadas homogéneas de los puntos en el espacio tridimensional. Así, en lugar de representar un punto como (x, y, z), lo hacemos como (x, y, z, W), donde dos de estos cuádruplos representan el mismo punto si uno es multiplicador distinto de cero del otro. Como sucede en el espacio bidimensional, la representación estándar de un punto (x, y, z, W) con W <> 0 se indica con (x/W, y/W, z/W, 1). La transformación de un punto a esta forma se denomina homogeneización, igual que antes. También existe una interpretación geométrica. Cada punto en el espacio tridimensional se representa con una línea que pasa por el origen en el espacio de cuatro dimensiones, y las representaciones homogeneizadas de estos puntos forman en subespacio tridimensional de un espacio de cuatro dimensiones definido por la ecuación W = 1.

7. Composición de transformaciones

El propósito básico de la composición de transformaciones es ganar eficiencia al aplicar a un punto una sola transformación compuesta, en lugar de aplicar una serie sucesiva de transformaciones.

Considere la rotación de un objeto con respecto a un punto arbitrario P1. Como sólo sabemos rotar con respecto al origen, podemos convertir nuestro problema original a tres problemas separados. Así, para rotar con respecto a P1, necesitamos una secuencia de tres transformaciones fundamentales:

1. Trasladar P1 para que quede en el origen.

2. Rotar.

3. Trasladar para que el punto en el origen regrese a P1.

Esta secuencia se ilustra en la figura 9, en la cual se rota un triángulo con respecto a P1(x1, y1). La primera traslación es (-x1, -y1), mientras que la segunda es la inversa, (x1, y1). El resultado es bastante diferente del que se obtiene al aplicar sólo la rotación.

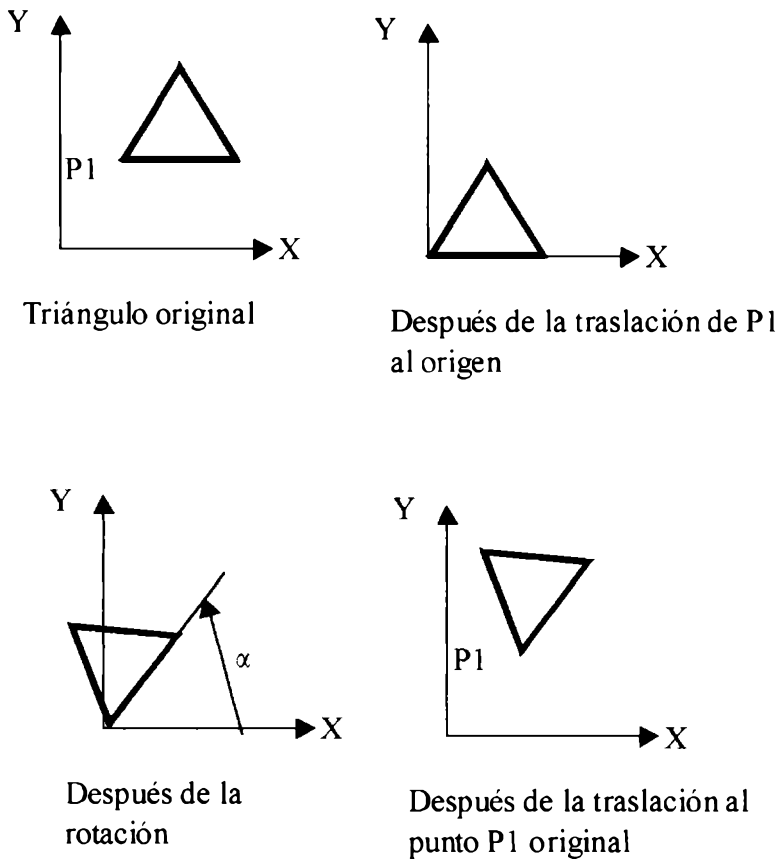


Figura 9: Rotación, de un triángulo con respecto al punto P1, con un ángulo α.

La rotación neta es:

$T(x1, y1) \cdot R(\alpha) \cdot T(-x1, -y1)$.

Se utiliza una estrategia similar para escalar un objeto con respecto a un punto arbitrario P1. Primero se hace el traslado de manera que P1 pase al origen, después se escala y luego se traslada de regreso a P1.

De manera similar se pude componer matrices de transformación tridimensionales. De esta forma, las rotaciones, representadas por matrices de 3 x 3, constituyen la esquina superior izquierda de una matriz de 4 x 4, mientras que las traslaciones están, en las tres primeras componentes de la última columna de la derecha de dicha matriz.

Bibliografía

[Arvo93] Arvo, J., "Transfer Equations in Global Illumination", Global Illumination, SIGGRAPH'93 Course Notes, Vol. 42, August 1993.

[Lischinski94] Lischinski, D.; B. Smits and D. P. Greenberg, "Bounds and Error Estimates for Radiosity", Computer Graphics Proceedings, Annual Conference Series, 1994, pp. 67-74.

[Arvo94] Arvo, J; K. Torrance and B. Smits, "A Framework for the Analysis of Error in Global Illumination Algorithms", Computer Graphics Proceedings, Annual Conference Series, 1994, pp. 75-84.

[Born64] Born, M and E. Wolf, "Principles of Optics", The Macmillan Company, NY 1964,Capítulo III.

[Greene86] Greene N. "Environment Mapping and Other Applications of World Projections", IEEE CG&A, 6(11), November 1986, pp 21-29.

[Arvo96] Arvo, J. "Backward Ray Tracing", Developments in Ray Tracing, Siggraph' 86 course notes, Vol. 12, August 1986 (Retyped January 1995).

[Glassner91] A. Glassner, Editor. "An Introduction to Ray Tracing", Academic Press, 1991.

[Blinn76] Blinn, J.F. and M.E. Newell, "Texture and Reflection in Computer Generated Images", Comunications of the ACM, 19(10), Octubre 1976, 542-547.

[Foley90] Foley, J., A. van Dam, S. Feiner and J. Hughes, "Computer Graphics Principles and Practice, 2nd. Edition", Addison-Wesley, 1990.

[Foley96] Foley, J., A. van Dam, S. Feiner, J. Hughes and Richard L. Phillips, "Introducción a la graficación por computador", Addison-Wesley, 1996.

[Upstill92] Upstill, S.,"The RenderMan Companion". Addison-Wesley, 1992.

[Patow95] Patow, G. "Accurate Reflections Through Z-Buffered Environment Map", Proceedings of the XV International Conference of the Chilean Computer Sience Society (SCCC), 1995, pp 385-392.

[Brunet93] Brunet, P.; Navazo, I.; Vinacua, A. "A Modelling Scheme for the approximate Representation of Closed Surfaces". In: Geometric Modelling, Computing/Supplementum, 8. Springer-Verlag, 1993. P. 75-90. ISBN 3-540-82399-9

[Carpenter84] Carpenter, L., "The A-Buffer, an Antialiased Hidden Surface Method", Siggraph 84, pp 103-108.

[Watt92] Watt A. y Watt M., "Advanced Animation and Rendering Techniques. Theory and Practice", Addison-Wesley, 1992

[Schilling93] "EXACT:Algorithm and Hardware Architecture for an Improved A-Buffer", Schilling, A. and S. Wolfgang, Siggraph 93, pp 85-91

[Kat79a] Kay, D. S., "Transparency, Refraction and Rey Tracing for Computer Synthesized Images", tesis de maestría, Program of Computer Graphics, Cornell University, Ithaca, NuevA York, enero de 1979.

[Whit80] Whitted, T., "An Improved Illumination Model for Shaded Display", CACM, 23(6), junio de 1980, pp 343-349.

[Musgrave93] Musgrave, F. K., "Methods for Realistic Landscape Rendering", Doctoral Dissertion, Yale University, 1990.

[Nishita85] Nishita, T., Nakamae, E., "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interrflection", Siggraph 85, pp23-30.

[Voorhies94] Voorhies, D. and J. Foran, "Reflection Vector Shading Hardware", Proccedings of Siggraph 94, julio de 1994, 163-166.