Instituto Politécnico Nacional

Escuela Superior de Física y Matemáticas

Método de Phong

TESIS

QUE PARA OBTENER EL TÍTULO DE

Licenciado en Física y Matemáticas

PRESENTA

Edgar Omar Curiel Anaya

Director de Tesis:

Eduardo Virueña

Sinodales:

Adrián Alcántar

Víctor Hugo Ibarra

Luis Fernando Ramírez

Julio Cesar Salas

Julio de 2006

Agradecimientos

En primer lugar a *Dios*, ya que Él está detrás de todo lo que puedo explicar y de lo que nunca podré, como la vida misma.

A mis padres María Estela Anaya Velázquez y José Curiel Martínez, que con su infinito amor, paciencia y cuidado soy lo que soy ahora, sólo espero que me alcance la vida para pagar la deuda que tengo con ellos y si muriera y me dieran a elegir a mis padres de nueva cuenta, los volvería a escoger.

A mis hermanos Ana Laura, Oscar Octavio y Christian Michel Curiel Anaya, ya que nada hubiera sido como lo es ahora si no existieran, gracias por todos los momentos de felicidad que he vivido a su lado y gracias también por ser como son.

Al profesor Eduardo Virueña, quien esperó pacientemente por más de dos años para el término de esta tesis, sin contar desde luego con su invaluable ayuda y apoyo en todo momento.

A mis sinodales, profesores Adrián Alcántar, Víctor Hugo Ibarra, Luis Fernando Ramírez y Julio Cesar Salas por sus comentarios efectuados en la realización de este trabajo y por acceder a regalarme un poco de su valioso tiempo.

Para tí también Irma Sánchez, ya que sin tu apoyo y tus buenos deseos no lo hubiera conseguido, gracias por tu comprensión, gracias por ser única y auténtica.

A todos mis amigos que he tenido a lo largo y ancho de mi vida.

Índice general

Índice de figuras

Introducción

Desde la prehistoria, el hombre dibujaba sobre las paredes, o el techo de las cuevas animales que cazaba, o también pintaban escenas de significado mágico, por ejemplo, el rito de la fertilidad, etc. Los materiales que se usaban entonces eran el carbón de leña y diferentes tierras o grasas de animales. Las pinturas prehistóricas más conocidas son las de la cueva de Altamira en Santillana de Mar (Cantabria), donde se hallan los famosos bisontes así como las de la cueva de Lascaux en Francia. En España, se hallan también pinturas tanto en el área del Cantábrico y los Pirineos como en el área del Levante (Cataluña, Valencia, Murcia y Andalucía).

El ser humano siempre se ha preocupado por representar los objetos tal y como sus ojos los ven y, a través del tiempo, deja su huella traduciendo la impresión que le transmite un objeto reproduciendo su forma, su tamaño y su volumen, por medio de trazos, como en el arte egipcio, griego y japonés, sugiriendo sobre todo el aspecto del relieve por el juego de las sombras y de la luz; este último modo de expresión es ya visible en los frescos de Pompeya y en los artistas del Renacimiento italiano, como Leonardo Da Vinci. En términos generales, el arte se ha desarrollado en función de las condiciones de existencia de cada época, cultura y conocimientos acerca de los instrumentos y técnicas utilizadas por los artistas.

Las técnicas del dibujo son diversas y han variado con el tiempo; en general, los instrumentos más utilizados han sido el lápiz, la pluma, el carbón, el pastel, el óleo, etc. La sutil gradación de la luz y la sombra en *La Virgen de las Rocas*, de Leonardo, o en *Mujer bañándose en un arroyo*, de Rembrandt, no se hubieran logrado en aquel entonces más que con el óleo.

Las computadoras se han convertido en una herramienta poderosa para la producción rápida y económica de ilustraciones. Prácticamente no existe ningún área en la cual no puedan utilizarse los despliegues gráficos con alguna ventaja; así que no es sorprendente hallar las gráficas por computadora en tantas aplicaciones. Aunque las primeras aplicaciones en ciencia e ingeniería tenían que basarse en un equipo costoso y complicado, los adelantos en la tecnología de computación han hecho de las gráficas una herramienta práctica. Hoy en día, se puede advertir que estas gráficas se utilizan rutinariamente en áreas diversas como: administración, industria, gobierno, arte, entretenimiento, publicidad, educación, investigación, capacitación y medicina [7]. Con el avance de la tecnología y desde la construcción de la Marck 1 (primera computadora electromecánica) hasta las computadoras modernas es posible realizar casi cualquier tipo de gráfica.

Lo que se verá en el capítulo 1 corresponde al modelado de una cámara sintética, el cual es indispensable para la representación de cualquier objeto tridimensional y en esta sección se examina una técnica de recorte para conjuntos conexos que se podrá utilizar en posibles extensiones. Es importante hacer notar que se modelan las funciones básicas de este aparato.

La parte central de este escrito se encuentra en el capítulo 2 ya que los despliegues realistas de objetos se obtienen generando proyecciones en perspectiva (capítulo 1) con superficies ocultas eliminadas y después aplicando modelos de sombreado y color a las superficies visibles. Un modelo de sombreado se utiliza para calcular la intensidad de la luz que debe observarse cuando se visualiza una superficie. Estos cálculos de intensidad se basan en las propiedades ópticas de las superficies, las posiciones relativas de las superficies y su orientación con respecto a las fuentes de luz. Primero se considera la forma en que pueden modelarse los cálculos de intensidad a partir de las leyes de la óptica y después se aplica un cierto modelo de color como lo es el método de Phong. La intensidad de la luz que se observa en cada superficie de un objeto depende del tipo de fuentes de luz situadas en la vecindad y de las características de la superficie del objeto. Algunos objetos tienen superficies brillosas y algunos tienen superficies opacas o mate. Además, algunos objetos se construyen con materiales opacos, mientras que otros son más o menos transparentes. Todo modelo de sombreado que origine intensidades realistas sobre la superficie de un objeto, debe tomar en consideración estas propiedades.

En el capítulo 3 se explora una técnica de modelado por medio de fractales y funciones de ruido para la construcción de ciertos mapas de apariencia celulosa y rocosa, se examina también el manejo de color y sombreado por medio de la interpolación. Se verá que se puede obtener una gran gama de texturas utilizando parámetros adecuados dentro de las funciones que conforman cada mapa.

Capítulo 1

Elaboración de una cámara sintética

El propósito principal del siguiente capítulo es mostrar el modelo de una cámara sintética, teniendo en consideración que se cuenta con un sistema de coordenadas rectangulares *(x, y, z)* previamente establecido, así como el manejo de operaciones básicas entre vectores, el libro de Spiegel [12] proporciona una descripción más completa acerca del tema. Por otro lado, se expondrán conceptos fundamentales para el proceso de visualización en tercera dimensión como lo son ventana, *viewport* y proyecciones. En los libros de Foley ([4],[5]) se puede encontrar la explicación de los temas a tratar en este capítulo.

1.1. Ventana y viewport

Algunos paquetes gráficos permiten al programador especificar coordenadas de salida en el sistema de coordenadas de mundo, usando cualquier unidad para el programa de aplicación ya sean metros, milímetros, centímetros, años luz, etc. El término *mundo* se utiliza de forma regular ya que el programa de aplicación representa al mundo real que va a ser interactivamente creado o desplegado para el usuario.

Dadas las especificaciones de las primitivas de salida en coordenadas de mundo, las subrutinas del paquete gráfico nos indican como mapear las coordenadas de mundo al sistema de coordenadas de la pantalla[1], una forma de

[1]Un término más general sería coordenadas de dispositivo.

obtener éstas, es que la aplicación del programador especifique una región rectangular en coordenadas de mundo llamada ventana y su correspondiente región rectangular en coordenadas de pantalla llamada *viewport*, en la cual es mapeado el contenido de la ventana. Si la ventana y el *viewport* no tienen la misma razón en sus dimensiones ocurrirá un escalamiento no uniforme y, por tanto, la primitiva de salida será deformada, aunque existen primitivas que no son afectadas por lo antes ya mencionado. En la figura 1.1 se muestra este concepto.

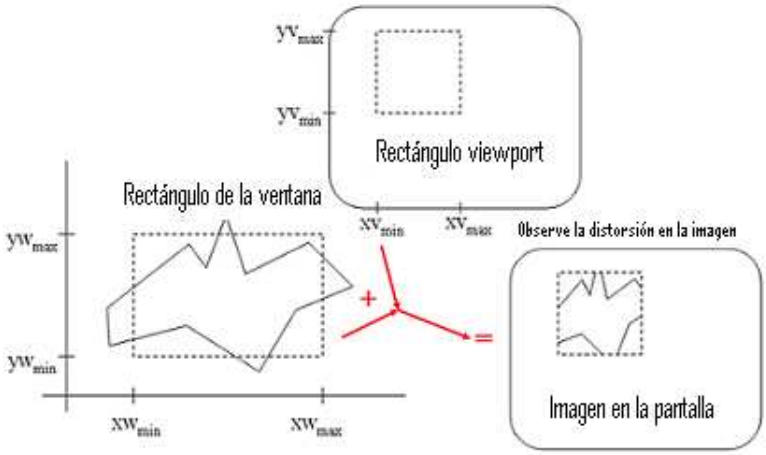

Figura 1.1: Ventana y viewport.

1.2. El proceso de una vista 3D

El proceso de visualización 3D es inherentemente más complicado que el proceso de visualización de 2D. En 2D simplemente se especifica la ventana sobre el mundo en 2D y un *viewport* sobre una superficie de dimensión dos.

La complejidad del caso 3D es ocasionada, en parte, por el aumento de una dimensión, así como el hecho de realizar un desplegado en 2D. La solución a este problema del desplegado de 3D a 2D es realizar proyecciones, las cuales transforman objetos 3D en una proyección plana 2D. En una visualización 3D, se tiene que especificar el volumen de vista en el mundo físico, una proyección sobre un plano y un *viewport* donde se visualiza la superficie. Conceptualmente, los objetos físicos son cortados contra un volumen de visualización para posteriormente ser proyectados. La figura 1.2 muestra el modelo conceptual del proceso de visualización 3D.

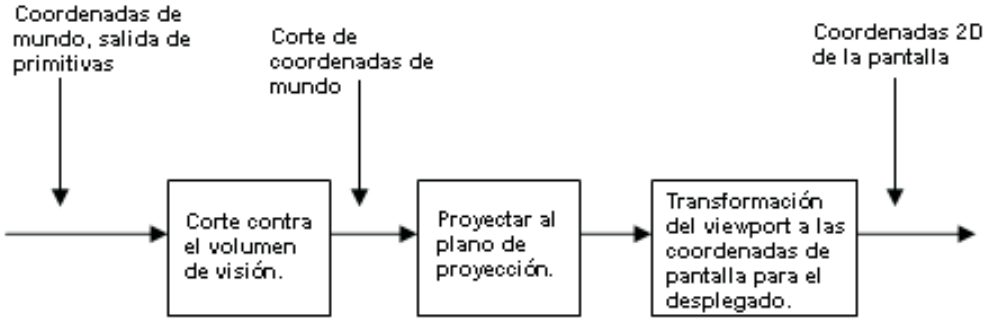

Figura 1.2: Modelo conceptual del proceso de visualización 3D.

1.2.1. Proyecciones

En general, las proyecciones transforman puntos en un sistema de coordenadas de dimensión n en puntos de sistemas de coordenadas de dimensión menor a n. Es de interés particular tratar proyecciones 3D a 2D. La proyección de objetos tridimensionales es definida por rayos con un punto de emisión llamado centro de proyección, y que pasa por cada punto del objeto e intersecta al plano de proyección. En este caso en particular se tratarán las proyecciones geométricas planas (el nombre se debe a que la proyección es

dentro de un plano) y pueden ser divididas en dos clases base: perspectiva y paralela; la diferencia entre ellas radica en el centro de proyección al plano de la misma. Si la distancia de uno al otro es finita se dice que es proyección en perspectiva y si la distancia es infinita entonces la proyección será denominada paralela. El efecto visual de la proyección perspectiva es similar al sistema de visión humana, donde el ojo es el centro de proyección y el tamaño de los objetos varía inversamente con la distancia de los objetos al centro de proyección, la cual se utilizará.

En la proyección perspectiva, cualquier conjunto de líneas paralelas no se visualizan como tales en el plano de proyección, sino que converge a un punto fuga. Como máximo se pueden encontrar tres de estos puntos correspondientes al número de ejes principales cortados por el plano de proyección. Por ejemplo, si el plano de proyección lo corta solamente el eje z (y es normal a él), únicamente el eje z tiene un punto fuga principal, ya que las líneas son paralelas tanto al eje y o x siendo además paralelas al plano de proyección y no tienen punto fuga. Las proyecciones perspectivas son categorizadas por su número principal de puntos fuga y, por tanto, por el número de ejes del plano de proyección que es cortado. La figura 1.3 muestra dos diferentes proyecciones perspectivas con un punto fuga de un cubo.

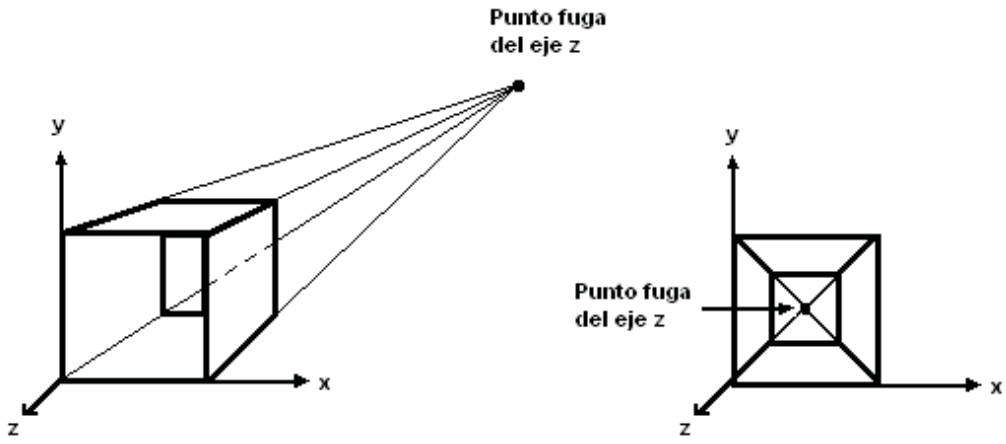

Figura 1.3: Proyección perspectiva con un punto.

Es claro que hay proyecciones a un punto fuga porque las líneas paralelas a los ejes x y y no convergen, únicamente las líneas paralelas al eje z lo hacen. En ingeniería, arquitectura, diseño industrial, etc., se usan comúnmente dos puntos en perspectiva, por otro lado, es poco frecuente usar tres puntos en perspectiva, ya que se pierde realismo.

1.2.2. Especificación de una vista 3D arbitraria

Como lo sugiere la figura 1.2 el proceso de visualización 3D involucra no sólo una proyección, sino además un volumen de visión en el cual el mundo 3D es cortado. Tanto la proyección como el volumen de visión dan toda la información necesaria para cortar y proyectar en el espacio 2D. Se especificará el volumen de vista a construir por medio del concepto de proyección geométrica plana incluido en la sección 1.2.1.

El plano de proyección (en ocasiones también llamado plano de visión), está definido por un punto llamado "punto de referencia de vista" (VRP) y una normal al plano VPN. El plano de visión puede estar en cualquier lado con respecto a los objetos de mundo que serán proyectados; podría estar de frente, de lado, a lo largo, o detrás de los objetos.

Dado el plano de visión, se necesita una ventana dentro de él. El objetivo de la ventana es similar a lo ya explicado en la sección 1.1 y, por tanto, todo lo que se encuentre fuera de la ventana no será proyectado, cabe mencionar que la ventana es de suma importancia en la definición del volumen de visión, para definirla en el plano de visión es necesaria la especificación de las coordenadas mínimas y máximas de ésta a lo largo de dos ejes ortogonales. Tales ejes son parte del sistema de referencia de coordenadas de vista VRC.

El origen del sistema es llamado eje n. El segundo eje de VRC se encuentra desde el vector up VUP de visión, el cual determina la dirección del eje v sobre el plano de visión. El eje v se ha definido por la proyección VUP paralela a VPN en el plano de visión. La dirección queda definida de tal forma que u, v, n sean ortogonales como en la figura 1.4.

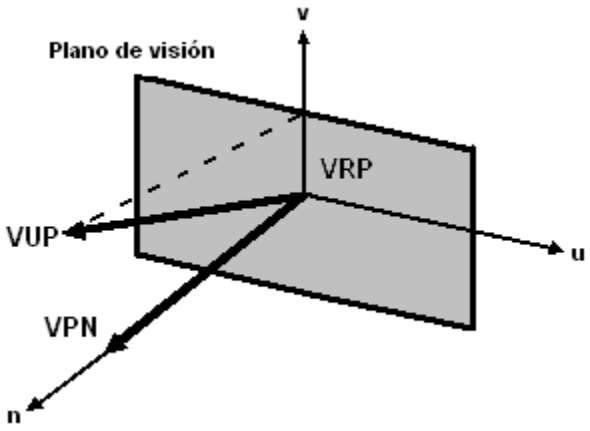

Figura 1.4: Plano de visión.

Con el sistema VRC definido, los valores mínimo y máximo de u y v de la ventana pueden definirse como en la figura 1.5: En la siguiente ilustración se muestra que el centro de la ventana CW no necesita estar en el centro del punto de referencia de visión VRP.

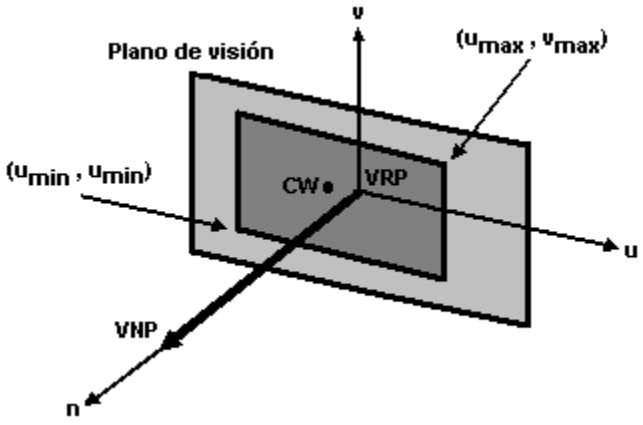

Figura 1.5: Plano de visión con ventana.

El centro de la dirección de la proyección DOP se define por el punto de referencia de proyección PRP más un indicador del tipo de proyección. Si la proyección es de tipo perspectiva, el centro de proyección es PRP. El PRP está especificado en el sistema VRC y no en el sistema de coordenas de

mundo. La posición de PRP relativa a VRP no cambia aún cuando VUP y VRP se muevan. La ventaja de esto es que el programador puede especificar la dirección de proyección requerida.

El volumen de visión acota una porción del mundo que es cortada y después proyectada al plano de visión. Para la proyección perspectiva, el volumen de visión es una semipiramide infinita con vértice en PRP y los lados pasan por las esquinas de la ventana como en la figura 1.6. Los objetos con posiciones detrás del centro de proyección no están incluidos en el volumen de visión y por tanto no son proyectados. En realidad el ojo humano "visualiza" en forma de cono irregular como volumen de visión y no tiene forma piramidal, aunque ésta es una buena aproximación a la realidad.

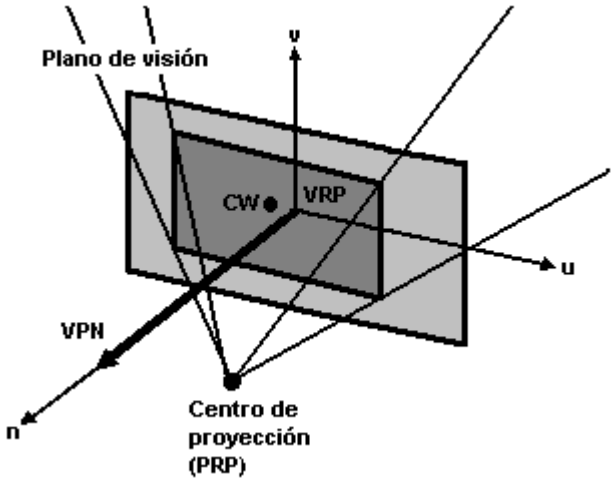

Figura 1.6: Volumen de visión.

Es posible que se quiera tener un volumen de visión finito y así limitar el número de primitivas proyectadas en el plano de visión. En la figura 1.7 muestra como el volumen de visión se vuelve finito con un plano frontal de corte y un plano posterior de corte respecto a VPN, una vez hecho esto se centra la atención en la parte del mundo que nos interesa. Para la proyección perspectiva los objetos muy distantes del centro de proyección podrían visualizarse en el plano de visión como "gotas" sin forma distinguible y además, si el objeto está muy cerca del centro de proyección puede extenderse a lo

largo de toda la pantalla, también perdiendo la forma y si se especifica adecuadamente el volumen de visión, se ahorran dichos problemas, para hacerlo únicamente se da z_{max} y z_{min}.

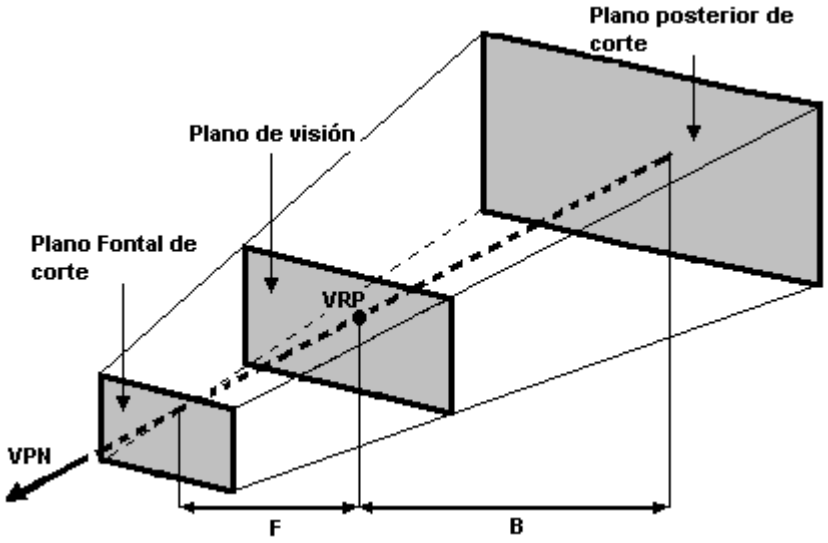

Figura 1.7: Recorte del volumen de visión.

1.3. Recorte

En el transcurso de esta sección se explicará de forma analítica un algoritmo de recorte, si bien este proceso no corta objetos tridimensionales, puede ser de gran ayuda para quien esté interesado en hacer cortes de líneas contra rectángulos. El algoritmo es válido no sólo para rectángulos sino también para todo polígono convexo. Antes de comenzar este método, es necesario primero resolver el problema de caracterizar tanto el rectángulo de corte como un punto P situado dentro de dicho rectángulo. Para resolver el primer problema únicamente se requiere utilizar 4 bits que contengan sus dimensiones $x_{min}, x_{max}, y_{min}$ y y_{max} con las condiciones que $x_{min} \leq x_{max}$ y $y_{min} \leq y_{max}$, para el segundo, es de manera casi natural decir que cualquier punto $P = (x, y)$ está dentro de la zona de corte, siempre y cuando:

$$x_{min} \leq x \leq x_{max} \text{ y } y_{min} \leq y \leq y_{max}.$$

Verificar lo anterior punto a punto en cada línea sería una gran carga para el procesador, para el recorte sólo son necesarios los puntos extremos de cada línea y se olvida por completo de la supuesta infinidad de puntos intermedios que tiene dicha primitiva, ya que en realidad para fines prácticos, se tiene que considerar que los recursos de una computadora son finitos y por lo cual dicha línea se representa totalmente por un número finito de píxeles.

Para explicar el algoritmo se considera la figura 1.8, la cual tiene 9 zonas con 9 códigos distintos, el rectángulo de corte queda representado por el código 0000, para poder asignar los códigos a cada zona se tiene que tomar en cuenta la dirección de los planos que conforman el rectángulo de corte como se observa en la figura 1.8, asignando 1 si algún punto está situado por debajo de cualquier plano que conforme la zona de corte, y en caso contrario 0 si está por arriba.

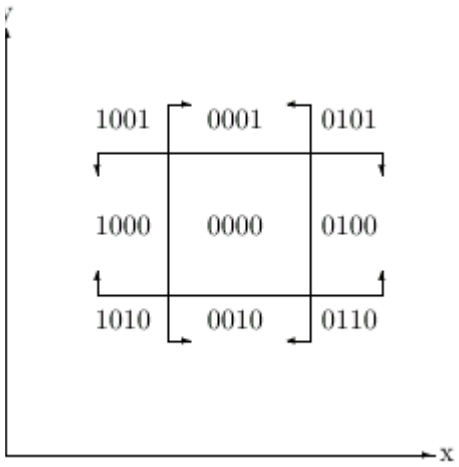

Figura 1.8: Representación de códigos de zonas.

Si la zona de corte que se quiere es un triángulo, se obtendrían 7 en vez de 9 zonas de códigos distintos, y desde luego el número de dígitos que compone cada código es igual al número de planos que conforman el convexo de corte, por lo cual, en este ejemplo un punto $P = (x, y)$ se encuentra dentro de la zona de corte si tiene código 000. Por otro lado, el hecho de tener menos zonas de código equivale a hacer menos comparaciones para cada punto que en el caso de zona de corte rectangular.

Una vez establecido el código para cada zona del caso original, se da a la tarea de mostrar los segmentos de recta que aparecerán en la zona 0000, para lograrlo se vale de la figura 1.9, la cual muestra los puntos extremos de cada línea, que de antemano, se ve que la línea que conforman los puntos G (cod(G) = 0100) y H (cod(H) = 0110) debe ser descartada, una forma de caracterizar dichos puntos es haciendo una operación *and* bit a bit de la siguiente forma:

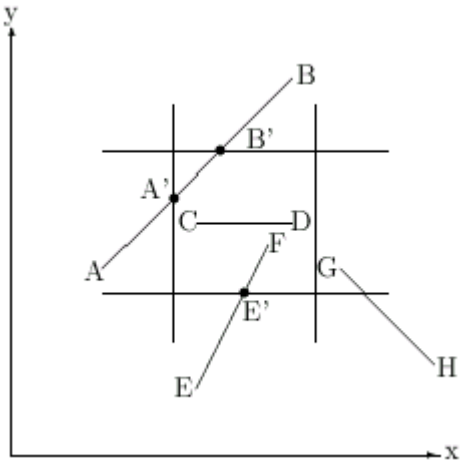

Figura 1.9: Casos más significativos.

$$\begin{array}{c} 0100 \\ 0110 \\ \overline{0100} \\ \Uparrow \end{array}$$

Como 0100 > 0 se descarta.

Al dar 0100≠0 en el *and*, indica que no existe intersección alguna con la línea y los bordes del rectángulo de corte. Ahora, si se considera la línea que se forma por los puntos A (cod(A) = 1000) y B (cod(B) = 0001), gráficamente se observa que no se tiene que desechar toda la línea, sino que se tienen que recortar los segmentos $\overline{AA'}$ y $\overline{BB'}$, para realizar lo anterior se hace una operacin *and*:

$$\begin{array}{r} 1000 \\ 0001 \\ \hline 0000 \end{array}$$

$$\Uparrow$$

Como $0000 = 0$ no se tiene que descartar.

Al dar 0 en el *and*, indica que existe por lo menos una intersección de la línea en cuestión con máximo un borde del rectángulo de corte, y en este caso particular es encontrar las intersecciones de la línea con las rectas $f(y) = x_{min}$ y $f(x) = y_{max}$ por medio de las siguientes ecuaciones paramétricas, considerando que $A = (a_x, a_y)$, $B = (b_x, b_y)$ y con $t \in [0, 1]$:

$$\gamma_x(t) = a_x + t(b_x - a_x) \tag{1.1}$$

$$\gamma_y(t) = a_y + t(b_y - a_y) \tag{1.2}$$

Para la intersección de la recta con la función constante $f(y) = x_{min}$ se utiliza la ecuación 1.1 y se despeja a t_0 de la siguiente manera:

$$a'_x = x_{min} = a_x + t_0(b_x - a_x) \Rightarrow t_0 = \frac{x_{min} - a_x}{b_x - a_x}$$

Por otro lado, si se sustituye t_0 en la ecuación 1.2 se obtiene la componente y del punto $A' = (a'_x, a'_y)$:

$$a'_y = a_y + t_0(b_y - a_y)$$

De forma análoga para la intersección de la recta con la función constante $f(y) = y_{max}$ se utiliza la ecuación 1.2 y se despeja a t_1 como se muestra a continuación:

$$b'_y = y_{max} = a_y + t_1(b_y - a_y) \Rightarrow t_1 = \frac{y_{max} - a_y}{b_y - a_y}$$

Igualmente si se sustituye t_1 en la ecuación 1.2 se obtiene la componente x del punto $B' = (b'_x, b'_y)$:

$$b'_x = a_x + t_1(b_x - a_x)$$

En la línea formada por los puntos E y F se procede de forma similar que a la línea formada por los puntos A y B, pues al hacer el *and* nos da como resultado 0, sólo que en éste caso únicamente hay un punto de intersección que en la figura 1.9 está representado por E', en el caso de la línea formada por C y D se pinta de forma inmediata pues ambos puntos tienen el código 0000. Al aplicar este algoritmo a todas las líneas de la figura 1.9 da como resultado la figura 1.10.

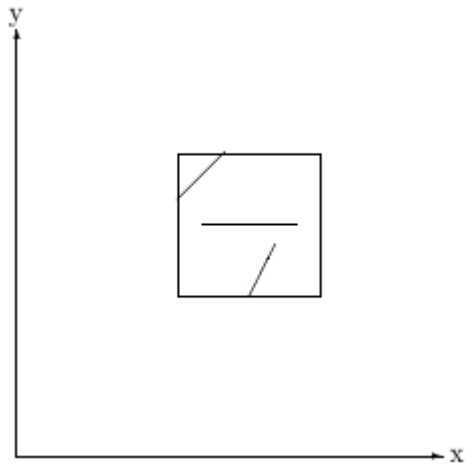

Figura 1.10: Recorte realizado.

1.4. Construcción del sistema de la cámara

Con lo visto anteriormente en las secciones 1.1 y 1.2 se puede obtener un dispositivo generador de escenas. Cuando se quiere tomar una fotografía a una persona con una cámara común se tiene que tomar en cuenta cuál es el punto central de atención a capturar, es decir, si se quiere que únicamente salga la cara, tal vez medio cuerpo o quizá que al revelar el rollo se obtenga una fotografía de cuerpo entero incluyendo un cierto grado de rotación, pues en general, el modelo debe de ser capaz de emular este tipo de propiedades esenciales que dicho aparato posee, sin tomar en consideración los efectos de deformación que puedieran producir las lentes a los objetos. Para lograr este objetivo se observa la figura 1.11, la cual es un buen bosquejo que sirve para la representación del problema, el punto que está marcado con la letra c no es más que el punto central de atención (*look at*) antes mencionado.

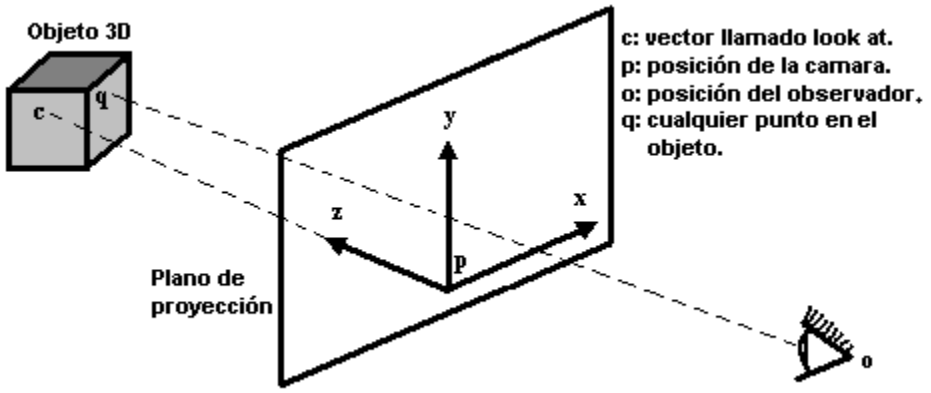

Figura 1.11: Cámara sintética.

Con base en c, es posible orientar la cámara sintética en dirección de dicho punto situándola en el plano imaginario de proyección de la figura, y tomando como origen a p como la posición de la cámara.

Suponiendo que la persona a la que se pretende retratar está de excursión en un bosque y se pretende que de fondo salga la copa de un árbol sin perder atención a la persona. Para conseguir lo anterior únicamente es necesario ponerse de rodillas y colocar la cámara en posición adecuada, por lo cual se tiene que incorporar al análisis de cámara un vector que apunte hacia el cielo (*sky*) de la cámara, mismo que va cambiando dependiendo de la toma que se quiere realizar, en la figura 1.12 dicho vector está representado con la letra s.

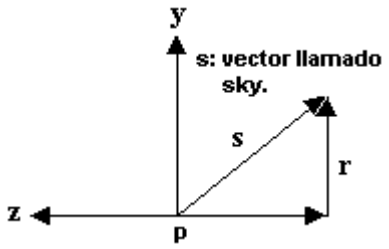

Figura 1.12: Representación del vector s.

Se tienen ya las condiciones para establecer el sistema normalizado coordenado, valiéndose de las figuras 1.11 y 1.12. Tomando a c, la coordenada z queda expresada como:

$$z = \frac{(c-p)}{\|c-p\|} \tag{1.3}$$

Para obtener y es necesario calcular el vector r utilizando la proyección del vector s de la siguiente forma $r = s - (s \cdot z)z$ así, la coordenada y queda expresada en términos del vector r como:

$$y = \frac{r}{\|r\|} \tag{1.4}$$

En el caso de la coordenada x es mucho más sencillo ya que se vale del producto cruz de la siguiente forma:

$$x = z \times y \tag{1.5}$$

Antes de dar el código del sistema de la cámara, se presenta el constructor[2] de la clase[3] $CCamera$ y se considera la existencia de la clase $CVector$ así como de las operaciones elementales, producto cruz, producto punto, normalización, longitud de un vector y multiplicación por escalar. La importancia del constructor radica en que se le pasarán los parámetros requeridos como son: la posición de la cámara, del observador, el punto de atención (*look at*) y si se quiere la foto movida, únicamente se varia el vector *sky*. Con las siguientes líneas que están escritas en $C++$ se logra lo mencionado en este párrafo:

```
CCamera::CCamera(CVector o,CVector p,CVector l_a,CVector s)
{
    CVector position,sky;
    position=p;
    look_at=l_a;
    sky=s;
    obs=o;
```

[2]Un constructor crea instancias de una clase, el nombre de estos coinciden con el de la clase, y jamás devuelven un valor, ni siquiera *void*.

[3]Las clases son estructuras que no sólo contienen una declaración de datos, sino también una declaración de funciones.

```
    System(position,look_at,sky);
}
```

Con lo anterior es posible crear una instancia de la clase $CCamera$, además que dentro del constructor es llamada la función $System$, que como su nombre lo dice, es la que se encarga de obtener el sistema de la cámara sintética. A continuación se presenta el código referente a las ecuaciones 1.3, 1.4 y 1.5 (ejes coordenados) de ésta sección:

```
void CCamera::System(CVector position, CVector look_at, CVector
sky) {
    double a=0;
    CVector temp1,temp2;

    temp1=look_at-position;
    z=VNormaliza(temp1);

    a=sky%z;
    temp1=z*a;
    temp2=temp1-sky;
    y=VNormaliza(temp2);

    x=y*z;
    pos=position;
}
```

1.4.1. Proyección de un punto al plano

De la figura 1.11 $q = (q_x, q_y, q_z)$ representa un punto arbitrario del objeto y el punto $o = (o_x, o_y, o_z)$ representa la ubicación del observador, resta proyectar cada punto q a $\mathbb{R}^2$ por medio de las siguientes ecuaciones paramétricas:

$$x_{proy} = o_x + (q_x - o_x)u \tag{1.6}$$

$$y_{proy} = o_y + (q_y - o_y)u \tag{1.7}$$

$$z_{proy} = o_z + (q_z - o_z)u \tag{1.8}$$

En donde u ∈ [0,1], la proyección que se utiliza es de tipo perspectiva que en general, se entiende como proyección a la transformación de puntos de un sistema de referencia de dimensión n, a un sistema de dimensión $n-1$. Es importante notar que el centro de la proyección perspectiva es el punto o.

Es momento de mencionar lo que será el análogo del revelado del rollo, retomando las ecuaciones 1.6, 1.7 y 1.8 haciendo cero la componente z_{proy} se obtiene que

$$u = -\frac{o_z}{q_z - o_z}$$

y sustituyendo el valor de u en las componentes x_{proy} y y_{proy} se tiene:

$$x_{proy} = \frac{o_x q_z - q_x o_z}{q_z - o_z} \tag{1.9}$$

$$y_{proy} = \frac{o_y q_z - q_y o_z}{q_z - o_z} \tag{1.10}$$

A continuación se presenta el código correspondiente a esta parte:

```
CVector CCamera::projection(CVector q, CVector o)
{
	double a=0;
	CVector temp;

	a=o.z-q.z;
	temp.x=(o.z*q.x-o.x*q.z)/a;
	temp.y=(o.z*q.y-o.y*q.z)/a;
	temp.z=0;
	return temp;
}
```

Capítulo 2

Modelo de Phong

A mediados del siglo XVII se creía que la luz consistía en alguna especie de partículas que emanaban de una fuente luminosa, muchos científicos entre ellos Newton, apoyaban esta teoría corpuscular, por esa época Huygens y otros propusieron que la luz podía ser un fenómeno ondulatorio. En 1887, Heinrich Hertz logró con un circuito oscilante de pequeñas dimensiones, producir ondas electromagnéticas de corta longitud de onda y demostró que poseían todas las propiedades de las ondas luminosas. La teoría electromagnética clásica no podía explicar algunos fenómenos asociados con la emisión y absorsión de la luz. En 1905, Einstein amplió una idea propuesta cinco años antes por Plank, y postuló que la energía de un haz luminoso se hallaba concentrada en paquetes, o fotones. Los fenómenos de la propagación de la luz pueden explicarse mejor mediante la teoría de la onda electromagnética, mientras que la interacción de la luz con la materia, en los procesos de emisión y absorción es un fenómeno corpuscular. El método de Phong está basado en la naturaleza de la luz y la reflexión difusa así como especular, muchos fenómenos ópticos conocidos implican el comportamiento de una onda que incide en la superficie que separa dos medios ópticos, como aire, vidrio o agua.

2.1. Breve descripción del modelo

Bui-Toung Phong crea el sombreado Phong en la Universidad de Utah en 1974, es un método que perfecciona el de Gouraud y que utiliza la interpolación de normales y también realiza un modelo de iluminación bastante utilizado por lo completo que es. El método de Phong apoya los tres tipos

de interacciones material-luz: ambiente, difusa y especular. Si se tiene un conjunto de fuentes de luz puntuales, con componentes independientes para cada uno de los tres colores primarios para cada uno de los tres tipos de interacciones material-luz; entonces, se puede describir la iluminación para una fuente de luz i para cada punto q sobre una superficie. Se construye el método suponiendo que se puede calcular la cantidad de cada una de las luces incidentes reflejada en el punto de interés. Por ejemplo, para el término difuso rojo de la fuente i, el valor de la intensidad del color rojo depende de las propiedades del material, la orientación de la superficie, la dirección de la fuente de luz y la distancia entre la fuente de luz al observador.

2.2. Propiedades de la luz

Lo que se percibe como "luz" o diferentes colores, es una banda de frecuencias dentro del espectro electromagnético. En la figura 2.1 se muestra los intervalos de frecuencia de algunas bandas. Cada valor de frecuencia contenido en la banda visible corresponde a un color distinto, que en el extremo de baja frecuencia hay un color rojo ($4{,}3 \times 10^{14}$ Hertz) y la frecuencia más alta que se puede percibir corresponde al color violeta ($7{,}5 \times 10^{14}$ Hertz). Aproximadamente, el ojo humano puede distinguir cerca de 400,000 colores diferentes, los cuales se pueden describir en términos de la frecuencia, o bien, su longitud de onda.

Tanto el Sol como un foco común, emiten todas las frecuencias contenidas en el intervalo visible para producir la luz blanca que, cuando incide sobre un objeto, algunas de las frecuencias se reflejan y algunas son absorbidas por el objeto. La combinación de frecuencias presentes en la luz reflejada determinan lo que percibimos como el color del objeto, que si predominan las bajas frecuencias, el objeto se describe como rojo. En este caso, se dice que la luz percibida tiene frecuencia dominante.

Además de la frecuencia, cuando se observa una fuente de luz, el ojo humano responde al color y a otras dos sensaciones básicas. Una de éstas llamada brillantez, la cual está relacionada con la intensidad de la luz: cuanto mayor sea la intensidad (o energía), más brillante se ve la fuente. Otra característica es la saturación o pureza, la cual describe cuán puro se ve el color de la luz. Los colores pastel y los pálidos se describen como menos pu-

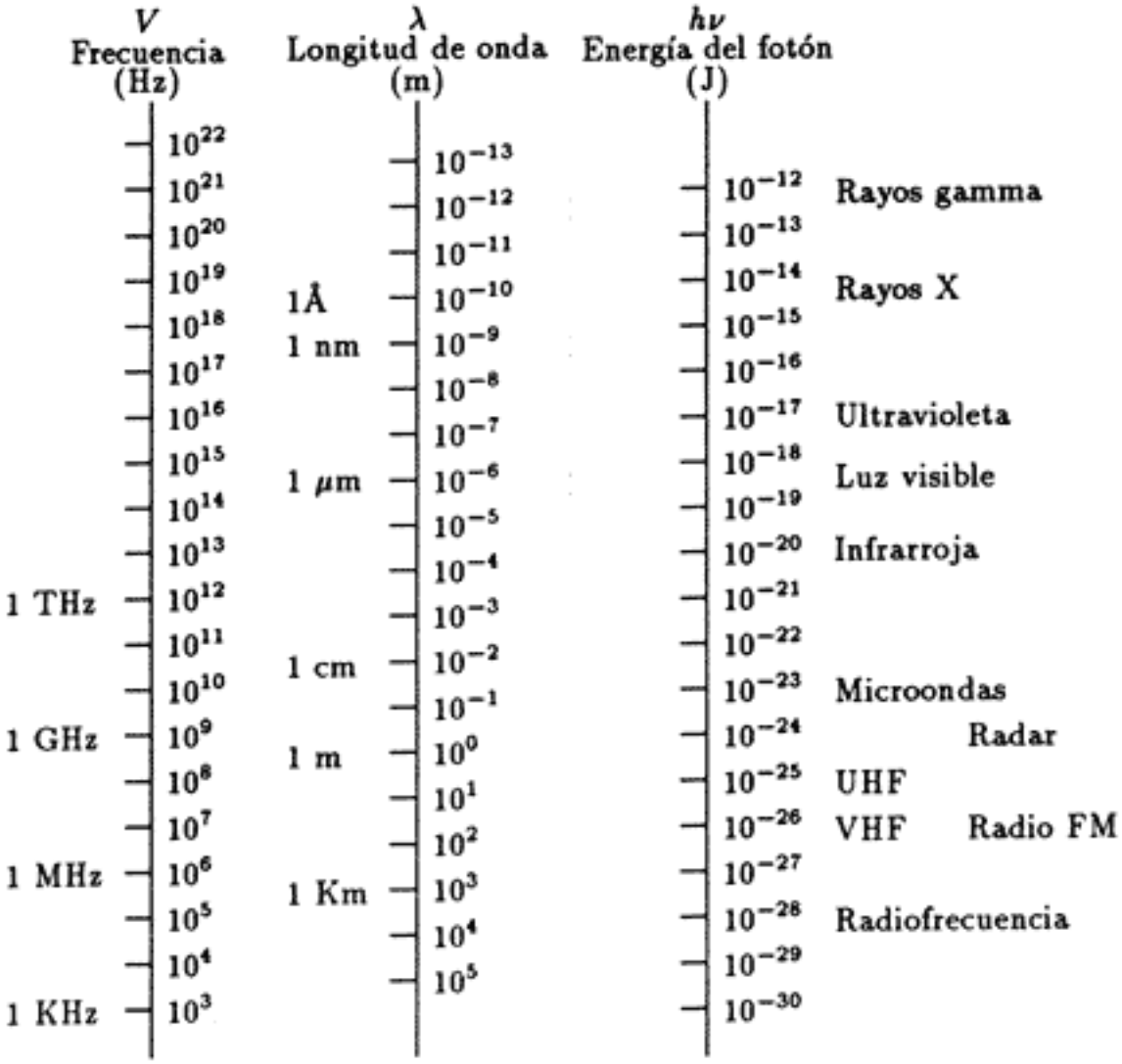

Figura 2.1: Espectro electromagnético.

ros. Estas tres características (frecuencia, brillantez y saturación), se utilizan comúnmente para describir las diferentes propiedades que se perciben en la fuente de luz.

El término "cromaticidad"se utiliza para hacer referencia colectiva de las dos propiedades que describen las características del color, la pureza y la frecuencia.

2.3. Modelo RGB

El ojo percibe el color a través de la simulación de tres pigmentos visuales en los conos de la retina, dichos pigmentos tienen una sensibilidad pico de longitudes de onda de cerca de 630 nm (rojo), 530 nm (verde) y 450 nm (azul), esta teoría de visión de tres estímulos es la base para desplegar la salida del color en el monitor mediante la combinación de los tres colores primarios rojo, verde y azul (de ahí el nombre de RGB).

Se puede representar el modelo RGB con el cubo unitario, como se muestra en la figura 2.2. El origen representa el negro y el vértice con coordenadas $(1, 1, 1)$ el blanco, los vértices sobre los ejes representan los colores primarios y los vértices restantes representan el color complementario de cada uno de los colores primarios.

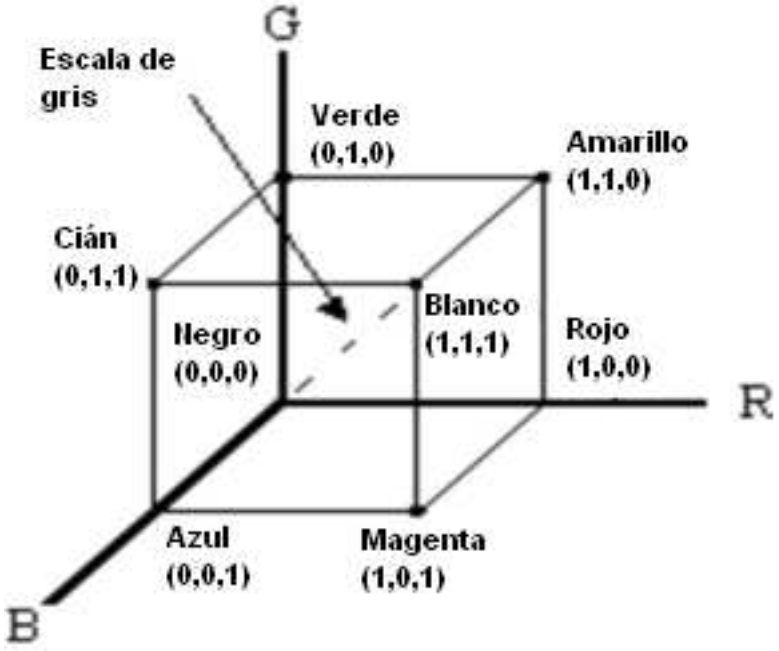

Figura 2.2: Modelo RGB.

El esquema de color es un modelo aditivo: las intensidades de los colores primarios se suman para producir otros colores, cada punto de color contenido en los límites del cubo puede representarse como una triada (R, G, B), donde los los valores de R, G y B se asignan en el intervalo $[0, 1]$, por ejemplo, el color magenta es generado por la suma del rojo con el azul. Por otro lado, cada punto situado en la diagonal tiene una contribución igual de cada color primario, produciéndose así la escala de grises[1].

2.4. Trazo de rayos

Como se sabe, los fotones salen de un emisor (lámpara) y viajan en el espacio hasta chocar con los objetos, cuando esto ocurre, gran parte de aquellos son filtrados por éste, impidiendo el paso de unos y permitiendo el paso a otros o en todo caso escapar, modificando así su trayectoria en la mayoría de los casos. Entonces, cuando el trazo de los rayos se implanta de forma que emule el viaje natural de los fotones, se trata del trazado de rayos hacia

[1]La información de las secciones 2.2 que 2.3 fue obtenida del libro de Hearn [7].

adelante, sin embargo, muchos de estos fotones que salen de la fuente nunca llegan al observador, es decir, que no tienen nada que ver con la construcción de una imagen. Dado este comportamieto, es caro computacionalmente hablando realizar lo anterior, y por tanto, es necesario encontrar una forma alternativa.

Con la ventana definida, si un fotón[2] llega al observador, necesariamente ha tenido que cruzar el plano de proyección siguiendo la dirección del rayo. Entonces, si se trazan exclusivamente los rayos de luz que parten del observador, a través de los píxeles que conforman el plano de proyección (conocidos como rayos primarios) se tiene que asegurar de trabajar únicamente con los rayos útiles, resolviendose así el problema de "hacer de más".

En otras palabras, se coloca un observador dentro del espacio donde se define la escena, asociándole una dirección hacia donde va a ver así como una amplitud visual. A cada una de las direcciones se le asocia un rayo para el cuál se debe calcular la iluminación que viene desde esa dirección.

En la figura 2.3 se muestra la interacción con varios objetos y múltiples fuentes de emisión, en ella se observa el comportamiento de la luz en dirección contraria a su movimiento natural. Se tiene además, que el origen del rayo primario esta en la posición del observador. Desde luego se tiene que tener en cuenta el número de iteraciones que se desea realizar para cada rayo y es posible aplicar un algoritmo recursivo para obtener resultados de maneara más eficiente.

[2]El fotón (del griego: luz) es la partícula mediadora de la interacción electromagnética y la expresión cuántica de la luz. Los fotones son partículas fundamentales, componente de todas las manifestaciones de radiación electromagnética.

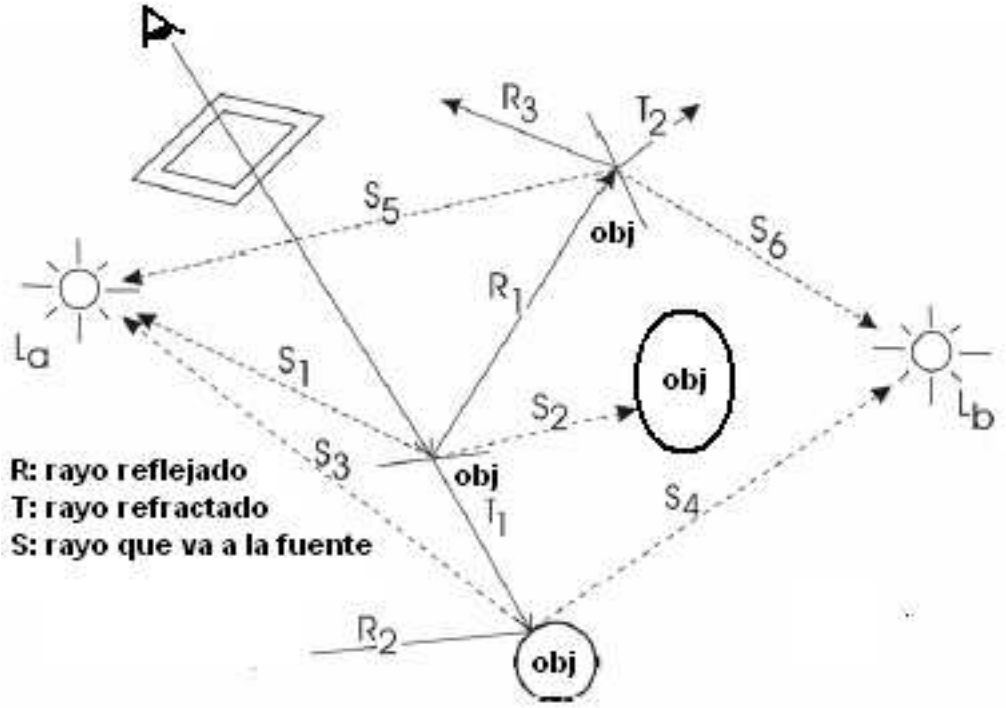

Figura 2.3: Trazo de rayos.

A continuación se presenta el algoritmo de trazo de rayos.

```
Por cada p'ixel de la pantalla (tamano de la imagen)
{
  Rayo r =rayo desde el observador que pasa por dicho p'ixel.
  Color Pixel = Ray_Trace (r);
}

Color Ray_Trace(Rayo r)
{
  Intersecci'on inter_mas_cercana;//Variable de tipo intersecci'on
  Objeto obj;//Variable de tipo objeto.
  Por cada objeto en la escena
  {
```

```
    inter_mas_cercana = Calcular la intersecci'on
    (si es que la hay)con el objeto;
    if(hubo intersecci'on)
       return RTColor(inter_mas_cercana, obj);
    else return negro;
  }
}

Color RTColor(Interseccion i, Objeto obj)
{
  Por cada fuente de luz determinar si ilumina o no el punto
  de la intersecci'on.
  color = color del objeto;

  Si el objeto refleja color_reflejado =
  cte. de reflexi'on * Ray_Trace(rayo reflejado);

  Si el objeto es transparente color_refractado =
  cte. de refraccion * Ray_Trace(rayo refractado);

  return (combina_colores(color,color_reflejado,
                                color_refractado));
}
```

2.5. Fuentes de luz

Cuando se observa un objeto opaco no luminoso, se nota que la luz que se refleja en las superficies del mismo. La luz reflejada total es la suma de las contribuciones de las fuentes de luz y otras superficies reflejantes en la escena (figura 2.3), en ocasiones, las fuentes de luz se conocen como fuentes de emisión de luz, se empleará el término fuente de luz para referirse a un objeto que emite energía radiante, como un foco o el sol.

Para fines prácticos se considera sólo un tipo de emisor denominado fuente puntual, la cual emite rayos de forma radial desde su posición como se observa en la figura 2.4. Cuando la luz incide en una superficie opaca, parte de ésta se refleja y otra parte se absorbe. La cantidad de luz incidente que

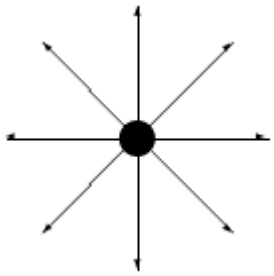

Figura 2.4: Fuente de luz.

una superficie refleja depende del tipo de material, los materiales brillantes reflejan más luz incidente y las superficies opacas la absorben más, de modo similar, ocurre en el caso de una superficie transparente.

Las superficies rugosas o ásperas tienden a dispersar la luz reflejada en todas direcciones, esta iluminación difusa se llama reflexión difusa. Además de la reflexión difusa, las fuentes de luz crean manchas brillantes, que reciben el nombre de reflexión especular. Este efecto de toque de luz es más intenso en superficies brillantes que en superficies opacas.

2.6. Modelos básicos de iluminación

En lo siguinete del capítulo se expondrán modelos empíricos que ofrecen métodos sencillos y rápidos para calcular la intesidad de un punto (en este caso se habla de un píxel) en una superficie determinada y que además producen resultados bastante buenos a modo de dar realismo para la mayor parte de las escenas, los cálculos de la iluminación se basan en las propiedades ópticas de cada superficie.

2.6.1. Luz ambiental

Es posible establecer un nivel general de brillantez para una escena, a fin de producir una iluminación uniforme que se conoce como luz ambiental o luz de fondo, este tipo de luz produce un sombreado plano, la cual no tiene

características de espacio ni de dirección alguna, además de que es constante en cada una de las superficies que se involucran en una escena.

2.6.2. Reflexión difusa

Se comienza diciendo que las reflexión difusa (la producida por luz ambiental) es constante en cada superficie e independiente de la dirección de vista en cada escena. La cantidad fraccional de luz incidente que se refleja de manera difusa se puede establecer para cada superficie con el parámentro k_a, que es el coeficiente de reflexión difusa o de reflexividad difusa.

Al parámetro k_a se asigna un valor constante entre 0 y 1, de acuerdo con las propiedades de reflexión que se quiera dar a la superficie. Si es necesario representar una superficie brillante, lo que se hace es especificar un valor cercano a 1 para k_a, y en caso contrario cercano a 0 para objetos opacos.

Retomando lo anterior en esta sección se expresa la ecuación para el cálculo de la intensidad de reflexión difusa en cualquier punto de una superficie expuesta sólo a la luz ambiental como:

$$I_{ambdif} = k_a I_a \tag{2.1}$$

Una superficie reflectora ideal, es aquella donde la reflexión difusa de la superficie se dispersa en todas direcciones con la misma intensidad y es independiente de la dirección de vista, aunque por otro lado, en este tipo de formas la brillantez depende de la orientación de la fuente, por lo cual, un objeto cuya orientación es perpendicular a la dirección de la luz incidente tiene una apariencia más intensa que si la superficie estuviera en un ángulo oblicuo a la dirección de la luz que llega.

Expresando el ángulo de incidencia entre la dirección de la luz de llegada y la normal a la superficie y utilizando la figura 2.5 (l representa el vector unitario dirigido a la fuente de luz, n la normal unitaria a la superficie y θ el ángulo de incidencia entre l y n) entonces, el área proyectada de un parche de superficie perpendicular a la dirección de la luz es proporcional a $\cos\theta$, y como l y n son unitarios se tiene que $\cos\theta = l \cdot n$, por otro lado si I_l es la

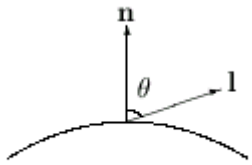

Figura 2.5: Superficie.

intensidad de la fuente de luz puntual, entonces dicha ecuación de reflexión difusa para un punto en una superficie se expresa como:

$$I_{l,dif} = k_d I_l \cos\theta = k_d I_l (n \cdot l) \tag{2.2}$$

De las ecuaciones 2.1 y 2.2, se desprende el modelo de reflexión difusa total (con $k_a, k_d \in [0,1]$) expresada de la siguiente manera:

$$I_{dif} = k_a I_a + k_d I_l (n \cdot l) \tag{2.3}$$

2.6.3. Reflexión especular y el modelo rápido de Phong

En la sección 2.6.2 se ha definido reflexión especular, la cual es el resultado del total, o casi el total de la reflexión de la luz incidente en una región concentrada alrededor de un ángulo de reflexión especular, el cual equivale al ángulo de la luz de incidencia, con los dos ángulos medidos en lados opuestos del vector normal de superficie unitaria n. En la figura 2.6, se utiliza r para representar el vector unitario en la dirección de la reflexión especular ideal; l para expresar el vector unitario dirigido hacia la fuente de luz de punto; y v como el vector unitario que apunta hacia el observador desde la posición de la superficie. El ángulo ϕ es el ángulo de vista con respecto de la dirección de reflexión especular r, así como θ es el ángulo de reflexión especular. Para un reflector ideal, sólo será observada la luz reflejada cuando coinciden los vectores v y r ($\phi = 0$).

Los objetos que no son reflectores ideales presentan reflexiones en un intervalo finito de posiciones de vista en relación con el vector n. Las superficies brillantes tienen un rango reducido de reflexión especular, mientras que las

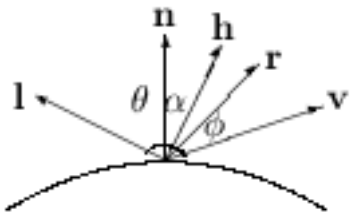

Figura 2.6: Superficie.

superficies opacas, lo contrario. El modelo empírico de Phong, establece que la intensidad de la reflexión especular proporcional a $(\cos\phi)^{n_s}$, con $\phi \in [0, 90])$, donde n_s es el parámetro de reflexión especular, el cual se determina por el tipo de superficie que se quiere modelar, como por ejemplo, para superficies brillantes se modela con un valor alto de n_s (100 o más) y viceversa para las opacas. Cabe mencionar que para un reflector perfecto, n_s es infinito.

La intensidad de la reflexión especular depende de las propiedades del material de la superficie, así como de otros factores como la polarización y el color de la luz incidente, podemos modelar de manera aproximada variaciones de intensidad especular monocromática al utilizar un coeficiente de reflexión especular, W(θ), el cual puede ser sustituido por una constante $k_s \in [0, 1]$ a fines prácticos para lograr la representación de objetos ya sean brillantes u opacos, y así obtener la primer ecuación expresada como:

$$I_{pec} = k_s I_l (\cos\phi)^{n_s} \tag{2.4}$$

Reescribiendo ahora la ecuación 2.4 utilizando desde luego la figura 2.6 a modo de obtener el modelo rápido de Phong, ya que los vectores v y r son unitarios, se tiene que $\cos\phi = v \cdot r$, además, se puede calcular el vector r en términos de los vectores l y n como $r = (2n \cdot l)n - l$, el cual es el vector de reflexión especular como se mencionó anteriormente, por otro lado el vector unitario h se obtiene como $h = (l + v)/\|l + v\|$, el cual, es equidistante entre los vectores l y v, y así, llegar a la ecuación definitiva de intensidad especular:

$$I_{pec} = k_s I_l (n \cdot h)^{n_s} \tag{2.5}$$

2.7. Reflexiones combinadas con fuentes de luz múltiples

Retomando lo anterior de las subsecciones 2.6.1, 2.6.2 y 2.6.3, se obtiene el modelo a partir de las ecuaciones 2.3 y 2.5 para la reflexión de una fuente de luz a partir de un punto en una superficie iluminada como sigue:

$$I = I_{dif} + I_{pec} = k_a I_a + k_d I_l (n \cdot l) + k_s I_l (n \cdot h)^{n_s} \tag{2.6}$$

Para el caso de fuentes de luz múltiples es de forma similar que la ecuación 2.6, y desde luego, considerando que la reflexión difusa ambiental siempre está presente en cada escena que se diseñe, se tendría tan sólo que sumar las contribuciones individuales que producen las fuentes puntuales en cada punto de la superficie iluminada, es decir, de manera más explicita que el modelo de reflexión para fuentes puntuales múltiples está dado de manera muy natural por:

$$I = k_a I_a + \sum_{i=1}^{n} I_{l_i} [k_d (n \cdot l_i) + k_s I_l (n \cdot h_i)^{n_s}] \tag{2.7}$$

2.8. Atenuación de la intensidad

Hasta ahora en el modelo de iluminación no se ha considerado la distancia de la fuente de luz a la superficie, es decir, si dos objetos tuvieran las mismas propiedades ópticas y además uno detras del otro, tan sólo se sobrepondrían y no se distinguirían el uno del otro, que para efecto de realismo resultaría fatal, es por ello que es necesario considerar que conforme la energía radiante desde una fuente de luz a través del espacio, su amplitud se atenúa en un factor de $1/d^2$,donde d es la distancia de la fuente al punto en la superficie en cuestión, pero a modo de reducir errores de variación grandes se utiliza una función de atenuación cuadrática inversa normalizada expresada como:

$$f(d) = min(1, \frac{1}{a_0 + a_1 d + a_2 d^2}) \tag{2.8}$$

Los coeficientes a_0, a_1 y a_2 son coeficientes que se pueden dar al gusto de cada usuario. Tomando en consideración lo tratado en este capítulo se puede decir que ha quedado listo el modelo básico de iluminación en virtud de las ecuaciones 2.7 y 2.8:

$$I = k_a I_a + \sum_{i=1}^{n} f(d_i) I_{l_i} [k_d (n \cdot l_i) + k_s I_l (n \cdot h_i)^{n_s}] \tag{2.9}$$

Donde, desde luego, d_i es la distancia que ha recorrido la luz desde la fuente a un punto en una superficie.

En las figuras 2.7, 2.8 y 2.9, se muestra la aplicación tanto de las secciones anteriores como del modelo de Phong.

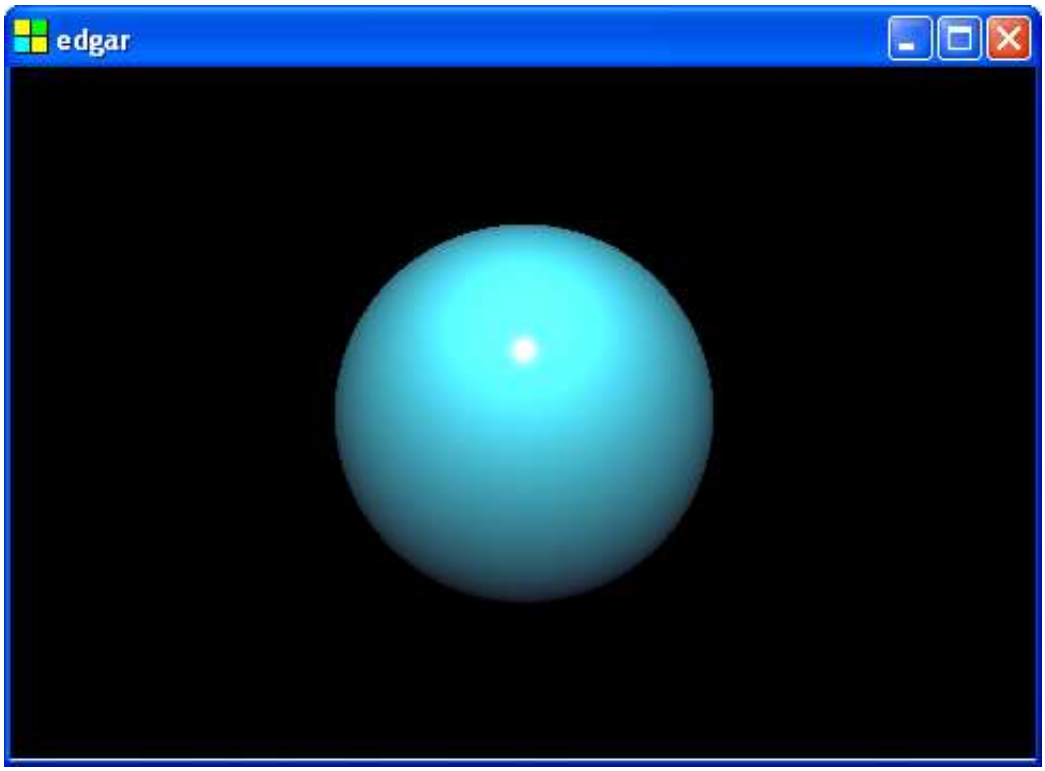

Figura 2.7: Aplicación del modelo de Phong.

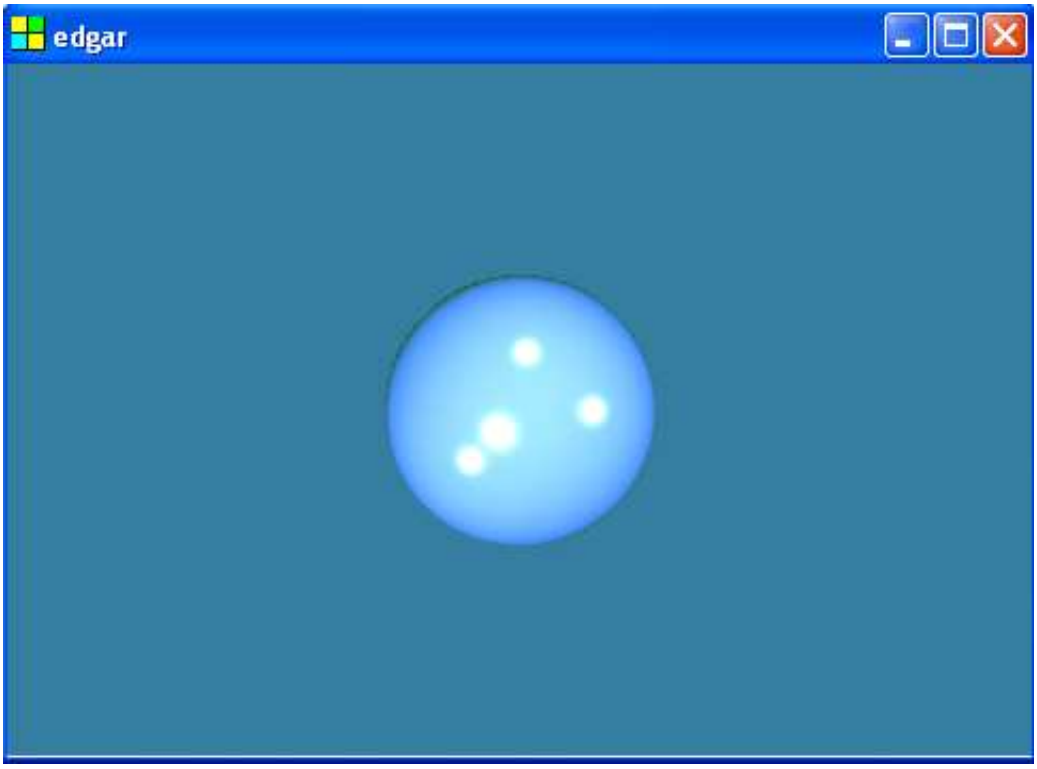

Figura 2.8: Aplicación del modelo de Phong con múltipes fuentes de luz.

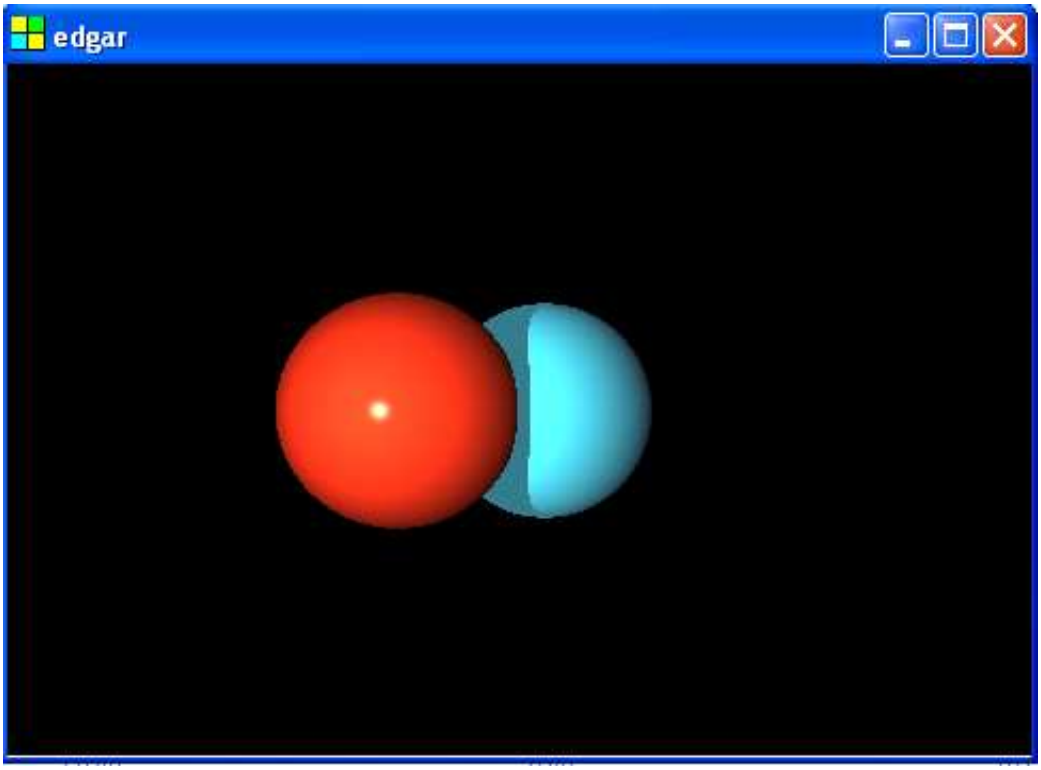

Figura 2.9: Aplicación del modelo de Phong y trazo de rayos para la obtención de sombras.

Capítulo 3

Fractales y texturas

El objeto de este capítulo es graficar ciertas texturas, que sin necesidad del estudio del comportamiento de la luz se logra obtener visualizaciones realistas, que dan la impresión que se utilizan algoritmos sumamente complicados y poco entendibles para personas que no están familiarizadas en este tipo de aplicaciones. Lo que se verá a continuación es que con tan sólo un sencillo método de interpolación, la generación de números aleatorios y fractales se pueden obtener resultados sorprendentes, como por ejemplo: rocas, madera, el fondo de una piscina, y hasta la cáscara de una naranja por mencionar algunas.

3.1. Fractales

3.1.1. Reseña histórica del fractal

Los fractales fueron concebidos por el francés Henri Poincare, sus ideas se extendieron fundamentalmente por dos matemáticos también franceses: Gaston Julia y Pierre Fatou. Se trabajó mucho en este campo, pero el estudio quedó congelado algún tiempo aunque fue renovado a mediados de los 70 por el Dr. Benoit Mandelbrot, de la Universidad de Yale, por sus experimentos de computadora es considerado como el padre de la Geometría Fractal. En honor a él, uno de los conjuntos lleva su nombre. Otros matemáticos de diferentes partes del mundo, como Peano de Italia, Hausdorff de Alemania, Besicovitch desde Rusia entre otros trabajaron también en esta área, sin mencionar investigadores contemporáneos como el Dr. Robert L. Devaney [2], de la Universidad de Boston que ha estado explorando esta rama de la

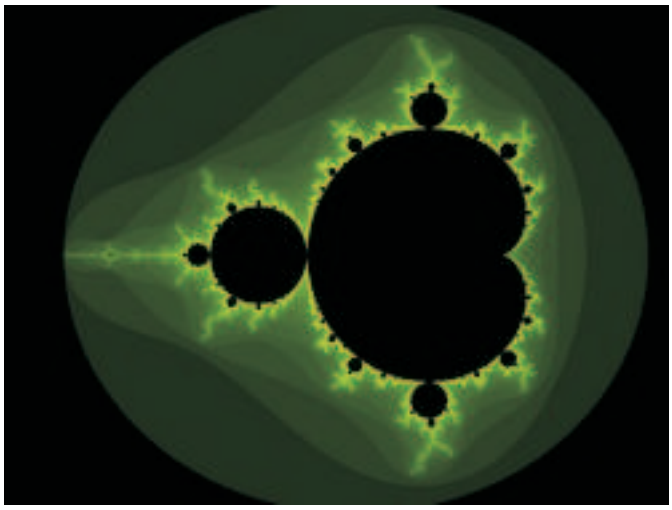

Figura 3.1: Conjunto Mandelbrot.

matemática con la ayuda de las computadoras modernas, como por ejemplo, el conjunto de Mandelbrot que se genera estudiando el comportamiento de la iteración de la ecuación compleja :

$$z_{n+1} = z_n^2 + c \tag{3.1}$$

donde z, c son números complejos, la condición inicial es $z = 0 + 0i$ y a c se le asignan puntos del plano complejo, este conjunto tiene la propiedad de contener copias de sí mismo a todas las escalas, dando como resultado la figura 3.1.

3.1.2. Fractales

Es difícil expresar una noción general de lo que significa "fractal[1]", puesto que muchas de estas descripciones no se pueden aplicar a todas las familias de fractales existentes. Sin embargo, todos los fractales tienen algo en común, ya que todos ellos son el producto de la iteración o recursión (en diferentes escalas), de un proceso geométrico elemental que combinan irregularidad y estructura, la cual da lugar a una forma final de una complicación aparentemente extraordinaria. Es decir, que cada porción del objeto tiene la

[1]Es una palabra derivada del latín *fractus* que significa romper, crear fragmentos irregulares.

información necesaria para reproducirlo todo.

Muchos objetos como costas, copos de nieve, nubes, helechos y montañas se pueden representar por medio de esta parte de la matemática. La principal herramienta de la geometría de fractal es su dimensión. Se está lo suficientemente familiarizado con la idea de que la dimensión de una recta es de orden 1, las superficies son de orden 2 y los volúmenes de orden 3, es menos claro, a propósito, el conjunto de Cantor[2] tiene dimensión 0.631 y la curva de Von Koch[3] tiene dimensión 1.262, este último número es mayor que 1 (pese a tener longitud infinita) y menor que 2 (teniendo área cero), es complicado hablar de dimensiones fraccionarias como fue el caso de los dos últimos ejemplos, ya que cuando se habla de este tema pensamos que el orden de una dimensión es entero.

El siguiente argumento fue tomado del libro de Kenneth [3] mismo que da una interpretación poco primitiva del significado de dimensión de un fractal, indicando como se reflejan las propiedades de escalamiento y similitud a sí mismo.

"Considere la figura 3.2. El segmento de línea está hecho de cuatro copias escalado en un factor de $\frac{1}{4}$, el segmento tiene dimensión $-\frac{log4}{log\frac{1}{4}} = 1$. verde.

Figura 3.2: Razón de $\frac{1}{4}$ para el segmento.

[2]Es en muchos sentidos el más simple de los fractales, el cual, es resultado de dividir en tres partes iguales un segmento de recta y quitar el pedazo de en medio y así se repite el proceso una y otra vez con los segmentos que van quedando en cada iteración.

[3]Se comienza con un triángulo equilátero, que a cada lado de éste, es dividido en tres partes iguales, se quita el trozo medio y en ese espacio se forma un semitriángulo equilátero de lado igual a la longuitud de las partes del lado inicial y nuevamente se repite el proceso en cada lado de los triángulos resultantes.

Por otro lado, el cuadrado de la figura 3.3 está formado por cuatro copias de sí mismo, escaladas en un factor de $\frac{1}{2}$ y su dimensión es de $-\frac{log4}{log\frac{1}{2}} = 2$ como era de suponerse.

Figura 3.3: $\frac{1}{2}$ para el cuadrado.

De la misma forma ocurre con la curva de Von Koch que corresponde a la figura 3.4 que está hecha de cuatro copias escaladas en un factor de $\frac{1}{3}$, y tiene dimensión $-\frac{log4}{log\frac{1}{3}} = \frac{log4}{log3} = 1.262$.

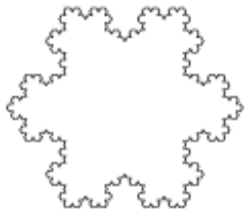

Figura 3.4: $\frac{1}{3}$ para la curva de Von Koch.

En general, si un conjunto se puede construir con m copias de sí mismo escaladas en un factor r, entonces es natural (con base a los argumentos dados en los dos últimos párrafos) pensar que la dimensión de un conjunto *podrá* ser obtenida por la siguiente ecuación:

$$-\frac{log(m)}{log(r)}\text{„}. \tag{3.2}$$

Sin embargo, existe otra forma de definir "dimensión" que es mucho más aplicable para cualquier conjunto. En el libro de Cherbit [1] existe suficiente información, principalmente acerca del tema de "dimensión de Hausdorff". Note que la expresión de la ecuación 3.2 está prácticamente basada en su propia similitud.

3.2. Textura generada por una función

Para esta sección se comenzará con la definición de textura generada por una función; que son imágenes que no están almacenadas en el disco duro

como archivos de bits (bmp, jpg, ...) sino que se realizan por medio de un procedimiento o por el algoritmo que es capaz de generarlo, una de las ventajas de trabajar con este tipo de texturas, es que, no ocupan demasiado espacio y en otros términos de lenguaje son "densas", es decir, se puede determinar el color en cualquier punto de la pantalla.

Las texturas generadas por una función están compuestas de mapas que son texturas sencillas (por lo general en blanco y negro) que se fusionan con otros, produciendo texturas complejas (un ejemplo es el mapa del fondo de una piscina). Algunos tipos de mapas son los geométricos y aleatorios, estos últimos son mucho más complicados de sintetizar y también requieren más ciclos de proceso, sin embargo, hay algunos algoritmos de interés, como lo son los algoritmos del mapa Celular y del Perlin Noise.

3.2.1. Función IntNoise

Las texturas más bonitas y complicadas tienen factores aleatorios, sin embargo, un problema inicial es la obtención de estos números, no puede usarse una función aleatoria basada en el reloj del sistema, pues la textura cambiaría de un cuadro a otro. Con IntNoise por otro lado, no se tienen estas limitaciones, ya que con seguridad al momento de consultar colores de la textura en cierto punto, seguirán siendo los mismos, además como función aleatoria no es continua e IntNoise(x) no tiene por qué parecerse a IntNoise(x+1). Otra propiedad con la que cuenta esta función, es que, si se tiene un grupo de píxeles que han de compartir un mismo número aleatorio, todos éstos involucran el mismo argumento. A continuación se presenta la implementación de esta función:

```
double IntNoise(int x)
{
    x=(x<<13)^x;
    return (((x * (x * x * 15731 + 789221) + 1376312589)
                          & 0x7fffffff) / 2147483648.0);
}
```

Esta función regresa valores de punto flotante en [0,1], y el código no sólo es un generador de números, sino también se pueden hacer diferentes generadores cambiando y dejando de igual tamaño los números primos que aparecen en la implementación.

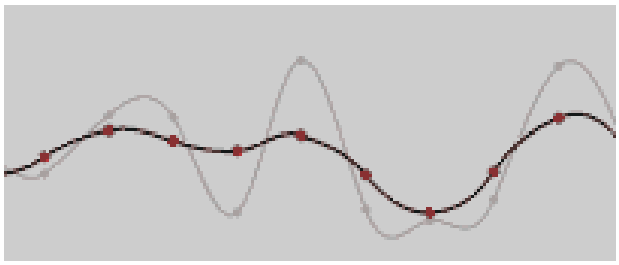

Figura 3.5: Suavizado.

3.2.2. Interpolación

La función IntNoise previamente analizada, es discontinua y toma a un entero como parámetro, pero, ¿qué sucedería si se quisiera generar un número al azar para x=1.2? la respuesta se encuentra en la interpolación, ya que con esta herramienta se puede definir de manera continua una función. Existen varios tipos de interpolación, como la lineal, cosenoidal, cúbica, entre otras, las diferencias en cada una de ellas varian en rapidez computacional así como en resultados visuales. A continuación se presenta la función que se utiliza, la cual es de tipo polinomial a saber $f := 3x^2 - 2x^3$, en el artículo de Perlin [10] presenta éste mismo polinomio para la obtención de la siguiente función de ruido, es claro que cuando se tiene como parámetro 0 la función devuelve a, si se evía 1 la función regresa b:

```
double InterPolacion(double a,double b,double x)
{
    return a+(b-a)*x*x*(3-2*x);
}
```

3.2.3. Smooth Noise

Debido a que el mapa Celular presenta cierto tipo de coherencia, es decir, el color de píxeles contiguos no cambia tan abruptamente, es posible suavizar la salida de la función IntNoise, para que así se tenga un resultado "menos aleatorio". La figura 3.5 ayuda a tener una mejor idea; dicho suavizado simplemente toma un valor devuelto de la función IntNoise y saca el promedio del valor, junto con valores adjuntos a él. El siguiente código es la implementación:

```
double SmoothNoise1D(int x)
{
    return (IntNoise(x)/2+IntNoise(x-1)/4+IntNoise(x+1)/4);
}
```

3.2.4. Mapa Perlin Noise

El mapa Perlin Noise es resultado de las secciones 3.2.1, 3.2.2 y 3.2.3 extendidos a dos dimensiones. A continuación se presenta la manera de usar la interpolación bidimensional ya sea lineal, cosenoidal o polinómica (dependiendo con la que se trabaje) para interpolar el punto en un cuadrado (2D):

```
a=InterPol(Punto(0,0),Punto(0,1),frac_x);
b=InterPol(Punto(1,0),Punto(1,1),frac_x);
final=InterPol(a,b,frac_y);
```

Los puntos (n_i, n_j) indican los valores aleatorios de los vértices del cuadrado en el que se encuentra, `frac_x` es 0 cuando está pegado al lado izquierdo del cuadrado, y 1 en el derecho. Se dice que `frac_y` es 0 cuando está arriba del cuadrado, y 1 cuando está abajo. Para tener una mejor idea considere la figura 3.6, en ella se encuentra la razón de por qué hace falta llevar a cabo tres procesos, en esta interpolación el plano base es la imagen, y la altura es el valor que da la función IntNoise. La interpretación geométrica se da de la siguiente manera: La rejilla gruesa indica los cuadrados en los que ha sido partida la imagen (periodo), la rejilla fina son los píxeles y en este caso se puede ver que el periodo es de 16, por tanto cada celda o cuadrado tiene 16^2 píxeles. Las 4 barras verticales amarillas son los valores de IntNoise para cada vértice, las 2 barras verdes apoyadas en el plano indican los valores aproximados de `frac_x` y `frac_y`, se puede ver como en este caso `frac_x=` $\frac{4}{16}$ y `frac_y`$=\frac{4}{16}$ aproximadamente. El punto de intersección es el píxel del cual se está calculando IntNoise2D(x,y). Las curvas rojas son las curvas del polinomio de interpolación, y los puntos verdes que hay en cada una de ellas están a la altura indicada por las variables a y b. Finalmente la curva azul es el polinomio de la última instrucción, el punto verde está a la altura del valor final.

El código de las siguientes funciones se extiende de una a dos dimensiones, tanto de IntNoise, SmoothNoise1D, así como la interpolación de un punto en un cuadrado (representado en la figura 3.6):

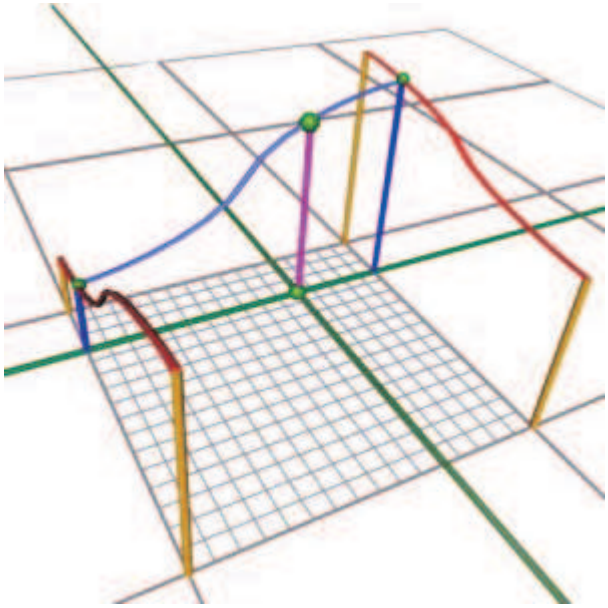

Figura 3.6: Interpolación 2D.

```
double IntNoise2D(double x,double y)
{
    int n;

    n=int(x+y*57);
    n=(n<<13)^n;
    return (((x * (x * x * 15731 + 789221) + 1376312589)
                          & 0x7fffffff) / 2147483648.0);
}
double SmoothNoise2D(double x,double y)
{
    double corners,sides,center;

    corners=(IntNoise2D(x-1,y-1)+IntNoise2D(x+1,y-1)+
             IntNoise2D(x-1,y+1)+IntNoise2D(x+1,y+1))/16;
    sides=(IntNoise2D(x-1,y)+IntNoise2D(x+1,y)+
             IntNoise2D(x,y-1)+IntNoise2D(x,y+1))/8;
    center=IntNoise2D(x,y)/4;
    return (corners+sides+center);
```

```
}
double InterpolatedNoise2D(double x,double y)
{
    int pasox,pasoy;
    double frac_x,frac_y,v1,v2,v3,v4,i1,i2;

    pasox=int(x);
    frac_x=x-pasox;

    pasoy=int(y);
    frac_y=y-pasoy;

    v1=SmoothNoise2D(pasox,pasoy);
    v2=SmoothNoise2D(pasox+1,pasoy);
    v3=SmoothNoise2D(pasox,pasoy+1);
    v4=SmoothNoise2D(pasox+1,pasoy+1);

    i1=InterPolacion(v1,v2,frac_x);
    i2=InterPolacion(v3,v4,frac_x);

    return (InterPolacion(i1,i2,frac_y));
}
```

Antes de escribir la función Perlin Noise, es necesario definir lo que es una onda, ya que este mapa especial es basado en el principio de superposición de onda así como en la generación de números aleatorios. Una onda es cualquier perturbación de una condición de equilibrio que se mueve o se propaga en el tiempo de una región a otra en el espacio clasificándose en longitudinales y transversales. Muchas de las propiedades observadas de la luz se explican mejor por medio de una teoría ondulatoria. Las ondas luminosas son fundamentalmente de la misma naturaleza que las ondas de radio, las infrarrojas, ultravioletas, los rayos X y los rayos gamma. Las ondas transversales se dan cuando el movimiento de las partículas son perpendiculares a la dirección de propagación de la onda, las ondas longitudinales son aquellas en las cuales las partículas se mueven hacia atrás, hacia adelante, y a lo largo de la dirección de propagación[4]. En la figura 3.7, se representan las partes de una onda, la amplitud es el desplazamiento máximo de la curva, la longitud de onda λ

[4]En los libros Sears [11] y Resnick [13] se explica de manera más extensa este tema.

Figura 3.7: Figura afica de una onda.

es la distancia entre dos puntos contiguos de la onda, que tengan la misma fase, el periodo τ, es el tiempo necesario para que un punto en determinada coordenada x cumpla un ciclo completo de movimiento y frecuencia f es el recíproco de el periodo.

Por otra parte el principio de superposición de onda se da cuando varias ondas se combinan en un punto, el desplazamiento de una partícula cualquiera en determinado momento, es simplemente la suma de los desplazamientos que podrían producir las ondas de manera individual.

La función Perlin Noise se define como un acopio (superposición de ondas) de muchas funciones suavizadas, con diferentes frecuencias y amplitudes y es además una función de ruido, para simplificar un poco más y evitar demasiados cálculos en cada paso del ciclo donde se superponen, es usado un valor llamado *persistencey* que especifica la amplitud para cada frecuencia dada (en este la frecuencia en cada paso es una potencia de dos). A cada una de estas funciones sumadas se le denomina octava (*octave*), debido a que la frecuencia en cada paso es el doble que de la anterior, en música las octavas tienen la misma propiedad, se pueden sumar tantas octavas como se quiera.

El código siguiente no es más que el mapa Perlin Noise, donde son llamadas la funciones de las secciones anteriores, la frecuencia es 2^i y la amplitud es $(persistencey)^i$ donde i es la i-ésima función sumada, y por último, es usado un corrimiento (*seed*) en cada una de las coordenadas x y y para obtener mejores resultados.

```
double PerlinNoise2D(double x,double y,int octaves,int seed)
{
    int n,i,frequency;
    double persistencey,total,amplitude;

    total=0;
    persistencey=2;
    n=octaves-1;

    for(i=0;i <= n;i++)
    {
        frequency=2^i;
        amplitude=pow(persistencey,double(i));
        total=total+InterpolatedNoise2D(x*frequency+seed,
                              y*frequency+seed)*amplitude;
    }
    return total;
}
```

La figura 3.8 es el resultado de la aplicación de este algoritmo con 6 octavas y corrimiento igual a 8800, es importante resaltar que se utiliza la interpolación lineal en la escala de grises para el coloreado, esta técnica será desarrollada con más precisión en la sección 3.2.6.

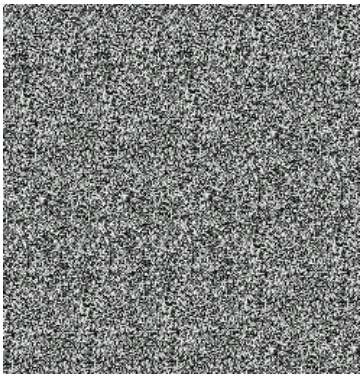

Figura 3.8: Perlin Noise 2D.

Se observa una apariencia aspera de tipo rocosa, por otro lado, si no se hubiera coloreado la respuesta de esta función sería como el de una “televisión sin antena”.

3.2.5. Mapa celular

Este mapa Celular cae dentro de las texturas generadas por una función, creado por Steven Worley, dicho algoritmo es útil para texturas con apariencia de células, mosaicos, escamas, etc. Se puede decir que el algoritmo tiene cuatro fases las cuales son:

1. Distribución aleatoria de puntos: Se marcan las coordenadas de los píxeles donde se considera que hay un punto, cada uno de estos puntos será el pseudo-centro de las células o mosaicos.

2. Cálculo de distancias: Para cada píxel del mapa, se calcula la distancia a cada punto (se pueden utilizar diferentes distancias como la euclidiana, cuadrada, etc).

3. Selección de las distancias menores: Se seleccionará la distancia del punto más cercano, dos o tres más próximos.

4. Cálculo de valores finales con estas distancias: El mapa celular permite jugar con los 3 ó 4 valores de las distancias menores para conseguir buenos efectos. Dependiendo qué fórmula sea utilizada para conseguir el valor final a partir de los valores primero, segundo y tercero así como un factor de corrección si fuera necesario, (tam_cas en píxeles) de esta manera se obtiene una gran variedad de resultados.

Siguiendo con la primera fase del algoritmo (distribución aleatoria de puntos), nos encontramos con dos situaciones:

En la primera fase es que, como se dijo anteriormente, una textura generada por una función es "infinita", y se necesitan coordenadas aleatorias de "infinitos" puntos en el plano, y aquí es donde se requiere a la función IntNoise.

La segunda, no es posible calcular distancias muy grandes entre puntos, ya que la precisión de una computadora es finita (teóricamente sí es posible), entonces se recurre a encerrar cada punto en una casilla, de tal manera que haya un y sólo un punto en cada casilla. Esto deberá producir un mapa de aspecto algo más regular.

En la figura 3.9 se representa el encasillamiento, quedando los puntos como se muestran, cada uno en su celda. El punto rojo indica el píxel que se está calculando. De esta manera se restringe a 9 (en la figura 3.9) el número de puntos a los que se tiene que calcular la distancia: El punto que hay en la celda en cuestión, y los puntos de las 8 celdas contiguas. Exactamente los puntos que hay dentro del rectángulo verde.

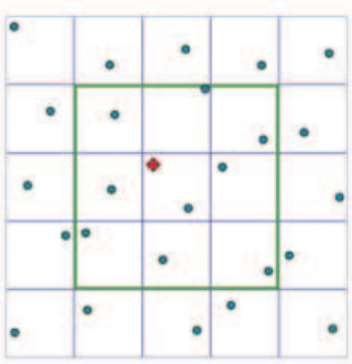

Figura 3.9: Puntos.

En la figura 3.10, se tienen circunstancias extremas, si la suerte jugara una mala pasada y la distribución de los puntos en una zona de la textura es especialmente inconveniente, podría ocurrir que el punto más cercano al píxel a estudiar no estuviera dentro del rectángulo verde de los 9 puntos.

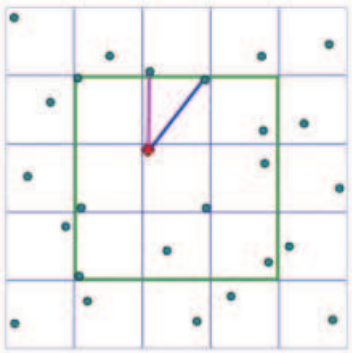

Figura 3.10: Puntos.

Se aprecia en la figura que el segmento rosa es más corto que el morado, así que la distancia del punto más cercano al píxel rojo, está fuera del cuadrado. Esto haría que la simplificación no fuera 100 % válida, provocando así en ciertos momentos errores en la textura.

La solución a este problema (se representa en la figura 3.11) está en limitar aún más la libertad de los puntos, haciendo que sólo puedan estar en un cuadrado interior a cada celda, centrado en medio, y que diste en un valor pequeño de la pared de la celda, siendo 1 el tamaño de la celda entera (por comodidad).

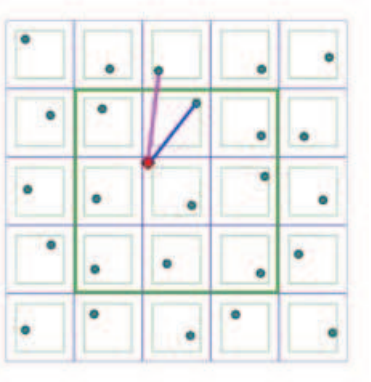

Figura 3.11: Puntos.

Esta solución es provisional, pues si bien garantiza que el punto más cercano estará en el gran rectángulo verde, probablemente no suceda lo mismo con el segundo o el tercero más cercano, sin embargo, para conseguir esto, habría que limitar aún más los puntos, siendo entonces apreciable la uniformidad de la textura. Además, ahora sí es realmente improbable que ocurra el error con el segundo o tercer punto más cercano. Por estas dos razones no se restringirá aún más a los puntos.

A continuación se presenta el código referente a esta parte, en donde se hacen notar las cuatro fases:

```
double Cellular(double x,double y,int width,int tam_cas
                                                ,int seed)
{ //Primera fase
------------------------------------------------------------
    double primero=2*tam_cas;
    double segundo=2*tam_cas;
    double tercero=2*tam_cas;
    double dist_aux,xpunto,ypunto;
    int casilla_pto;
    int n_casillas=int(width/tam_cas)+1;
    int casillax=int(x/tam_cas);
    int casillay=int(y/tam_cas);
    int casilla=n_casillas*casillay+casillax;
    for (int j=-1;j<2;j++)
```

```
    {
        for (int i=-1;i<2;i++)
        {
            casilla_pto=casilla+i+j*n_casillas;
            xpunto=(casillax+i)*tam_cas+Noise(casilla_pto+seed)
                                              *tam_cas;
            ypunto=(casillay+j)*tam_cas+Noise(casilla_pto+10+seed)
                                              *tam_cas;
//Segunda fase
-------------------------------------------------------------------
            dist_aux=sqrt((x-xpunto)*(x-xpunto)+(y-ypunto)
                                              *(y-ypunto));
//tercera fase
-------------------------------------------------------------------
            if (primero>dist_aux)
            {
                tercero=segundo;
                segundo=primero;
                primero=dist_aux;
            }
            else
            {
                if (segundo>dist_aux)
                {
                    tercero= segundo;
                    segundo=dist_aux;
                }
                else
                {
                    if (tercero>dist_aux)
                    {
                        tercero=dist_aux;
                    }
                }
            }
        }
    }
//Cuarta fase
```

```
-----------------------------------------------------------------
    return (2*primero/(segundo+tercero));
}
```

En la figura 3.12, se presentan resultados con diferentes valores de retorno, los cuales dan alusión a rocas en algunos de ellos y otros dan la apariencia de células.

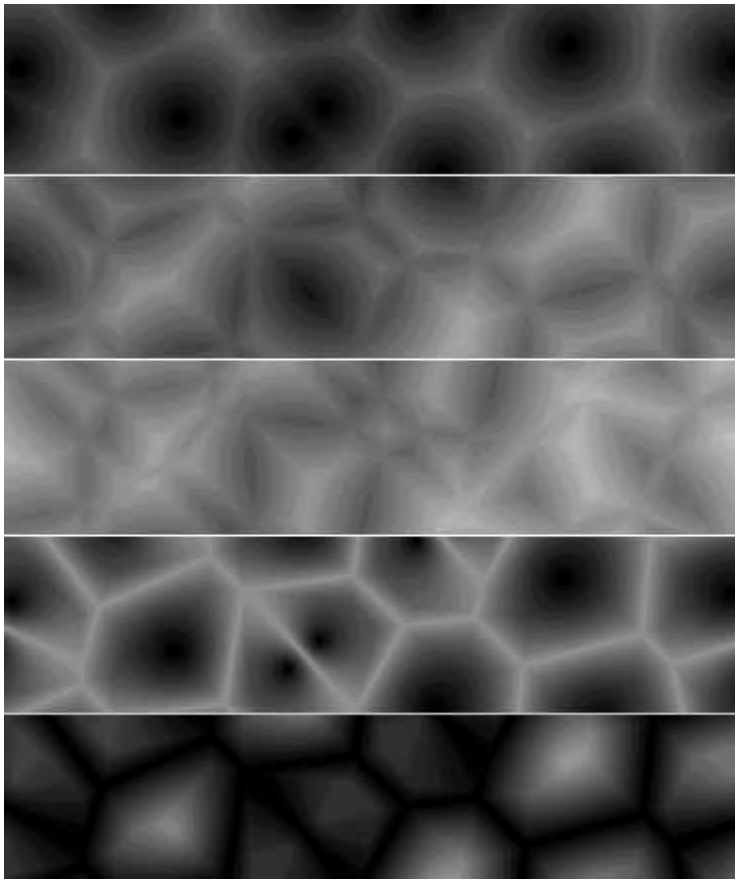

Figura 3.12: Mapa celular.

3.2.6. Combinación de los mapas Perlin Noise y celular

Una vez generados los mapas con algoritmos capaces de producir imágenes sin ningún color, se observará en esta sección como colorear un mapa, para obtener una excelente visualización de "el fondo de una piscina", que en síntesis, son como pequeñas parcelas como el Celular en las que se iluminan los bordes y, especialmente, las esquinas con una enorme deformación. Para ello es muy fácil, imaginando a los mapas como hojas de acetato y para cada punto el color que se represente sea la media de los valores que ambos mapas dan para ese punto, y también es posible hacer que uno tenga más énfasis que otro (como si uno de los papeles fuera menos transparente que el otro) dándole más peso a uno de ellos.

Otra forma sería enviar un mapa como parámetro de otro, que por lo general una función de un mapa de una textura generada por una función se compone básicamente de dos parámetros, la coordenada x y la coordenada y, se pueden programar dichas funciones para que posean parámetros auxiliares y así obtener efectos diversos, como octavas, tamaño de las celdas, grosor de las líneas, rugosidad de cierto material.

Cuando éstos son combinandos, son utilizados dos valores de mapas distintos para sumar cada uno a la coordenada x y a la y del mapa que se quiere desplazar, de esta forma se tiene que el efecto de deformación no sea unidimensional. Así que cuando el primer mapa tenga colores claros, el mapa sobre el que se aplica el efecto se desplazará hacia la izquierda, y cuando tenga valores oscuros hacia la derecha. Análogamente, cuando el segundo mapa tenga valores claros se desplazará hacia arriba, y cuando sean oscuros hacia abajo, en este trabajo se desplaza la textura Celular con un mapa Perlin Noise. Desafortunadamente en la realidad, el proceso de composición de una textura es en gran medida un proceso de ensayo y error.

Es el momento de darle color a los mapas para hacer texturas, para ello existen diversas maneras, una de ellas es hacer mapas específicos para cada canal RGB, o realizar modificaciones en distintos canales. Una forma más sencilla es aplicar un degradado (como los del *Photoshop*) que devuelva un color RGB para cada nivel en la escala de grises de los mapas. Por otro lado, para darle color a los mapas se comienza haciendo nuevamente uso de la interpolación lineal, y puesto que en este caso sólo se obtendrán dos valores

conocidos como el color inicial y el final, no es necesario hacer una interpolación polinómica.

Se tienen que conocer las coordenadas en el espacio RGB de los colores que se pretenden interpolar, es decir, saber que cantidad de rojo, verde y azul tienen ambos colores. Para este caso se utilizarán los colores del fondo del agua, que en esencia son diferentes tonalidades de azul, la función color_agua ha de devolver el color $RGB(40, 100, 162)$ si se le pasa como parámetro el valor 0 (máxima oscuridad), y el color $RGB(91, 187, 201)$ si se le pasa el valor 1 (punto de máxima iluminación por la luz). Lo que ha de hacer por tanto es hacer una interpolación lineal es devolver un valor entre cada componente RGB, para así obtener diferentes tonalidades de azul. El siguiente código es la implementación de lo aquí dicho, tanto de la función color_agua y de InterLin (interpolación lineal).

```
double InterLin(double a,double b,double x)
{return a+(b-a)*x;}

int color_agua (double valor)
{
    int r=int(InterLin(40,91,valor));
    int g=int(InterLin(100,187,valor));
    int b=int(InterLin(162,201,valor));
    return RGB(r,g,b);
}
```

La función Pintar se encarga de graficar "el fondo de una piscina", se emplean tanto PerlinNoise2D, Celular y color_agua, el cual manda un mapa como parámetro de otro, y para ser más claro se manda PerlinNoise2D a Celular. Por otra parte, se tiene que la iluminación en el agua tiene un poco de autosimilitud como ocurre en ciertos tipos fractales, así que es utilizado dos veces el mapa Celular, una con un *tam_cas* de 64 y otra de 48, además se hace más énfasis al primer mapa. Se tiene que, tanto *color*1 y *color*2 multiplican el valor de ambos mapas por sí mismo, puesto que ambos están en el intervalo [0,1], el efecto será que los números pequeños lo sean aún más, mientras que los valores altos en [0,1] permanecerán más o menos estables y de esta manera se consigue hacer que la luz parezca más intensa donde da, más contrastada y más escasa donde no llega.

```
void Pintar(CPaintDC *dc)
{
    int x,y;
    double color1,color2,dispx,dispy;

    color1=0;
    color2=0;
    dispx=0;
    dispy=0;

    for(y=0;y < CuadradosY;y++)
    {
        for(x=0;x < CuadradosX; x++)
        {
             dispx= PerlinNoise2D(x,y,3,65);
             dispy= PerlinNoise2D(x,y+555,3,65);

             color1 = Cellular(x*0.6+dispx,y+dispy,5,64,15000);
             color1 *= color1;

             color2 = Cellular(x*0.6+dispy,y+dispx+2312,17,48,8000);
             color2 *= color2;

             color1=(color1*0.30+color2*0.70);
             color1=color_agua(color1);
             dc->SetPixel(x,y,unsigned long (color1));
        }
    }
}
```

Finalmente la concatenación de las secciones anteriores de este capítulo se aprecia en la figura 3.13, la cual es muy realista a simple vista.

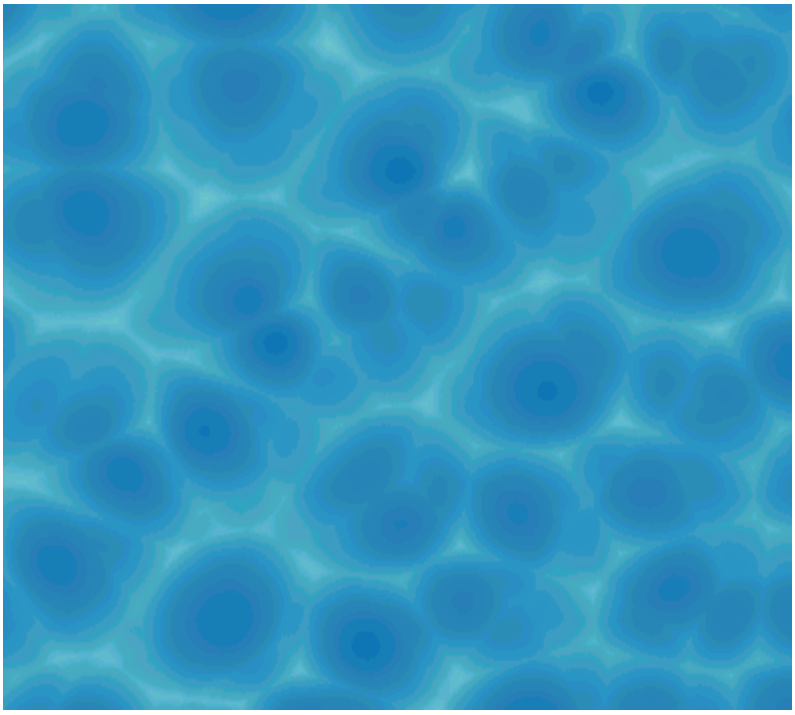

Figura 3.13: Fondo de una piscina.

Conclusiones y posibles extensiones

El método de Phong es un excelente algoritmo de iluminación, desafortunadamente el costo computacional es elevado cuando dentro de una escena se encuentran varios objetos; por otro lado, con el avance de la tecnología será posible en algún tiempo realizar escenas que involucren un basto número de objetos quizá en tiempo real o en un tiempo relativamente corto como lo hace *Open GL*, el cual es un sistema de graficación en tiempo real, aunque, como todo, tiene su parte negativa, en particular, las escenas que se forman son de menor realismo, como ejemplo considere la siguiente figura:

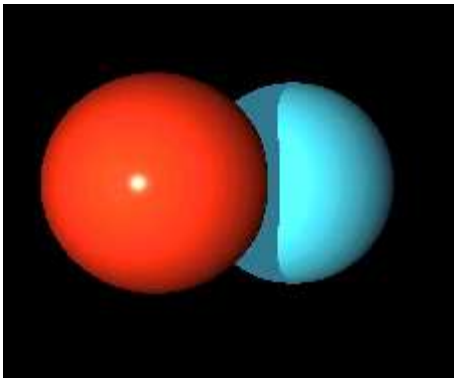

Figura 3.14: Con el método de Phong.

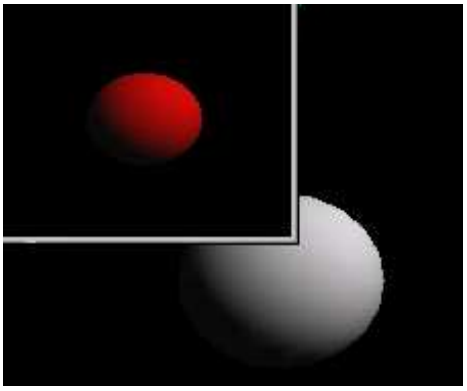

Figura 3.15: Con Open GL.

La sección 1.3 de recorte es de suma importancia debido a que un objeto tridimensional puede ser aproximado por triángulos, por ejemplo, con siete zonas de corte cada uno de ellos, para así poder aplicar el algoritmo de

Phong a cada uno de ellos que conforman dicho objeto y lograr resultados de calidad visual e incluso dar un cierto grado de imperfección en la superficie utilizando funciones de ruido pero aplicadas en la normal de cada punto y así obtener otro tipos de efectos.

Como se vio en el capítulo 3 es posible obtener buenos resultados sin necesidad del estudio del comportamiento de la luz, una aplicación casi natural de los mapas sería la creación de funciones capaces de mapear el color correspondiente de cada punto (x, y) del mapa a un objeto en dimensión tres.

Otra posible extensión del capítulo 1 sería emular las funciones de las lentes de una cámara común para lograr así, diferentes tipos de efectos como lo hace *Povray*. La única limitación de la aplicación del método de Phong es la imaginación de cada persona interesada en él.

Bibliografía

[1] Cherbit G., *Fractals Non-integral Dimensions and Applications*, John Wiley & Sons, 1991.

[2] Devaney Robert, *Chaos, Fractals and Dynamics Computer Experiments in Mathematics*, Addison-Wesley, 1990.

[3] Falconer Kenneth, *Fractal Geometry Mathematical Foundations and Applications*, John Wiley & Sons, 1990.

[4] Foley James, *Computer Graphics Problems and Practice*, Addison Wesley, 1997.

[5] Foley James, *Introduction to Computer Graphics*, Addison Wesley, 1990.

[6] Glassner Andrew , *An introduction to Ray Tracing*, Academic Press Limited, 1990.

[7] Hearn Donald, *Gráficas por Computadora*, Prentice Hall, 1988.

[8] Perez Virginia, *El Gran Libro De Visual C++*, Marcombo, 1997.

[9] Perlin Ken, *An Image Synthesizer*, SIGGRAPH: ACM Special Interest Group on Computer Graphics, volumen 19, número 3, 1985.

[10] Perlin Ken, *Hypertexture*, SIGGRAPH: ACM Special Interest Group on Computer Graphics, volumen 23, número 3, 1989.

[11] Sears Francis, *Física Universitaria*, Addison Wesley, 1981.

[12] Spiegel Murray, *Teoría y Problemas de Análisis Vectorial y una Introducción al Análisis Tensorial*, McGraw-Hill, 1981.

[13] Resnick Robert, *Física*, Compañia Editorial Continental, 2006.

Índice alfabético