Kapitel 23

Ray Tracing

23.1 Grundlagen

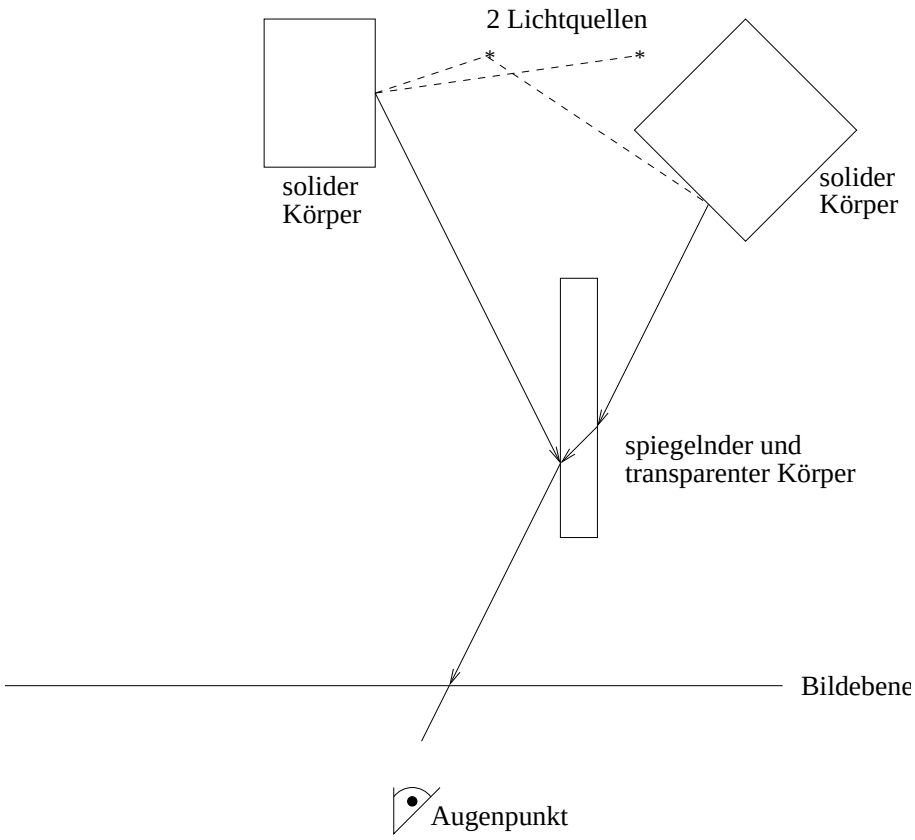

Abbildung 23.1: Prinzip der Strahlverfolgung

Verfahren mit *Ray Tracing* (Strahlverfolgung) eignen sich ausgezeichnet zur Modellierung von spiegelnden Reflexionen und von Transparenz mit Brechung (ohne Streuung). Globale Lichter werden mit einem ambienten Beleuchtungsterm ohne Richtung behandelt. Ausgehend vom Augenpunkt wird

durch jedes Pixel ein Strahl gelegt und der Schnittpunkt dieses Strahls mit dem ersten getroffenen Objekt bestimmt. Trifft der Strahl auf kein Objekt, so erhält das Pixel die Hintergrundfarbe. Ist das Objekt spiegelnd, so wird der Reflexionsstrahl berechnet und rekursiv weiterbehandelt. Ist das Objekt transparent, wird zusätzlich der gebrochene Strahl weiterbehandelt. Zur Berechnung von Schatten wird von jedem Schnittpunkt zwischen Strahl und Objekt zu jeder Lichtquelle ein zusätzlicher Strahl ausgesandt. Trifft dieser Strahl auf ein blockierendes Objekt, dann liegt der Schnittpunkt im Schatten dieser Lichtquelle, und das von ihr ausgestrahlte Licht geht in die Intensitätsberechnung des Punktes nicht ein.

23.2 Ermittlung sichtbarer Flächen durch Ray Tracing

Verfahren mit *Ray Tracing* ermitteln die Sichtbarkeit von Flächen, indem sie imaginäre Lichtstrahlen des Betrachters zu den Objekten der Szene verfolgen. Man wählt ein Projektionszentrum (das Auge des Betrachters) und ein Window in einer beliebigen Bildebene. Das Window wird durch ein regelmäßiges Gitter aufgeteilt, dessen Elemente den Pixeln in der gewünschten Auflösung entsprechen. Dann schickt man für jedes Pixel im Window einen *Augstrahl* vom Projektionszentrum durch den Mittelpunkt des Pixels auf die Szene. Das Pixel erhält die Farbe des ersten getroffenen Objekts. Das folgende Programm enthält den Pseudocode für diesen einfachen Ray Tracer.

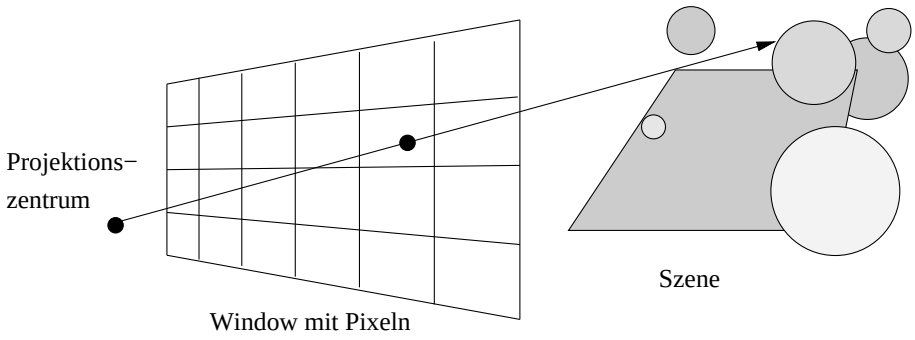

Abbildung 23.2: Strahl durch Bildebene

```
Wähle Projektionszentrum und Window in der Bildebene;
for(jede Rasterzeile des Bildes){
   for(jedes Pixel der Rasterzeile){
     Berechne den Strahl vom Projektionszentrum durch das Pixel;
     for(jedes Objekt der Szene){
        if(Objekt wird geschnitten und liegt bisher am nächsten)
           Speichere Schnittpunkt und Name des Objekts;
     }
     Setze das Pixel auf die Farbe des nächstliegenden Objektschnittpunkts;
   }
}
```

23.3 Berechnung von Schnittpunkten

Hauptaufgabe eines jeden Ray Tracers ist es, den Schnittpunkt eines Strahls mit einem Objekt zu bestimmen. Man benutzt dazu die parametrisierte Darstellung eines Vektors. Jeder Punkt (x,y,z) auf dem Strahl von (x_0,y_0,z_0) nach (x_1,y_1,z_1) wird durch einen bestimmten Wert t definiert mit

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad z = z_0 + t(z_1 - z_0) \; .$$

Zur Abkürzung definiert man $\Delta x, \Delta y$ und Δz als

$$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0 \; .$$

Damit kann man schreiben

$$x = x_0 + t\Delta x, \quad y = y_0 + t\Delta y, \quad z = z_0 + t\Delta z \; .$$

Ist (x_0,y_0,z_0) das Projektionszentrum und (x_1,y_1,z_1) der Mittelpunkt eines Pixels im Window, so durchläuft t zwischen diesen Punkten die Werte von Null bis Eins. Negative Werte für t liefern Punkte hinter dem Projektionszentrum, Werte größer als Eins stellen Punkte auf der Seite des Windows dar, die vom Projektionszentrum abgewandt ist. Man braucht für jeden Objekttyp eine Darstellung, mit der man den Wert von t am Schnittpunkt des Strahls mit dem Objekt bestimmen kann. Die Kugel bietet sich dafür als eines der einfachsten Objekte an. Aus diesem Grund tauchen Kugeln auch so oft in Ray-Tracing-Bildern auf. Eine Kugel um den Mittelpunkt (a,b,c) und Radius r kann durch die Gleichung

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$$

dargestellt werden. Zur Berechnung des Schnittpunkts multipliziert man die Gleichung aus und setzt x,y und z aus der Gleichung von oben ein.

Man erhält eine quadratische Gleichung in t, deren Koeffizienten nur die Konstanten aus den Gleichungen der Kugel und des Strahls enthalten. Man kann sie also mit der Lösungsformel für quadratische Gleichungen lösen. Wenn es keine reellen Lösungen gibt, schneidet der Strahl die Kugel nicht. Gibt es genau eine Lösung, dann berührt der Strahl die Kugel. Andernfalls geben die beiden Lösungen die Schnittpunkte mit der Kugel an. Der Schnittpunkt mit dem kleinsten positiven t-Wert liegt am nächsten.

Man muß zur Schattierung der Fläche die Flächennormale im Schnittpunkt berechnen. Im Fall der Kugel ist das besonders einfach, weil die (nicht normalisierte) Normale einfach der Vektor vom Kugelmittelpunkt zum Schnittpunkt ist: Die Kugel mit Mittelpunkt (a,b,c) hat im Schnittpunkt (x,y,z) die Flächennormale $((x-a)/r,(y-b)/r,(z-c)/r)$.

Es ist etwas schwieriger, den Schnittpunkt des Strahls mit einem Polygon zu berechnen. Um festzustellen, ob der Strahl ein Polygon schneidet, testet man zuerst, ob der Strahl die Ebene des Polygons schneidet und anschließend, ob der Schnittpunkt innerhalb des Polygons liegt.

23.4 Effizienzsteigerung zur Ermittlung sichtbarer Flächen

Die vorgestellte einfache, aber rechenintensive Version des Ray-Tracing-Algorithmus schneidet jeden Augstrahl mit jedem Objekt der Szene. Für ein Bild der Größe 1024×1024 mit 100 Objekten wären

daher 100 Mio. Schnittpunktberechnungen erforderlich. Ein System verbraucht bei typischen Szenen 75-95 Prozent der Rechenzeit für die Schnittpunktroutine. Daher konzentrieren sich die Ansätze zur Effizienzsteigerung auf die Beschleunigung der Schnittpunktberechnungen oder deren völlige Vermeidung.

Optimierung der Schnittpunktberechnungen

Die Formel für den Schnittpunkt mit einer Kugel läßt sich verbessern. Wenn man die Strahlen so transformiert, daß sie entlang der z-Achse verlaufen, kann man die gleiche Transformation auf die getesteten Objekte anwenden. Dann liegen alle Schnittpunkte bei $x = y = 0$. Dieser Schritt vereinfacht die Berechnung der Schnittpunkte. Das nächstliegende Objekt erhält man durch Sortieren der z-Werte. Mit der inversen Transformation kann man den Schnittpunkt dann für die Schattierungsberechnungen zurücktransformieren.

Begrenzungsvolumina eignen sich besonders gut dazu, die Zeit für die Berechnung der Schnittpunkte zu reduzieren. Man umgibt ein Objekt, für das die Schnittpunktberechnungen sehr aufwendig sind, mit einem Begrenzungsvolumen, dessen Schnittpunkte einfacher zu berechnen sind, z.B. Kugel, Ellipsoid oder Quader. Das Objekt wird nur dann getestet, wenn der Strahl das Begrenzungsvolumen schneidet.

Hierarchien

Begrenzungsvolumina legen zwar selbst noch keine Reihenfolge oder Häufigkeit der Schnittpunkttests fest. Sie können aber in verschachtelten Hierarchien organisiert werden. Dabei bilden die Objekte die Blätter, die inneren Knoten begrenzen ihre Söhne. Hat ein Strahl keinen Schnittpunkt mit einem Vaterknoten, dann gibt es garantiert auch keinen Schnittpunkt mit einem seiner Söhne. Beginnt der Schnittpunkttest also an der Wurzel, dann entfallen ggf. trivialerweise viele Zweige der Hierarchie (und damit viele Objekte).

Bereichsunterteilung

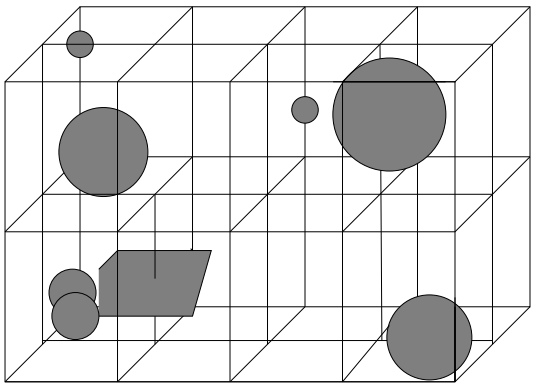

Abbildung 23.3: Unterteilung der Szene durch ein regelmäßiges Gitter

Die Hierarchie von Begrenzungsvolumina organisiert die Objekte von unten nach oben. Die Bereichsunterteilung teilt dagegen von oben nach unten auf. Zuerst berechnet man die *bounding box* der ganzen Szene. Bei einer Variante unterteilt man die *bounding box* dann in ein regelmäßiges Gitter gleich großer Bereiche. Jedem Bereich wird eine Liste mit den Objekten zugewiesen, die entweder ganz oder teilweise darin enthalten sind. Zur Erzeugung der Listen weist man jedes Objekt einem oder mehreren Bereichen zu, die dieses Objekt enthalten. Ein Strahl muß jetzt nur noch mit den Objekten in den durchlaufenen Bereichen geschnitten werden. Außerdem kann man die Bereiche in der Reihenfolge

untersuchen, in der sie vom Strahl durchlaufen werden. Sobald es in einem Bereich einen Schnittpunkt gibt, muß man keine weiteren Bereiche mehr testen. Man muß jedoch alle anderen Objekte des Bereichs untersuchen, um das Objekt mit dem nächstliegenden Schnittpunkt zu ermitteln.

23.5 Rekursives Ray Tracing

In diesem Abschnitt wird der Basisalgorithmus zum Ray Tracing auf die Behandlung von Schatten, Reflexion und Brechung erweitert. Der einfache Algorithmus ermittelte die Farbe eines Pixels am nächsten Schnittpunkt eines Augstrahls und eines Objekts mit einem beliebigen der früher beschriebenen Beleuchtungsmodelle. Zur Schattenberechnung wird ein zusätzlichen Strahl vom Schnittpunkt zu allen Lichtquellen geschickt. Schneidet einer dieser *Schattenstrahlen* auf seinem Weg ein Objekt, dann liegt das Objekt am betrachteten Punkt im Schatten, und der Schattierungsalgorithmus ignoriert den Beitrag der Lichtquelle dieses Schattenstrahls.

Der rekursive Ray-Tracing-Algorithmus startet zusätzlich zu den Schattenstrahlen weitere *Spiegelungs-* und *Brechungsstrahlen* im Schnittpunkt (siehe Abbildung 23.4). Diese Spiegelungs-, Reflexions- und Brechungsstrahlen heißen *Sekundärstrahlen*, um sie von den *Primärstrahlen* zu unterscheiden, die vom Auge ausgehen. Gibt es bei dem Objekt spiegelnde Reflexion, dann wird ein Spiegelungsstrahl um die Flächennormale in Richtung r gespiegelt. Ist das Objekt transparent, und es gibt keine totale innere Reflexion, startet man einen Brechungsstrahl entlang t in das Objekt. Der Winkel wird nach dem Gesetz von Snellius bestimmt.

Jeder dieser Reflexions- und Brechungsstrahlen kann wiederum neue Spiegelungs-, Reflexions- und Brechungsstrahlen starten.

Typische Bestandteile eines *Ray-Tracing*-Programms sind die Prozeduren `shade` und `intersect`, die sich gegenseitig aufrufen. `shade` berechnet die Farbe eines Punktes auf der Oberfläche. Hierzu werden die Beiträge der reflektierten und gebrochenen Strahlen benötigt. Diese Beiträge erfordern die Bestimmung der nächstgelegenen Schnittpunkte, welche die Prozedur `intersect` liefert. Der Abbruch der Strahlverfolgung erfolgt bei Erreichen einer vorgegebenen Rekursionstiefe oder wenn ein Strahl kein Objekt trifft.

Seien v, N die normierten Vektoren für Sehstrahl und Normale. Sei $n = n_2/n_1$ das Verhältnis der beiden Brechungsindizes. Dann sind die Vektoren r und t, die im Schnittpunkt P zur Weiterverfolgung des reflektierten und gebrochenen Strahls benötigt werden, definiert durch

$$\begin{aligned} r &= 2\cos(\phi)N - v \\ t &= (-\cos(\phi) - q)N + v \\ \text{mit } q &= \sqrt{n^2 - 1 + \cos^2(\phi)} \end{aligned}$$

Ist q ein imaginärer Ausdruck, liegt Totalreflexion vor, und der Transmissionsanteil wird gleich Null gesetzt.

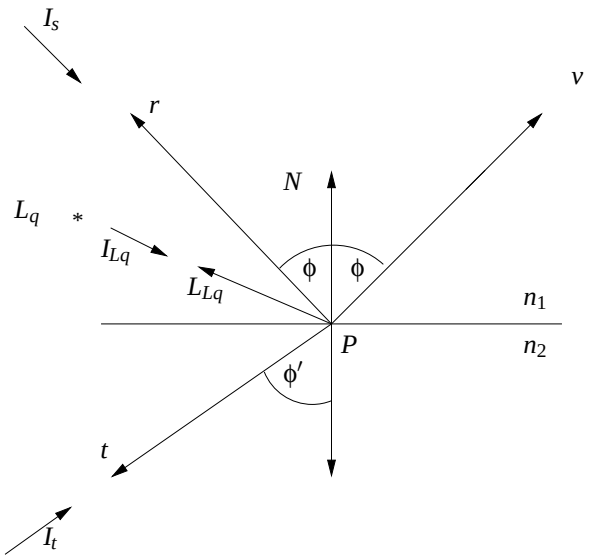

Abbildung 23.4: Einflußgrößen für Punkt P

In die Berechnung der Farbe im Punkt P geht die gewichtete Summe der in den einzelnen Rekursionsschritten berechneten Farbintensitäten ein. Unter Berücksichtigung des *Phong*-Modells ergibt sich

$$I = k_a I_a + k_d \sum_{Lq} I_{Lq}(N \cdot L_{Lq}) + k_s \sum_{Lq} I_{Lq}(A \cdot R_{Lq})^c + k_s I_s + k_t I_t$$

mit

k_a	ambienter Reflexionskoeffizient
I_a	Hintergrundbeleuchtung
k_d	diffuser Reflexionskoeffizient
I_{Lq}	Intensität der Lichtquelle
N	Normale auf Ebene
L_{Lq}	Vektor zur Lichtquelle
k_s	spiegelnder Reflexionskoeffizient
I_s	Reflexionssintensität (rekursiv ermittelt)
A	Vektor zum Augenpunkt
R_{Lq}	Reflexionsvektor für Lichtquelle L_q
c	spekularer Exponent
k_t	Transmissionskoeffizient (materialabhängig)
I_t	Transmissionsintensität (rekursiv ermittelt)

Pseudocode für einfaches rekursives Ray Tracing

```
Wähle Projektionszentrum und Window in der view plane;
for(jede Rasterzeile des Bildes){
   for(jedes Pixel P der Rasterzeile){
     Ermittle Strahl vom Projektionszentrum durch das Pixel;
     Color C = RT_intersect(ray, 1);
     Färbe Pixel P mit Farbe C;
   }
}

/* Schneide den Strahl mit Objekten, und berechne die Schattierung am nächsten */
/* Schnittpunkt. Der Parameter depth ist die aktuelle Tiefe im Strahlenbaum. */

RT_color RT_intersect(RT_ray ray, int depth)
{
   Ermittle den nächstliegenden Schnittpunkt mit einem Objekt;
   if(Objekt getroffen){
     Berechne die Normale im Schnittpunkt;
     return RT_shade(nächstliegendes getroffenes Objekt, Strahl, Schnittpunkt,
                     Normale, depth);
   }
   else
     return HINTERGRUND_FARBE;
}



/* Berechne Schattierung für einen Punkt durch Verfolgen der Schatten-, */
/* Reflexions- und Brechungsstrahlen */

RT_color RT_shade(
   RT_object object,          /* Geschnittenes Objekt */
   RT_ray ray,                /* Einfallender Strahl */
   RT_point point,            /* Schnittpunkt, der schattiert werden soll */
   RT_normal normal,          /* Normale in dem Punkt */
   int depth)                 /* Tiefe im Strahlenbaum */
{
RT_color color;               /* Farbe des Strahls */
RT_ray rRay, tRay, sRay;      /* Reflexions-, Brechungs- und Schattenstrahlen */
RT_color rColor, tColor;      /* Farbe des reflektierten und gebrochenen Strahls */

   color = Term für das ambiente Licht;


   for(jede Lichtquelle){
     sRay = Strahl von der Lichtquelle zum Punkt;
     if(Skalarprodukt der Normalen mit der Richtung zum Licht ist positiv){
        Berechne, wieviel Licht von opaken und transparenten Flächen blockiert wird;
        Skaliere damit die Terme für diffuse und spiegelnde Reflexion, bevor sie
        zur Farbe addiert werden;
     }
}
```

```
if(depth < maxDepth){        /* Bei zu großer Tiefe beenden */
     if(Objekt reflektiert){
        rRay = Strahl vom Punkt in Reflexionsrichtung;
        rColor = RT_intersect(rRay, depth + 1);
        Skaliere rColor mit dem Spiegelungskoeffizienten,
        und addiere den Wert zur Farbe;
     }
     if(Objekt ist transparent){
        tRay = Strahl vom Punkt in Brechungsrichtung;
        if(es gibt keine totale interne Reflexion){
           tColor = RT_intersect(tRay, depth + 1);
           Skaliere tColor mit dem Transmissionskoeffizienten,
           und addiere den Wert zur Farbe;
        }
     }
   }
   return color;             /* Liefere die Farbe des Strahls zurück */
}
```

Ray Tracing ist besonders anfällig für Probleme, die durch die begrenzte Rechengenauigkeit entstehen. Dies zeigt sich vor allem bei der Berechnung der Schnittpunkte sekundärer Strahlen mit den Objekten. Wenn man die *x*-, *y*- und *z*-Koordinaten eines Objekts mit einem Augstrahl berechnet hat, dienen sie zur Definition des Startpunkts eines Sekundärstrahls. Für diesen muß dann der Parameter t bestimmt werden. Wird das eben geschnittene Objekt mit dem neuen Strahl geschnitten, hat t wegen der begrenzten Rechengenauigkeit oft einen kleinen Wert ungleich Null. Dieser falsche Schnittpunkt kann sichtbare Probleme bewirken.

23.6 Public Domain Ray Tracer Povray

Der “Persistence of Vision Ray Tracer” (POV-Ray) ist ein urheberrechtlich geschütztes Freeware-Programm zum Berechnen einer fotorealitischen Projektion aus einer 3-dimensionalen Szenenbeschreibung auf der Grundlage der Strahlverfolgung (`http://www.povray.org`). Seine Implementierung basiert auf DKBTrace 2.12 von David Buck & Aaron Collins. Zur Unterstützung des Anwenders gibt es einige include-Files mit vordefinierten Farben, Formen und Texturen.

```
#include "colors.inc"
#include "textures.inc"

camera {
       location <3, 3, -1>
       look_at  <0, 1,  2>
}
light_source { <0, 4, -3> color White}
light_source { <0, 6, -4> color White}
light_source { <6, 4, -3> color White}

plane {
       <0, 1, 0>, 0
       pigment {
               checker
                      color White
                      color Black
       }
}

sphere {
       <0, 2, 4>, 2
       texture {
               pigment {color Orange}
               normal  {bumps 0.4  scale 0.2}
               finish  {phong 1}
       }
}

box {
       <-1, 0, -0.5>,
       < 1, 2,  1.5>
       pigment {
               DMFWood4
               scale 2
       }
       rotate y*(-20)
}
```

Povray-Quelltext scene.pov

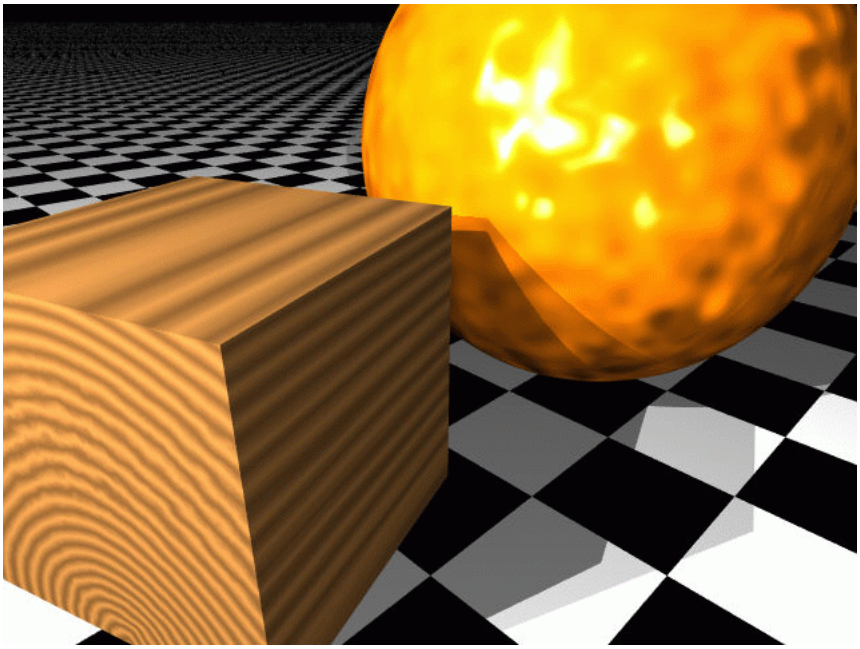

Abbildung 23.5: Povray-Bild zur Szenenbeschreibung

Der Aufruf von *povray* erfolgt zweckmäßigerweise mit einem Shell-Skript, dem der Name der zu bearbeitenden Datei als einziger Parameter übergeben wird:

```
#!/bin/sh

povray +I$1.pov +Oscene.tga +L/usr/lib/povray3/include +W320 +H200
```

Die vom Ray-Tracer erstellte Statistik lautet:

```
scene.pov Statistics, Resolution 320 x 200
-----------------------------------------------------------------------------
Pixels:           64000   Samples:           64000   Smpls/Pxl: 1.00
Rays:             64000   Saved:                 0   Max Level: 1/5
-----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
-----------------------------------------------------------------------------
Box                             238037           40342     16.95
Plane                           238037           62400     26.21
Sphere                          238037           50948     21.40
-----------------------------------------------------------------------------
Calls to Noise:                  0   Calls to DNoise:          83403
-----------------------------------------------------------------------------
Shadow Ray Tests:           522111   Succeeded:                24497
-----------------------------------------------------------------------------
Smallest Alloc:                 12 bytes   Largest:            12308
Peak memory used:           103504 bytes
-----------------------------------------------------------------------------
Time For Trace:    0 hours  0 minutes  18.0 seconds (18 seconds)
    Total Time:    0 hours  0 minutes  18.0 seconds (18 seconds)
```