Kapitel 17

Entfernen verdeckter Kanten und Flächen

Beim Entfernen verdeckter Kanten und Flächen werden unterschieden:

- Objektraum-Algorithmen (arbeiten auf Weltkoordinaten, vergleichen Objekte)
- Bildraum-Algorithmen (arbeiten auf Device-Koordinaten, vergleichen Pixel)

Das zunächst vorgestellte Back-Face Removal entfernt nur die Rückflächen einzelner Objekte, indem es im Objektraum (WC oder NPC) die Beziehung zwischen Betrachter und Objektfläche untersucht. Ob ein Objekt ein anderes verdeckt, kann dadurch nicht festgestellt werden.

17.1 Back-Face Removal/Culling

Ein Körper besteht aus Flächen, die dem Betrachter zugewandt sind (sogenannte Vorderflächen oder Front Faces) und solchen, die vom Betrachter abgewandt sind (sogenannte Rückflächen oder Back Faces). Die Rückflächen sind nicht sichtbar, da sie stets von Vorderflächen verdeckt sind. Die Unterdrückung der Rückflächen (*Back-Face Removal* oder *Back-Face Culling*) ist daher ein erster Schritt in Richtung einer natürlichen Darstellung des Kantenmodells. Für einen konvexen Körper bestimmt das Back-Face Culling exakt den sichtbaren Teil. Ist er dagegen nicht konvex, so werden zwar mehr Kanten angezeigt, als wirklich sichtbar sind, die Darstellung ist jedoch schon sehr realistisch. Ein weiterer Aspekt ist, daß bei einer komplexen Szene circa die Hälfte aller Flächen Rückflächen sind. Durch ihren Ausschluß halbieren sich in etwa die weiteren Berechnungen für Lighting und Shading.

Das Programm benutzt eine sehr effiziente Methode unter Verwendung der Normalenvektoren, um Vorder- und Rückflächen zu unterscheiden. Für die Flächen eines Objekts sind die Normalen so definiert, daß sie nach außen zeigen. Liegt der Betrachterstandpunkt auf der Außenseite einer Fläche, so handelt es sich um eine Vorder-, sonst um eine Rückfläche.

Wenn $\mathbf{n}$ der Normalenvektor der Fläche und $\mathbf{a}$ ein Eckpunkt ist, dann kann hieraus die Gleichung der Ebene, in der die Fläche liegt, in der *Hesseschen Normalform* bestimmt werden:

$$\mathbf{p} \cdot \mathbf{n}^{\mathrm{T}} - \mathbf{a} \cdot \mathbf{n}^{\mathrm{T}} = e.$$

Beim Einsetzen verschiedener Punkte $\mathbf{p}$ ergeben sich unterschiedliche Werte für e. Gilt $e = 0$, so liegt $\mathbf{p}$ in der Ebene, bei $e > 0$ befindet sich $\mathbf{p}$ außen, d.h., die Fläche ist von $\mathbf{p}$ aus sichtbar, und bei $e < 0$ liegt $\mathbf{p}$ innen, d.h., die Fläche ist von $\mathbf{p}$ aus unsichtbar.

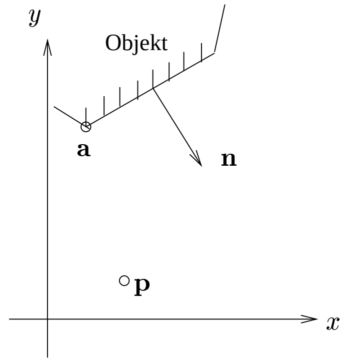


Abbildung 17.1: Back-Face Removal mit Hessescher Normalform

Um die Rückflächen zu erkennen, ist für alle Flächen die obige Ebenengleichung zu berechnen, in die der Betrachterstandpunkt eingesetzt wird. Ergibt sich $e \geq 0$, so handelt es sich um eine Vorderfläche, sonst um eine Rückfläche. Dabei ist zu beachten, daß ggf. für jeden Eckpunkt eine eigene Normale definiert ist. Dies ist zum Approximieren von gekrümmten Flächen notwendig. Bei einer Rückfläche sind definitionsgemäß alle Normalenvektoren abgewandt. Flächen, bei denen einige Normalen zum Betrachter und andere von ihm weg zeigen, werden teilweise dargestellt. Es ist Aufgabe des später vorzustellenden Shaders, die sichtbaren Pixel zu bestimmen. Bei der Liniendarstellung wird eine Kante gezeichnet, wenn mindestens ein Eckpunkt sichtbar ist.

Eine algorithmische Vereinfachung des Tests auf Sichtbarkeit ergibt sich durch die Transformation der Ebenengleichung ins NPC. Hierbei wird ausgenutzt, daß der Betrachterstandpunkt im NPC per Definition die homogenen Koordinaten $(0, 0, 1, 0)$ besitzt. (Er liegt im Unendlichen auf der positiven z-Achse.) Daraus folgt, daß das Skalarprodukt zwischen Ebene und Betrachterstandpunkt im NPC identisch mit der z-Komponente des homogenen Normalenvektors der Fläche ist.

Zur Klassifizierung des Flächentyps wird daher im Programm zunächst nur die z-Komponente der homogenen Normalenvektoren ins NPC transformiert. Durch diese sehr effiziente Methode, Rückflächen zu eliminieren, halbiert sich in etwa die Berechnung zur Darstellung eines Objekts.

17.2 Hidden-Surface Removal

Bei der Darstellung von 3-D-Szenen ist das Problem zu lösen, daß der Betrachter nur die Objekte sehen sollte, die im Vordergrund liegen; alle anderen sind zumindest teilweise verdeckt. Das Entfernen der nicht sichtbaren Flächen wird als *Hidden-Surface Removal* bezeichnet.

17.2.1 z-Buffer-Algorithmus

Der hier vorgestellte *z-Buffer-Algorithmus* löst das Problem nach dem folgenden Prinzip: Für alle Pixel des Bildschirms wird die z-Komponente als Wert für die Tiefe im Raum gespeichert. Die benötigte Datenstruktur, der sogenannte z-Buffer, ist im Programm ein 2-dimensionales Array, das für jedes Pixel den z-Wert des in diesem Punkt dem Betrachter am nächsten liegenden Objekts enthält. In einem gleichgroßen Feld, dem Frame Buffer, werden die Farbwerte der Pixel gespeichert. Die Menge der Farbwerte stellt den Bildschirmspeicher dar.

Der z-Wert wird mit Null und der Farbwert mit der Hintergrundfarbe der Szene initialisiert.

Laut Definition der Transformationspipeline treten nach dem Clipping im NPC nur noch z-Werte zwischen Null und Eins auf. Die Initialisierung mit Null entspricht dem größtmöglichen Abstand vom Betrachter (back plane).

Durch die Rasterung der Flächen erhalten die Pixel auch einen interpolierten z-Wert, der mit dem z-Buffer-Wert an dieser Stelle verglichen wird. Ist der z-Wert des Pixels größer, so ist das Pixel (vorläufig) sichtbar. Sein z-Wert wird in den z-Buffer und sein Farbwert in den Bildschirmspeicher eingetragen. Ist der z-Wert des Pixels kleiner, so ist der zugehörige Teil der Fläche verdeckt; die Inhalte von z-Buffer und Bildschirmspeicher bleiben erhalten. Nach der Abarbeitung aller Flächen enthält der Bildschirmspeicher die Abbildung der sichtbaren Flächen bzw. Flächenteile und der z-Buffer die zugehörige Tiefeninformation.

Der z-Buffer-Algorithmus entscheidet pixelweise über die Verdeckungseigenschaften und ist daher sehr allgemeingültig. Die darzustellenden ebenen Flächen brauchen nicht vorsortiert zu werden und dürfen sich gegenseitig durchdringen. Da er am Ende der Transformationspipeline im DC arbeitet, wird er als Bildraumalgorithmus klassifiziert.

Der z-Buffer-Algorithmus wird vom Programm in der inneren Schleife des Scanline-Algorithmus aufgerufen und läßt sich wie folgt skizzieren:

```
Für jede Fläche F tue:
        Für jedes Pixel (x, y) auf dieser Fläche tue:
        berechne Farbe c und Tiefe z
        falls z > tiefe[x, y]:
        dann wird c an der Stelle x,y im Frame Buffer eingetragen
        und tiefe[x, y] auf z gesetzt.
```

Ist die Fläche durch $Ax + By + Cz + D = 0$ gegeben, so ist die Tiefe im Punkt (x, y):

$$z = -\frac{Ax + By + D}{C}$$

Für auf einer Scanline benachbarte Punkte (x_i, y_j) und (x_{i+1}, y_j) ergibt sich

$$z_{i+1} = z_i - \frac{A}{C}$$

Für die Punkte (x_i, y_j) und (x_i, y_{j+1}) zweier benachbarter Scanlines ergibt sich

$$z_{j+1} = z_j - \frac{B}{C}$$

Das größte Problem bei der Implementierung des z-Buffers ist sein Speicherplatzbedarf. Eine ausreichende Auflösung der Tiefeninformation ergibt sich erst mit einem 32-Bit z-Wert, d.h. $maxint = (2^{31} - 1)$. Für die RGB-Tripel werden drei Byte benötigt, die in einem 32-Bit Integer kodiert werden. Pro Pixel belegen der z-Buffer und der Bildschirmspeicher also acht Byte. Bei einer Bildschirmauflösung von 1024×768 Pixeln ergeben sich folglich 6 MB.

17.3 Beispiel-Applet zur Wire-Frame-Projektion

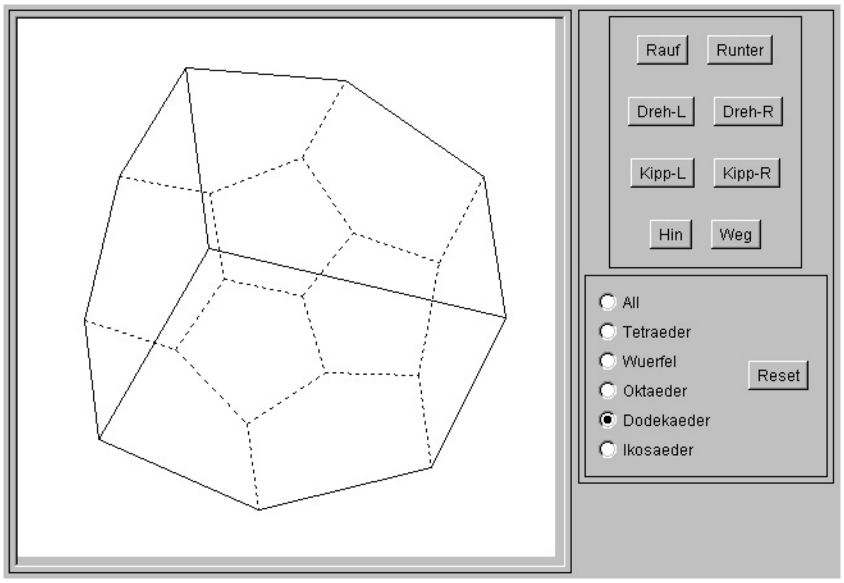


Abbildung 17.2: Screenshot vom 3D-wire-Applet

Kapitel 18

Rasterung von Flächen

Die reellwertigen Koordinaten der Objekte im DC am Ende der Transformationspipeline müssen in ganzzahlige Pixelkoordinaten gerundet werden. Die Rasterung von Flächen kann mit Hilfe des Scanline-Verfahrens auf die von Linien zurückgeführt werden. In einem zweistufigen Algorithmus wird die Fläche zunächst in eine Menge horizontaler Linien, die sogenannten *Spans*, zerlegt. Die Eckpunkte dieser Spans ergeben sich durch Rasterung der Kanten nach dem vorgestellten Verfahren. In der zweiten Stufe wird dann jeder der Spans gerastert.

Das *Scanline*-Verfahren wird zur Vereinfachung und Beschleunigung des Rendering-Programms nur auf Dreiecke angewandt, denn Dreiecke sind die einfachsten Polygone. Gegenüber allgemeinen Polygonen bieten sie den Vorteil, daß sie planar und konvex sind. Für das Scanline-Verfahren eignen sie sich ausgezeichnet, da für jede Bildschirmzeile (Scanline) maximal zwei Schnittpunkte mit den Dreieckskanten auftreten.

Ein konvexes Polygon läßt sich gemäß der Abbildung triangulisieren, indem von einem beliebigen Eckpunkt aus zu allen nicht benachbarten Eckpunkten Diagonalen gezogen werden.

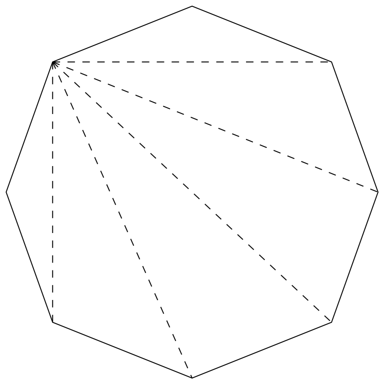


Abbildung 18.1: Triangulisierung eines konvexen Polygons

Unter Rasterung wird hier nicht nur das Füllen der Fläche in der Objektfarbe verstanden. Neben den Pixelkoordinaten müssen auch die z-Werte der Polygonpunkte interpoliert werden, um mit Hilfe dieser Tiefeninformation verdeckte Flächen zu unterdrücken (siehe z-Buffer-Algorithmus).

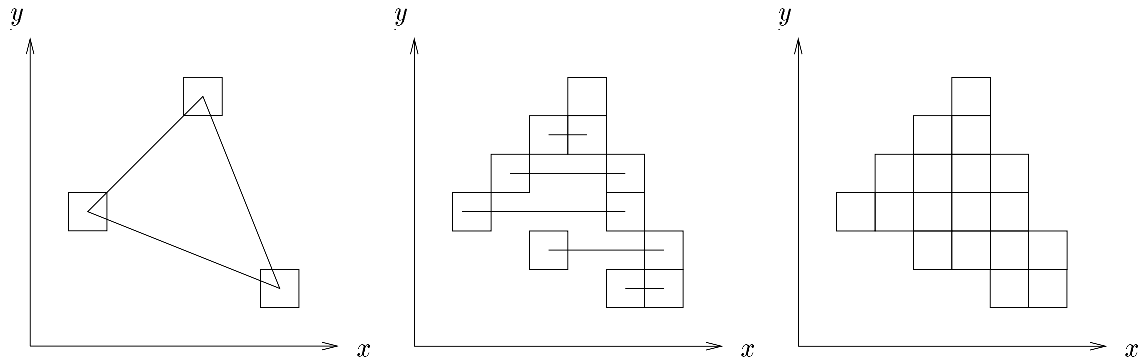


Abbildung 18.2: Scanline-Verfahren zur Rasterung von Dreiecken

Außerdem sind je nach Schattierungs-Verfahren auch die Farbwerte bzw. die Normalenvektoren, die in den Eckpunkten gegeben sind, über die Fläche zu interpolieren. Verschiedene Farbwerte auf den Objektflächen ergeben sich durch die individuelle Beleuchtung der Punkte. Die Interpolation der Normalen simuliert dabei einen gekrümmten Flächenverlauf. Grundlegend für die Schattierungs-Verfahren ist die Beleuchtung einer Szene mit Hilfe von unterschiedlichen Lichtmodellen.

18.1 Beleuchtung

Um künstlich hergestellte Bilder wirklich realistisch aussehen zu lassen, muß eine Beleuchtung der darzustellenden Objekte durchgeführt werden. Dieser Prozeß wird in der Computergrafik *Lighting* genannt. Die dabei verwendeten Modelle bilden die Lichtverhältnisse sowie die Oberflächenbeschaffenheiten der Objekte nicht völlig naturgetreu nach, sondern sind nur Näherungen, deren Qualität vom Rechenaufwand abhängt. Die erzeugten Bilder bzw. die eingesetzten Modelle stellen somit einen Kompromiß zwischen Darstellungsgenauigkeit und verfügbarer Rechenzeit dar.

Die Intensität des Lichts, das von einer Oberfläche empfangen wird, hängt von den Lichtquellen in der Umgebung sowie von der Struktur und der Materialart der beleuchteten Fläche ab. Ein Teil des empfangenen Lichts wird vom Körper absorbiert und der Rest reflektiert. Wird das gesamte Licht absorbiert, ist der Körper nicht direkt sichtbar, sondern er hebt sich schwarz vom Hintergrund ab. Erst durch die Reflexion des Lichts wird das Objekt selbst sichtbar. Seine Farbe hängt vom Anteil der absorbierten und reflektierten Farben ab.

In die Berechnung des Farbwertes eines Objektpunktes fließen ein:

- der Augenpunkt des Betrachters,
- die Zahl und Art der Lichtquellen,
- die Materialeigenschaften des Objekts,
- der Normalenvektor des Objekts in diesem Punkt.

18.1.1 Lichtquellen

Der Grafikstandard PHIGS PLUS definiert vier Modelle von Lichtquellen:

- Umgebungslicht (*ambient light*),
- gerichtetes Licht (*directed light* oder *infinite light*),
- Punktlicht (*point light* oder *positional light*),
- Strahler (*spot light*).

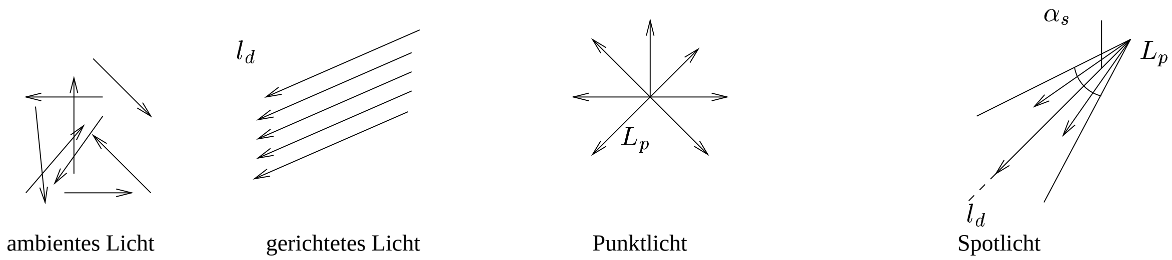


Abbildung 18.3: Vier Modelle von Lichtquellen (schematisch)

Das einfachste Lichtmodell ist das *Umgebungslicht*, das durch eine konstante Intensität I_a definiert ist. Diese ungerichtete Hintergrundbeleuchtung simuliert das Licht, das durch Reflexion an den einzelnen Gegenständen überall in einer Szene vorhanden ist.

Gerichtetes Licht ist definiert durch eine Intensität I und eine Lichtrichtung $\mathbf{l}_d$. Es ist gekennzeichnet durch Lichtstrahlen, die parallel aus dem Unendlichen in eine bestimmte Richtung strahlen, vergleichbar mit dem Sonnenlicht.

Ein *Punktlicht* mit der Intensität I_0 hat eine Position $\mathbf{L}_p$ im Raum (WC), von der aus es in alle Richtungen gleichförmig abstrahlt wie eine Glühbirne. Die Intensität des Lichts nimmt mit zunehmender Entfernung r ab nach der Formel

$$I(r) = \frac{I_0}{C_1 + C_2 \cdot r}.$$

Dabei bezeichnet r den Abstand von der Punktlichtquelle. C_1 und C_2 heißen Abschwächungskoeffizienten (*attenuation coefficients*, $C_1 \geq 1$, $C_2 \geq 0$).

Das aufwendigste Lichtmodell ist der *Strahler* (*Spot*). Dieser hat wie das Punktlicht eine Intensität I_0, eine Position $\mathbf{L}_p$ sowie die Abschwächungskoeffizienten C_1 und C_2. Dazu kommt eine bevorzugte Lichtrichtung $\mathbf{l}_d$, um die herum das Licht nur im Winkel α_s (*spread angle*) abgestrahlt wird. Außerhalb dieses Lichtkegels ist die Intensität Null. Der *concentration exponent* L_e definiert, wie stark die ausgesandte Lichtenergie mit zunehmendem Winkel zwischen $\mathbf{l}_d$ und der Strahlrichtung $\mathbf{r}$ abnimmt. Die Abnahme ist proportional zu

$$\cos(\mathbf{r}, \mathbf{l}_d)^{L_e}$$

innerhalb des durch α_s definierten Lichtkegels. Je größer L_e, um so stärker ist die Intensität im Zentrum des Lichtkegels gebündelt.

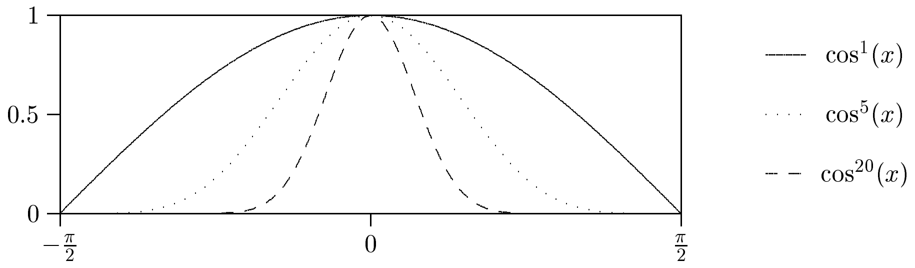


Abbildung 18.4: Lichtintensitätsverteilung beim Strahler (schematisch)

18.1.2 Reflexionseigenschaften

Die Lichtreflexion an Objekten im fast realistischen *Phong-Modell* besteht aus drei Termen:

- ambientes Licht,
- diffus reflektiertes gerichtetes Licht,
- spekular reflektiertes gerichtetes Licht.

Diese Terme werden berechnet und zu einer Farbe zusammengefaßt, um den resultierenden Farbeindruck zu erhalten:

$$\overline{C} = \sum_{i=1}^{n} (\overline{C_{a_i}} + \overline{C_{d_i}} + \overline{C_{s_i}}) \ .$$

Dabei ist $\overline{C}$ das gesamte von einem Punkt der Oberfläche reflektierte farbige Licht, n ist die Anzahl der Lichtquellen, und $\overline{C_{a_i}}$, $\overline{C_{d_i}}$, $\overline{C_{s_i}}$ sind die ambienten, diffusen und spekularen Anteile des reflektieren Lichts für die Lichtquelle i. Die Herleitungen müssen im RGB-Modell für jede der drei Grundfarben getrennt betrachtet werden.

18.1.3 Oberflächeneigenschaften

Das Reflexionsverhalten (und damit das Erscheinungsbild) eines Körpers wird durch folgende Oberflächeneigenschaften bestimmt:

k_a	ambienter Reflexionskoeffizient,
k_d	diffuser Reflexionskoeffizient,
k_s	spekularer Reflexionskoeffizient,
$\overline{O_d}$	diffuse Objektfarbe,
$\overline{O_s}$	spekulare Objektfarbe,
O_e	spekularer Exponent.

Die *ambiente Reflexion* gibt das Verhalten eines Körpers bei ambienter Beleuchtung wieder. Der ambiente Reflexionskoeffizient k_a gibt an, wieviel des einfallenden ambienten Lichts der Intensität I_a vom Objekt in seiner diffusen Objektfarbe $\overline{O_d}$ reflektiert wird:

$$\overline{C_a} = k_a \cdot I_a \cdot \overline{O_d} \ .$$

Die *diffuse Reflexion* basiert auf dem Phänomen der Streuung. Eintreffendes Licht dringt in den Körper ein und wird von seinen tieferen Schichten gleichmäßig in alle Richtungen gestreut. Das reflektierte Licht hat die diffuse Objektfarbe $\overline{O_d}$. Die Intensität I_e des einfallenden Lichts in einem Punkt ist proportional zum Cosinus des Winkels zwischen der Flächennormale $\mathbf{N}$ in dem Punkt und der Richtung zur Lichtquelle $\mathbf{L}$ (*Cosinusgesetz von Lambert*):

$$I_e \propto \cos(\mathbf{L}, \mathbf{N}) \ .$$

Das ambiente Licht wird hierbei nicht berücksichtigt, weshalb auch von gerichteter Reflexion gesprochen wird. Für einen diffusen Reflektor gilt

$$\overline{C_d} = k_d \cdot I_e \cdot \overline{O_d},$$

wobei k_d angibt, wie stark die diffuse Reflexion ist.

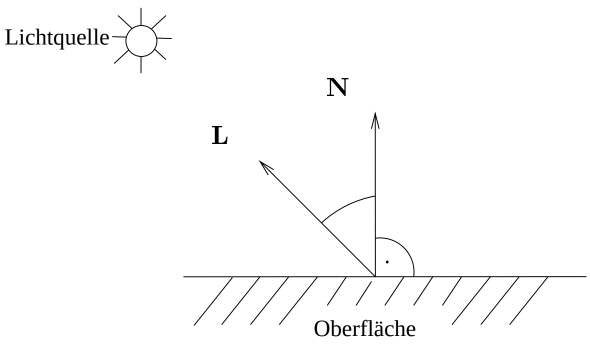


Abbildung 18.5: Cosinusgesetz von Lambert (für diffuse Reflexion)

Die *spekulare Reflexion* beruht auf dem Phänomen der Spiegelung. Eintreffendes Licht wird an der Oberfläche des Objekts gespiegelt (Einfallswinkel = Ausfallswinkel). Das reflektierte Licht hat die spekulare Objektfarbe $\overline{O_s}$, da keine tiefere Wechselwirkung mit dem Objekt stattfindet. Wird das Objekt aus der Reflexionsrichtung betrachtet, so hat es an dieser Stelle einen Glanzpunkt (*Highlight*) in der Farbe $\overline{O_s}$.

Natürliche Materialien sind zumeist keine perfekten Spiegel und strahlen deshalb nicht nur genau in die Reflexionsrichtung, sondern auch mit abnehmender Intensität in andere Richtungen. Von Phong stammt das Modell, daß die abgestrahlte Lichtmenge exponentiell mit dem Cosinus zwischen Reflexionsrichtung $\mathbf{R}$ und Betrachterrichtung $\mathbf{A}$ abnimmt. Der spekulare Exponent O_e bestimmt dabei den Öffnungswinkel des Streukegels um $\mathbf{R}$, unter dem viel Licht reflektiert wird. Für $O_e \leq 5$ ergeben sich große, für $O_e \gg 10$ kleine Streukegel.

Für einen spekularen Reflektor gilt:

$$\overline{C_s} = k_s \cdot I_e \cdot \overline{O_s} \cdot \cos(\mathbf{R}, \mathbf{A})^{O_e} \ .$$

Dabei ist I_e die Intensität des einfallenden nicht-ambienten Lichts. k_s gibt an, wieviel von I_e gespiegelt wird.

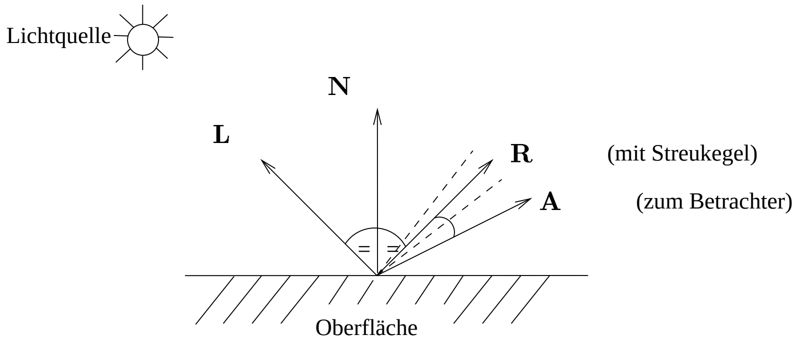


Abbildung 18.6: Spekulare Reflexion nach Phong

18.1.4 Materialeigenschaften

Um im Rendering-Programm Materialeigenschaften nachzuahmen, müssen vom Benutzer insbesondere k_a, k_d und k_s sinnvoll eingestellt werden. Damit ein Objekt nicht mehr Licht aussendet als es empfängt, sollte grundsätzlich $0 \leq k_a, k_d, k_s \leq 1$ gelten, für die Reflexion von gerichtetem Licht zusätzlich $k_d + k_s \leq 1$. Wird k_a im Verhältnis zu k_d und k_s sehr groß gewählt, so ist der beleuchtete Gegenstand sehr kontrastarm. Für matte Gegenstände sollte $k_d \gg k_s$, für spiegelnde $k_s > k_d$ gelten. Je größer O_e ist, um so ähnlicher wird das Reflexionsverhalten dem eines idealen Spiegels.

18.2 Schattierungsalgorithmen

Nach den Grundlagen über die Beleuchtung und die Rasterung von Flächen werden in diesem Abschnitt drei Schattierungsalgorithmen vorgestellt, die mit unterschiedlicher Genauigkeit reale Beleuchtungsverhältnisse nachbilden. Hierbei handelt es sich um inkrementelle Verfahren, die die vorgestellten empirischen Beleuchtungsmodelle verwenden. Die Rasterung der Flächen erfolgt nach der Triangulisierung; deshalb heißen diese Verfahren auch *Dreieck-Shading*-Algorithmen.

18.2.1 Flat-Shading

Das einfachste Schattierungsmodell für ebenflächig begrenzte Objekte ist das *Flat-Shading*. Es verwendet konstante Farbwerte für die einzelnen Dreiecke der Oberfläche. Dazu werden die drei Eckpunkte im WC (wegen der erforderlichen Entfernungen) mit dem vorgestellten Phong-Modell beleuchtet. Nach der Abbildung in die Pixelkoordinaten (DC) werden mit dem Rasteralgorithmus alle Pixel der Dreiecksfläche in der Farbe des Mittelwertes der drei Eckfarbwerte gesetzt:

$$\overline{C_i} = \frac{\overline{C_A} + \overline{C_B} + \overline{C_C}}{3} \; \forall P_i \in \Delta(P_A, P_B, P_C) \; .$$

Dabei bezeichnet $\overline{C_i}$ den RGB-Farbwert für Pixel P_i (siehe Abbildung 18.7).

Beim Flat-Shading wird die Oberfläche des Objekts grob schattiert, die Helligkeitsübergänge sind nicht fließend. Die Qualität der erzeugten Bilder hängt vor allem von der Größe der Dreiecke ab. Das Hauptproblem dieses Verfahrens hängt mit der Eigenschaft des menschlichen Auges zusammen, sprunghafte Intensitätsunterschiede verstärkt wahrzunehmen. Der Randbereich einer helleren Fläche erscheint heller und der Randbereich einer angrenzenden dunkleren Fläche dunkler als der Rest des jeweiligen Polygons. Dieser Effekt wird *Mach-Band-Effekt* genannt.

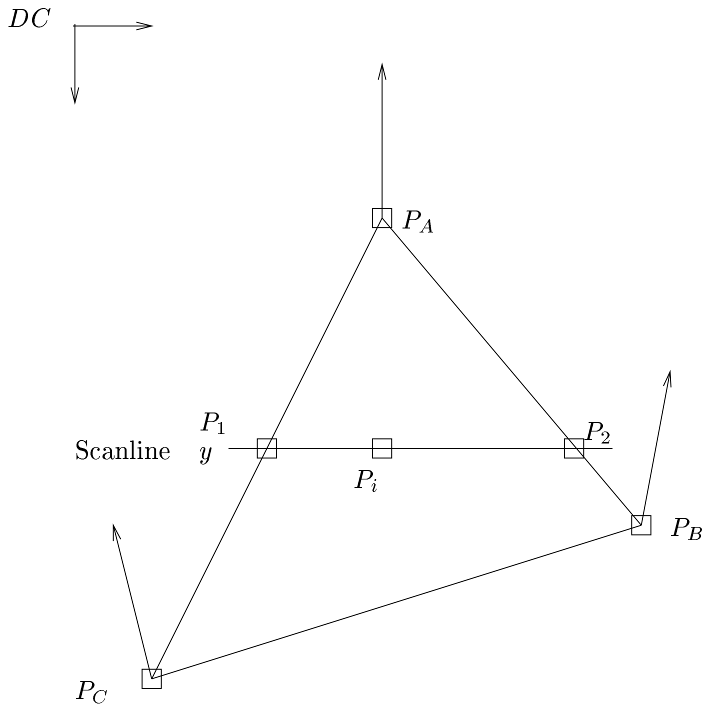


Abbildung 18.7: Dreieck-Shading

18.2.2 Gouraud-Shading

Eine Verbesserung der konstanten Schattierung stellt der Algorithmus von Gouraud dar, der die Farbe eines Pixels im Inneren des Dreiecks durch Interpolation der Eckfarbwerte bestimmt. Dazu werden in diesem scanline-orientierten Algorithmus in den Eckpunkten die Farbwerte ermittelt, die dann zur Interpolation entlang der Polygonkanten verwendet werden.

In der Abbildung berechnen sich die Farbwerte der Pixel $P_1(x_1, y)$ und $P_2(x_2, y)$ wie folgt:

$$\overline{C_1} = \overline{C_A}\frac{y - y_C}{y_A - y_C} + \overline{C_C}\frac{y_A - y}{y_A - y_C}, \quad \overline{C_2} = \overline{C_A}\frac{y - y_B}{y_A - y_B} + \overline{C_B}\frac{y_A - y}{y_A - y_B} \; .$$

Für den Punkt $P_i(x_i, y)$ innerhalb der Fläche ergibt sich die Farbe aus den Randpunkten P_1 und P_2 der Scanline:

$$\overline{C_i} = \overline{C_1}\frac{x_2 - x_i}{x_2 - x_1} + \overline{C_2}\frac{x_i - x_1}{x_2 - x_1} \; .$$

Die Interpolation der Farbwerte ist im entwickelten Programm Teil des Rasteralgorithmus.

Das Gouraud-Shading beseitigt den Mach-Band-Effekt nur zum Teil, da das Auge sogar auf Unstetigkeiten in der zweiten Ableitung der Farbwertkurve in der beschriebenen Art reagiert. Ein weiterer Nachteil liegt in den verformt dargestellten Spiegelungsflächen, die auf die Polygonaufteilung zurückzuführen sind. Dadurch erscheinen die Highlights der spekularen Reflexion auf großen ebenen Flächen evtl. gar nicht oder auf gekrümmten Flächen verzerrt. Eine Möglichkeit, diese unrealistischen Effekte zu begrenzen, besteht in der Verkleinerung der approximierenden Polygone, was einen höheren Rechenaufwand zur Folge hat.

18.2.3 Phong-Shading

Eine zufriedenstellende Lösung der angesprochenen Probleme läßt sich durch den Algorithmus von Phong erzielen. Basierend auf dem besprochenen Beleuchtungsmodell führt dieser Algorithmus die Farbwertberechnung für jedes Pixel der Dreiecksfläche explizit durch. Die dazu benötigten Normalenvektoren müssen zunächst aus den Normalenvektoren in den Eckpunkten berechnet werden. Für die Fläche in der Abbildung ergeben sich die Normalenvektoren in P_1 und P_2 durch lineare Interpolation der in P_A und P_C bzw. P_A und P_B und daraus wiederum die Normalenvektoren entlang der Scanline. Dabei ist zu beachten, daß für die Beleuchtung die Koordinaten und Normalen im WC herangezogen werden.

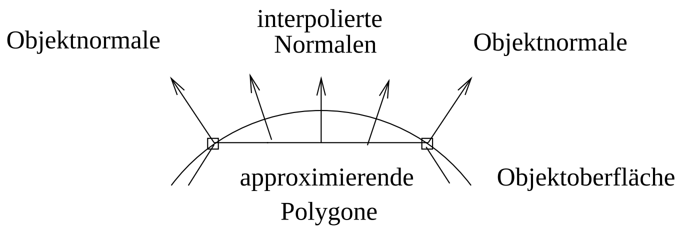


Abbildung 18.8: Interpolation der Normalen beim Phong-Shading

Durch die Interpolation der Normalen ist das Phong-Shading in der Lage, den ursprünglich gekrümmten Verlauf der Oberfläche wiederherzustellen, obwohl das Objekt durch planare Polygone approximiert wird. Dadurch ergibt sich eine fast natürliche spekulare Reflexion mit scharfen Highlights.

Auch mit weniger approximierenden Polygonen ergeben sich bessere Bilder als beim Gouraud-Shading; der Mach-Band-Effekt wird weitgehend unterdrückt. Diese hohe Qualität hat ihren Preis: Statt der Beleuchtung von drei Eckpunkten beim Flat- und Gouraud-Shading müssen beim Phong-Shading alle Pixel des Dreiecks beleuchtet werden, was den Rechenaufwand vervielfacht.

18.3 Schatten

Zur Berechnung von Schatten eignen sich alle *Hidden-Surface*-Algorithmen, da die verdeckten Flächen einer Szene genau den beschatteten Flächen entsprechen, wenn die Position der Lichtquelle und des Betrachters zusammenfallen.

Zunächst wird daher als Phase 1 aus der Sicht der Lichtquelle die Szene in einen Schattentiefenpuffer abgebildet. Phase 2 berechnet dann für den jeweiligen Betrachtungsstandpunkt die Szene mit einem modifizierten Tiefenpuffer-Algorithmus: Ergibt die Überprüfung des z-Wertes mit dem Eintrag `tiefe[x,y]` im Tiefenpuffer, daß dieses Pixel sichtbar ist, so wird der Punkt $P(x, y, z)$ in den Koordinatenraum von Phase 1 transformiert. Ist die z-Koordinate z' des transformierten Punktes P größer oder gleich dem Eintrag $[x', y']$ im Schattentiefenpuffer, dann liegt der Punkt P nicht im Schatten, andernfalls liegt er im Schatten, und seine Intensität muß entsprechend reduziert werden.

18.4 Texture Mapping

Zur realistischen Gestaltung von Oberflächen verwendet man ein zweidimensionales Musterfeld (*texture map*), aus dem für jedes Pixel die zugehörige Farbe ermittelt werden kann. Zugrunde gelegt sind zwei Abbildungen

Musterraum	(u, v)	
$\downarrow$		Abbildung des Musters
Objektraum	(x_0, y_0, z_0)	
$\downarrow$		Projektion
Bildraum	(x, y)	

Die Verknüpfung der zugehörigen inversen Transformationen liefert die zum Einfärben eines Pixels benötigte Information.

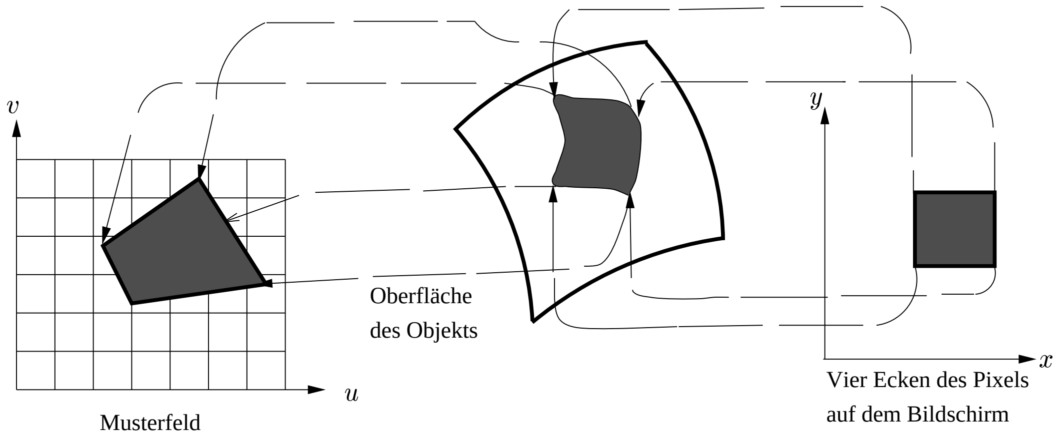


Abbildung 18.9: Transformationskette beim Texture Mapping

Zum Einfärben eines Pixels im Bildraum wird die gewichtete Durchschnittsintensität aller überdeckten Pixel im Musterraum benutzt.

Für spezielle Objekte wie stark gemasertes Holz ist die Beschreibung über Musterfelder zu speicheraufwendig, so daß prozedural definierte Muster verwendet werden.

Um den Eindruck von rauhen Oberflächen zu erwecken, speichert man im Musterfeld Zusatzinformationen über die Veränderung des Normalenvektors (*bump mapping*).

18.5 Beispiel-Applet zum Rendering

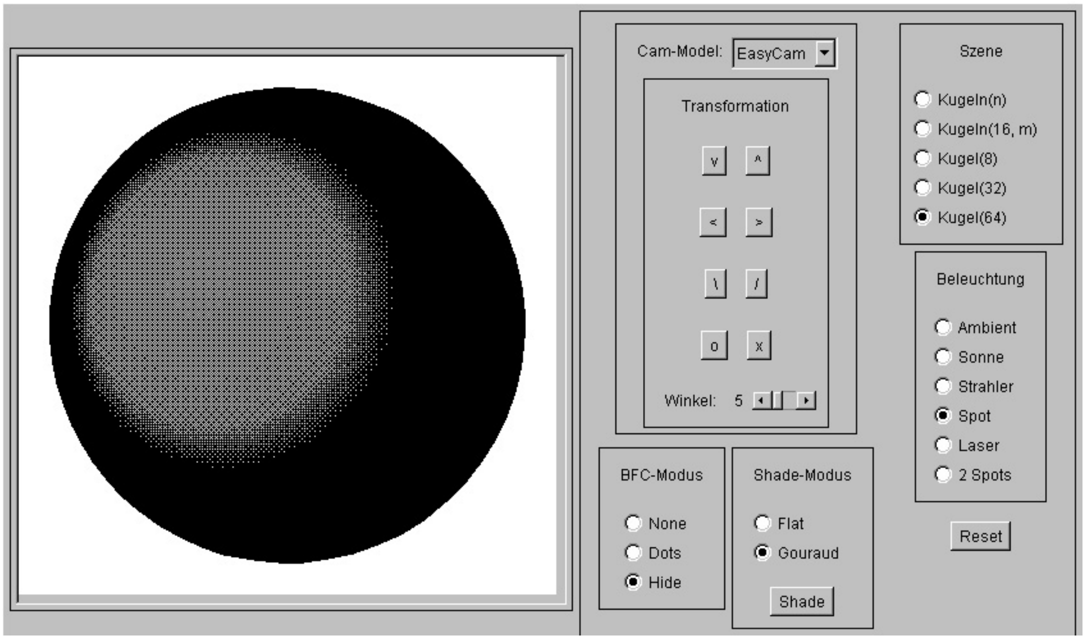


Abbildung 18.10: Screenshot vom 3D-solid-Applet

Kapitel 19

Die Viewing-Pipeline im Überblick

Hier wird im Überblick der gesamte Rendering-Prozeß schematisch dargestellt. Diese sogenannte *Viewing-Pipeline* beinhaltet zusammengefaßte Transformationen und zeigt die einzelnen Algorithmen in den jeweiligen Koordinatensystemen. Die Abweichung vom Vorgehen entlang der Transformationspipeline erklärt sich durch die Einbindung der aufwandsreduzierenden Verfahren Umgebungs-Clipping, Clipping und Back-Face Culling. Diese verringern die Anzahl und die Größe der Polygone, für die die aufwendigen Schattierungsalgorithmen durchgeführt werden müssen.

Durch die Einführung von Umgebungsboxen (Bounding Volumes) wird beim *Umgebungsclipping* zunächst statt eines komplexen Objekts ein umgebender Quader getestet. Ist dieser ganz unsichtbar, wird das Objekt nicht weiter bearbeitet. Ist er vollständig sichtbar, kann auf das (recht zeitaufwendige) Clipping verzichtet werden.

Die Viewing-Pipeline wird auf jedes Objekt der Szene angewandt. Wenn durch das Umgebungs-Clipping festgestellt wird, daß ein Objekt zumindest teilweise im View Window sichtbar ist, beginnt der Algorithmus mit dem Rendern des Objekts. Bei der Objektgenerierung werden die einzelnen Polygone des Objektprototyps im MC sukzessive berechnet und ggf. zusätzlich deformiert. Ein solches Polygon wird danach sofort gemäß der weiteren Viewing-Pipeline behandelt und unter Verwendung des z-Buffer-Algorithmus in den Bildschirmspeicher hineinschattiert. Bei der Liniendarstellung werden die Kanten des Polygons sukzessive gerastert.

Die Linien bzw. Polygone sind nach der Objektgenerierung unabhängig voneinander, so daß sie einzeln berechnet und dargestellt werden können. Die gesamte Szene besteht wegen der Flächenrepräsentation der Objekte praktisch nur aus ihnen. Der z-Buffer-Algorithmus sorgt bei der Flächendarstellung dafür, die Verdeckungseigenschaften pixelweise zu berechnen, so daß nach dem Rendering aller Polygone im Bildschirmspeicher das fertige Bild der Szene steht und ausgegeben werden kann. Die einzelnen Polygone sind sowohl bei der Liniendarstellung als auch trianguliert beim Flat-Shading gut zu erkennen. Beim Gouraud- und besonders beim Phong-Shading tritt der gewünschte Effekt auf, daß sie fließend ineinander übergehen und nicht mehr explizit zu erkennen sind; das Bild wirkt jetzt realistisch.

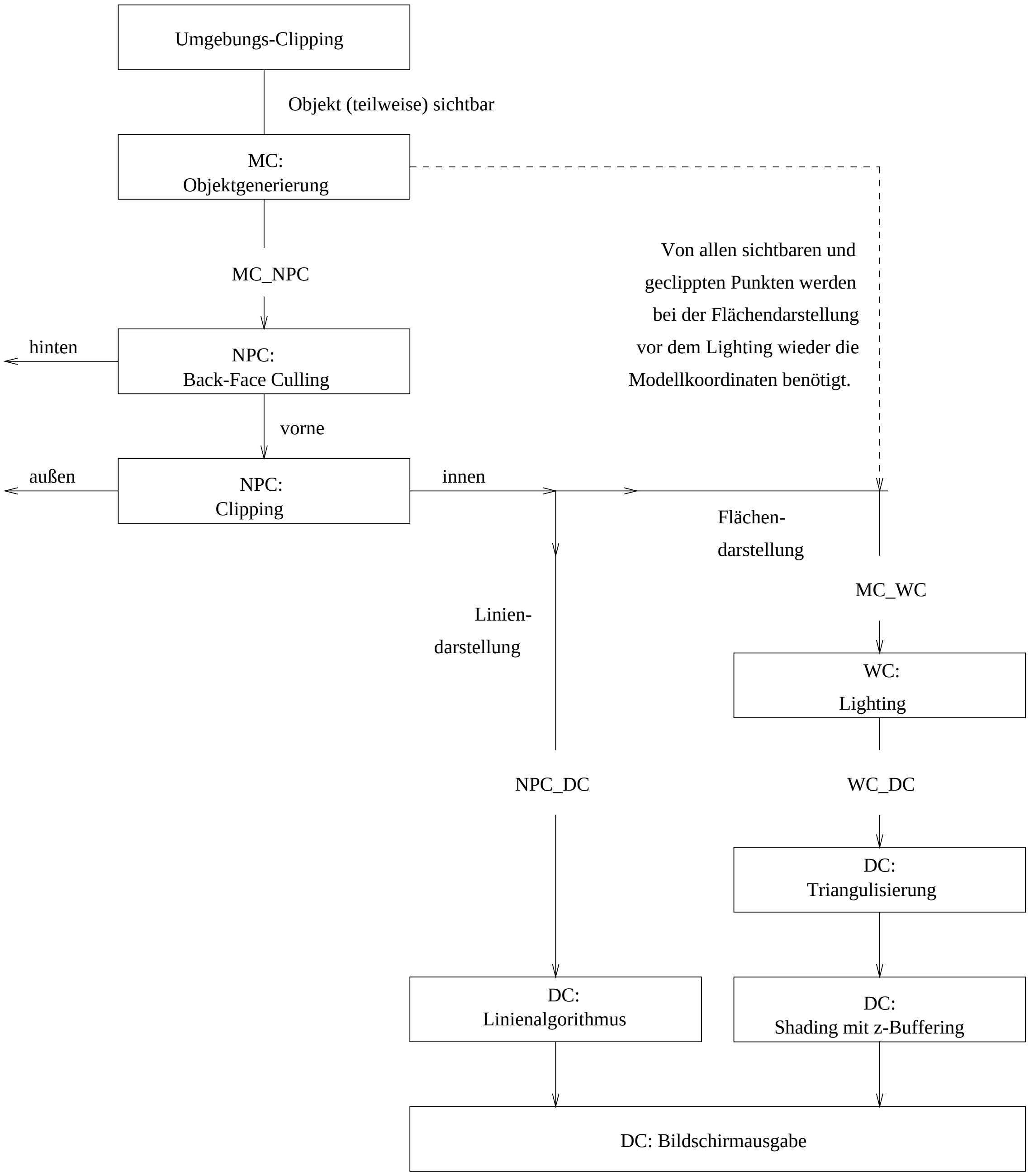


Abbildung 19.1: Komplette Viewing-Pipeline

Kapitel 20

Radiosity

20.1 Globale Beleuchtung

Ein Beleuchtungsmodell berücksichtigt bei der Berechnung der Farbe für einen Punkt das von den Lichtquellen direkt abgegebene Licht und das Licht, das den Punkt nach Reflexion und Transmission durch seine eigene und andere Flächen erreicht. Dieses indirekt reflektierte und hindurchgelassene Licht heißt *globale Beleuchtung*. Als *lokale Beleuchtung* bezeichnet man das Licht, das direkt von den Lichtquellen auf den schattierten Punkt fällt. Bisher wurde die globale Beleuchtung durch einen Term für ambiente Beleuchtung modelliert, der für alle Punkte auf allen Objekten konstant war. Dieser Term berücksichtigte weder die Positionen von Objekt und Betrachter noch Objekte, die das Umgebungslicht blockieren könnten.

In realen Szenen kommt nur ein geringer Teil des Lichts aus direkten Lichtquellen. Es gibt zwei Klassen von Algorithmen zur Erzeugung von Bildern, die die Bedeutung globaler Beleuchtung hervorheben. Das nächste Kapitel behandelt den Ray Tracing-Algorithmus, der neben der Ermittlung sichtbarer Flächen und deren Schattierung gleichzeitig Schatten, Reflexion und Brechung berechnet. Globale spiegelnde Reflexion und Transmission ergänzen dabei die für eine Fläche berechnete lokale spiegelnde, diffuse und ambiente Beleuchtung. Im Gegensatz dazu trennen die in diesem Kapitel behandelten Radiosity-Verfahren die Schattierung völlig von der Ermittlung sichtbarer Flächen. Sie berechnen erst in einem vom Blickpunkt unabhängigen Schritt alle Interaktionen einer Szene mit den Lichtquellen. Dann berechnen sie mit konventionellen Algorithmen zur Ermittlung sichtbarer Flächen und Schattierung durch Interpolation ein oder mehrere Bilder für die gewünschten Standpunkte des Betrachters.

Vom Blickpunkt abhängige Algorithmen (z.B. Ray Tracing) diskretisieren die Bildebene, um die Punkte zu ermitteln, an denen die Beleuchtungsgleichung für eine bestimmte Blickrichtung ausgewertet wird. Beleuchtungsalgorithmen, die vom Blickpunkt unabhängig sind (z.B. Radiosity), diskretisieren dagegen die Szene und verarbeiten sie weiter, um genügend Informationen zur Auswertung der Beleuchtungsgleichung an jedem beliebigen Punkt und für jede Blickrichtung zu erhalten. Die Algorithmen, die vom Blickpunkt abhängig sind, eignen sich gut zur Behandlung von Spiegelungen, die stark vom Standpunkt des Betrachters abhängen. Sie erfordern jedoch bei der Behandlung diffuser Phänomene zusätzlichen Aufwand, da sich diese über große Bildbereiche oder zwischen Bildern aus verschiedenen Blickpunkten wenig ändern. Die Algorithmen, die nicht vom Blickpunkt abhängen, modellieren diffuse Phänomene dagegen effizient, haben aber bei der Behandlung von Spiegelungen einen enorm hohen Speicherbedarf.

20.2 Physikalische Ausgangslage

Dieser Algorithmus geht von der Erhaltung der Lichtenergie in einer geschlossenen Szene aus. Jeder von einer Fläche abgestrahlten oder reflektierten Energie entspricht die Reflexion oder Absorption durch andere Flächen. Die gesamte von einer Fläche abgegebene Energie heißt *Strahlung* oder *Radiosity*. Sie besteht aus der Summe der abgestrahlten Energie, der reflektierten Energie und der Energie, die durch die Fläche hindurchtritt. Ansätze, die die Strahlung der Flächen einer Szene berechnen, heißen daher *Radiosity-Verfahren*. Im Gegensatz zu konventionellen Rendering-Algorithmen berechnen Radiosity-Verfahren erst unabhängig vom Blickpunkt alle Lichtinteraktionen einer Szene. Der Term für das ambiente Licht ist nicht mehr nötig, da sie die Reflexionen zwischen den Objekten genauer behandeln. Dann werden eine oder mehrere Darstellungen gerastert. Dabei fällt nur noch der Aufwand für die Ermittlung der sichtbaren Flächen und für die Schattierung durch Interpolation an.

20.3 Die Radiosity-Gleichung (Beleuchtungsgleichung)

Die bisher betrachteten Schattierungsalgorithmen behandeln die Lichtquellen immer unabhängig von den beleuchteten Flächen. Bei den Radiosity-Verfahren kann dagegen jede Fläche Licht abstrahlen. Daher werden alle Lichtquellen mit einem inhärenten Flächeninhalt modelliert. Die Szene wird in eine endliche Anzahl n diskreter Flächenelemente (*Patches*) zerlegt. Jedes dieser Elemente hat endliche Größe, strahlt über die gesamte Fläche gleichmäßig Licht ab und reflektiert Licht. Wenn man jedes der n Flächenelemente als opaken, diffusen Lambertschen Strahler und Reflektierer betrachtet, gilt für Fläche i

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij}, 1 \leq i \leq n \ .$$

mit

E_i	=	von der Fläche i abgegebene Eigenstrahlung
ρ_i	=	Reflexionsvermögen der Fläche i
n	=	Anzahl der Flächen
F_{ij}	=	Anteil an der von Fläche j abgegebenen Energie, die auf Fläche i auftrifft, genannt *Formfaktor*
B_i	=	gesamte von Fläche i abgestrahlte Energie (Summe aus Eigenstrahlung und Reflexion als Energie pro Zeit und pro Flächeneinheit), genannt *Radiosity* der Fläche i

Die Gleichung besagt, daß die Energie, die eine Flächeneinheit verläßt, aus der Summe des abgestrahlten und des reflektierten Lichts besteht. Das reflektierte Licht berechnet sich aus der Summe des einfallenden Lichts, multipliziert mit dem Reflexionsvermögen. Das einfallende Licht besteht wiederum aus der Summe des Lichts, das alle Flächen der Szene verläßt, multipliziert mit dem Lichtanteil, der eine Flächeneinheit des empfangenden Flächenelements erreicht. $B_j F_{ij}$ ist der Betrag des Lichts, das die ganze Fläche j verläßt und eine Flächeneinheit von Fläche i erreicht.

Zwischen den Formfaktoren in diffusen Szenen besteht eine nützliche Beziehung:

$$F_i \cdot F_{ij} = F_j \cdot F_{ji} \ ,$$

wobei F_i und F_j die Flächeninhalte sind.

Umordnen der Ausdrücke liefert

$$B_i - \rho_i \sum_{1 \le j \le n} B_j F_{ij} = E_i \ .$$

Also kann der Austausch von Licht innerhalb der Flächenelemente der Szene durch ein Gleichungssystem ausgedrückt werden:

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2n} \\ . & . & . & \\ . & . & . & \\ . & . & . & \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1-\rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ . \\ . \\ . \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ . \\ . \\ . \\ E_n \end{pmatrix}$$

Man beachte, daß der Beitrag eines Flächenelements zu seiner eigenen reflektierten Energie berücksichtigt werden muß (es könnte zum Beispiel konkav sein). Daher haben im allgemeinen nicht alle Terme auf der Diagonalen den Wert Eins.

Wenn man die Gleichung löst, erhält man für jedes Flächenelement einen Strahlungswert. Die Elemente können dann für jeden gewünschten Blickpunkt mit einem gewöhnlichen Algorithmus zur Ermittlung sichtbarer Flächen gerastert werden. Die Strahlungswerte sind die Intensitäten dieses Elements.

Gauß-Seidel-Iterations-Verfahren zur Lösung von linearen Gleichungssystemen

Die i-te Gleichung eines linearen Gleichungssystems $Ax = b$ lautet:

$$\sum_{j=1}^{n} A[i,j]x[j] = b[i] \ .$$

Wenn alle Diagonalelemente von A ungleich Null sind, gilt:

$$x[i] = \frac{1}{A[i,i]} \left(b[i] - \sum_{j \ne i} A[i,j]x[j] \right) \ .$$

Das Iterations-Verfahren startet mit einer Schätzung x_1, die auf der rechten Seite eingesetzt wird zur Berechnung von $x_1[1]$. Im nächsten Schritt wird zur Berechnung von $x_1[2]$ bereits $x_1[1]$ benutzt. Ein Iterationsschritt lautet also:

$$x_k[i] = \frac{1}{A[i,i]} \left(b[i] - \sum_{j=1}^{i-1} x_k[j]A[i,j] - \sum_{j=i+1}^{n} x_{k-1}[j]A[i,j] \right) \ .$$

Bei Konvergenz wird das Verfahren nach k Iterationsschritten abgebrochen, wenn $b - Ax_k$ genügend klein geworden ist.

20.4 Berechnung der Formfaktoren

Die Formfaktoren werden bestimmt mit Hilfe der Gleichung

$$F_{ij} = \frac{1}{F_i} \int_{F_i} \int_{F_j} \frac{\cos(\phi_i)\cos(\phi_j)}{\pi r_{ij}^2} b_{ij} dF_j dF_i$$

mit

ϕ_i	$=$	Winkel zwischen Normale auf Fläche i und Verbindungslinie zwischen Fläche i und Fläche j
ϕ_j	$=$	Winkel zwischen Normale auf Fläche j und Verbindungslinie zwischen Fläche i und Fläche j
r_{ij}	$=$	Entfernung zwischen Fläche dF_i und Fläche dF_j
b_{ij}	$=$	Blockierungsfunktion, falls Teile von Fläche j aus der Sicht von Fläche i verdeckt sind.
F_i	$=$	Flächeninhalt von Fläche i

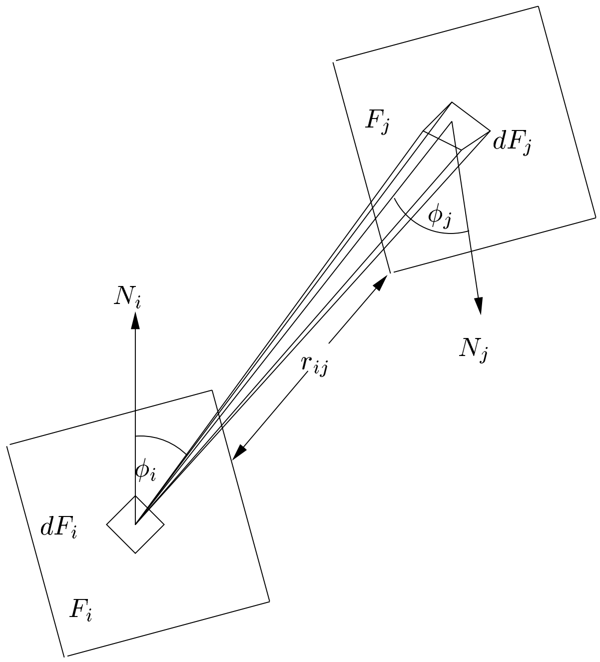


Abbildung 20.1: Der Berechnung von Formfaktoren zugrundeliegende Geometrie

Die geometrische Interpretation beschreibt den Formfaktor als das Verhältnis der Basisfläche einer Halbkugel zur Orthogonalprojektion der auf die Halbkugel projizierten Fläche: Erst projiziert man die von F_i aus sichtbaren Teile von F_j auf eine Halbkugel mit Radius Eins um dF_i, projiziert diese Projektion orthogonal auf die kreisförmige Grundfläche der Halbkugel und dividiert schließlich durch die Kreisfläche. Die Projektion auf die halbe Einheitskugel entspricht in der Gleichung dem Term $\cos\phi_j/r_{ij}^2$, die Projektion auf die Grundfläche entspricht der Multiplikation mit $\cos\phi_i$, und die Division durch den Flächeninhalt des Einheitskreises liefert den Wert π im Nenner.

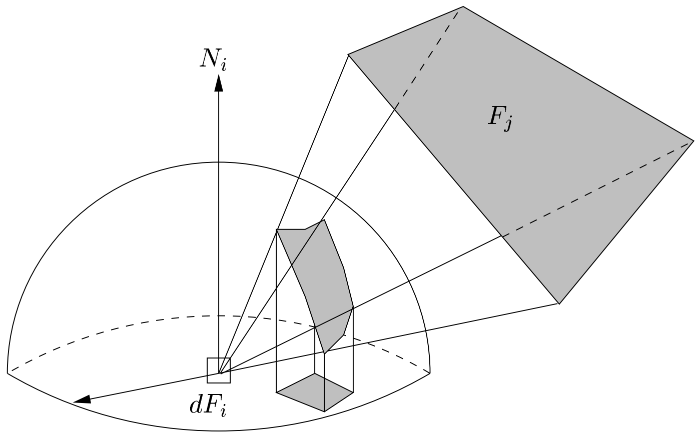


Abbildung 20.2: Geometrische Interpretation des Formfaktors

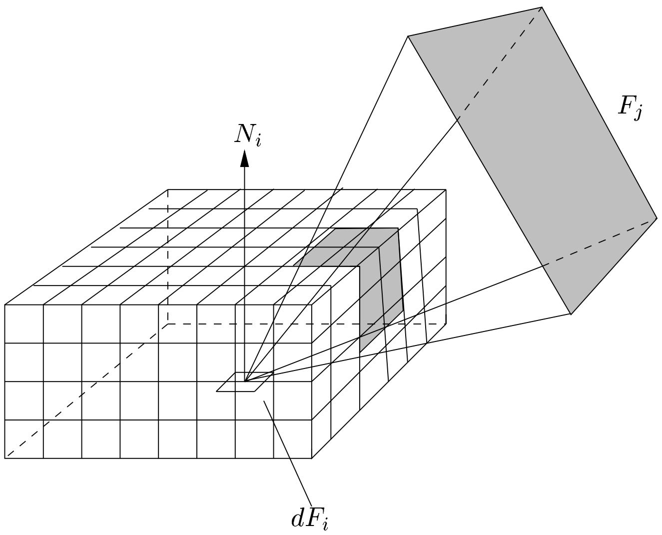


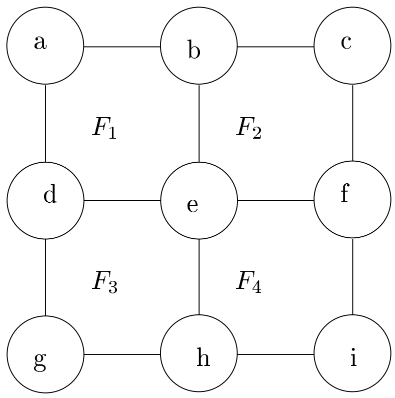
Abbildung 20.3: Simulation der Halbkugel durch Halbwürfel

Zur numerischen Berechnung wird die Halbkugel durch einen *Halbwürfel* (hemi-cube) mit dem Zentrum im Ursprung und dem Normalvektor in der z-Achse ersetzt. Die Oberseite des Würfels ist dabei parallel zur Fläche. Jede Seite des Halbwürfels wird in ein Raster gleich großer quadratischer Zellen aufgeteilt. (Die Auflösungen reichen von 50×50 bis zu mehreren Hundert pro Seite.) Dann wird jedes Flächenelement auf die Seiten des Halbwürfels projiziert. Dazu benutzt man einen Algorithmus, der für jede Zeile die Kennung des nächsten schneidenden Elements speichert. Diese Elementpuffer kann man mit einem z-Puffer-Algorithmus für jede Seite des Halbwürfels berechnen, bei dem man für jede Zelle statt der Schattierung die Kennung der nächstliegenden Fläche speichert. Jeder Zelle des Halbwürfels ist ein *Delta-Formfaktor* zugeordnet, der von der Position der Zelle abhängt und vorher berechnet wird. Für eine beliebig feine Rasterung der Quaderoberfläche ergibt sich der Formfaktor F_{ij} als Summe aller von der Fläche F_j überdeckten Rasterzellen. Wird eine Rasterzelle von mehreren Flächen überdeckt, wird sie der Fläche mit der geringsten Entfernung zugerechnet. Für die Summe aller Formfaktoren einer Fläche F_i gilt

$$\sum_{j=1}^{n} F_{ij} = 1 \ .$$

20.5 Interpolation der Pixelfarben

Für jede Fläche i liege nun ihre *Radiosity* B_i vor. Hieraus interpoliert man die Strahlung für die Eckpunkte:



$$\begin{array}{lcllcl}
B(e) & = & (B_1 + B_2 + B_3 + B_4)/4 & & & \\
B(a) & = & B_1 + (B_1 - B(e)) & B(c) & = & B_2 + (B_2 - B(e)) \\
B(g) & = & B_3 + (B_3 - B(e)) & B(i) & = & B_4 + (B_4 - B(e)) \\
 & & & & & \\
B(b) & = & (B(a) + (B(c))/2 & B(d) & = & (B(a) + B(g))/2 \\
B(f) & = & (B(c) + (B(i))/2 & B(h) & = & (B(g) + B(i))/2
\end{array}$$

Nun wird jedes Patch in zwei Dreiecke zerteilt, so daß nach der Projektion die Strahlungswerte an den Dreiecksecken zur zweifachen Interpolation verwendet werden können.

20.6 Schrittweise Verfeinerung

In Anbetracht des hohen Aufwands beim bisher beschriebenen Radiosity-Algorithmus stellt sich die Frage, ob man die Ergebnisse der Beleuchtungsberechnung inkrementell approximieren kann. Die Auswertung der i-ten Zeile des Gleichungssystems liefert eine Schätzung für die Strahlung B_i des Flächenelements, die auf den Schätzungen für die Strahlungswerte der anderen Flächenelemente basiert. Jeder Term in der Summe der Gleichung beschreibt die Auswirkung des Elements j auf die Strahlung des Elements i:

$$\text{Beitrag von } B_j \text{ zu } B_i = \rho_i B_j F_{ij} \text{ für alle } j \ .$$

Diese Methode "sammelt" also das Licht der restlichen Szene ein. Der Ansatz zur schrittweisen Verfeinerung verteilt dagegen die Strahlung eines Flächenelements auf die Szene. Dazu bietet sich die Anpassung obiger Gleichung zu

$$\text{Beitrag von } B_i \text{ zu } B_j = \rho_j B_i F_{ji} \text{ für alle } j \ .$$

an. Wenn man eine Schätzung für B_i hat, kann man den Beitrag des Flächenelements i zum Rest der Szene ermitteln, indem man vorstehende Gleichung für jedes Element j auswertet. Dazu braucht man leider F_{ji} für alle j. Jeder dieser Werte wird mit einem separaten Halbwürfel bestimmt. Dies erfordert ebensoviel Aufwand an Speicherplatz und Rechenzeit wie der ursprüngliche Ansatz. Man kann die Gleichung jedoch umschreiben, wenn man die Reziprozitätsbeziehung berücksichtigt.

$$\text{Beitrag von } B_i \text{ zu } B_j = \rho_j B_i F_{ij} \frac{F_i}{F_j} \text{ für alle } j \ .$$

Zur Auswertung dieser Gleichung für alle j sind nur die Formfaktoren nötig, die mit einem einzigen Halbwürfel um das Flächenelement i berechnet wurden. Kann man die Formfaktoren des Elements i schnell berechnen (z.B. mit z-Puffer-Hardware), kann man sie wieder löschen, sobald die Strahlungen vom Flächenelement i aus berechnet sind. Man muß also immer nur einen einzigen Halbwürfel und dessen Formfaktoren gleichzeitig berechnen und speichern.
Sobald die Strahlung eines Elements verteilt wurde, wählt man ein anderes Element aus. Ein Element kann wieder Strahlung verteilen, sobald es neues Licht von anderen Elementen erhält. Dabei wird nicht die gesamte geschätzte Radiosity des Elements i verteilt, sondern nur der Betrag ΔB_i, den das Element i seit dem letzten Verteilen empfing. Der Algorithmus läuft so lange weiter, bis die gewünschte Genauigkeit erreicht ist. Es ist sinnvoll, das Element mit der größten Differenz zu nehmen, statt die Elemente in zufälliger Reihenfolge auszuwählen. Man wählt also das Element, das noch am meisten Energie abzustrahlen hat. Da die Strahlung pro Flächeneinheit gemessen wird, wählt man ein Flächenelement, bei dem $\Delta B_i F_i$ maximal ist. Am Anfang gilt für alle Flächenelemente $B_i = \Delta B_i = E_i$. Dieser Wert ist nur bei Lichtquellen ungleich Null.

Im folgenden bezeichnet ΔB_i also die noch nicht verteilte Strahlung, d.h. die Differenz zwischen der *Radiosity* im letzten und im gegenwärtigen Iterationsschritt. Es wird ausgenutzt, daß gilt $F_{ji} = (F_{ij} \cdot F_i)/F_j$.

```
for i:= 1 to n do                          {initialisiere}
   if patch i ist Lichtquelle              {für jede Fläche}
     then Bi := ΔBi := Emissionswert       {Strahlung und Differenz}
     else Bi := ΔBi := 0
end;
```

```
repeat
   for {jede Fläche i, beginnend bei der mit der größten Ausstrahlung} do begin
     Plaziere Hemicube auf Fläche i
     Berechne F_ij für alle 1 ≤ j ≤ n
     for j := 1 to n do begin                {für jede Fläche j tue}
         ΔR := ρ_j * ΔB_i * F_ij * F_i/F_j   {Strahlung von Fläche i}
         ΔB_j := ΔB_j + ΔR                   {Differenz erhöhen}
         B_j := B_j + ΔR                     {Strahlung erhöhen}
     end;
     ΔB_i := 0;                              {Überschuß ist verteilt}
   end
until fertig                                 {bis zur Konvergenz}
```

Bei jeder Ausführung der äußeren FOR-Schleife verteilt ein weiteres Flächenelement seine unverbrauchte Strahlung auf die Szene. Daher werden nach der ersten Ausführung nur die Flächen beleuchtet, die selbst Lichtquellen sind sowie solche, die beim Verteilen der Strahlung des ersten Elements direkt beleuchtet werden. Rastert man am Ende jeder Ausführung des Codes ein neues Bild, so wird das erste Bild relativ dunkel und die nachfolgenden Bilder immer heller.

Abbildung 20.4 faßt den gesamten Ablauf zusammen.

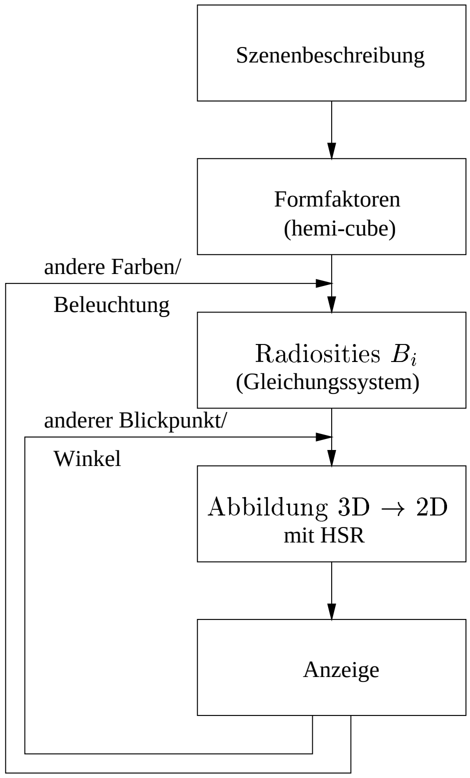
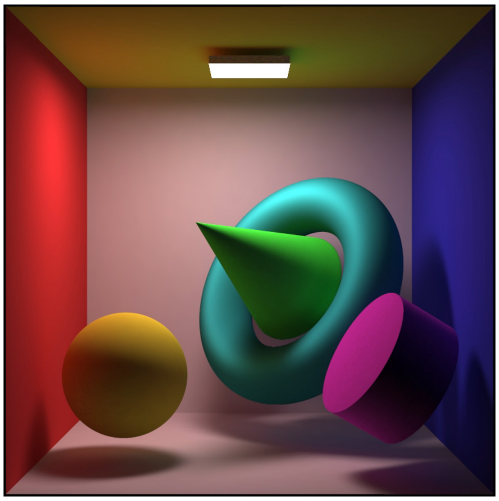


Abbildung 20.4: Ablaufschema beim Radiosityverfahren

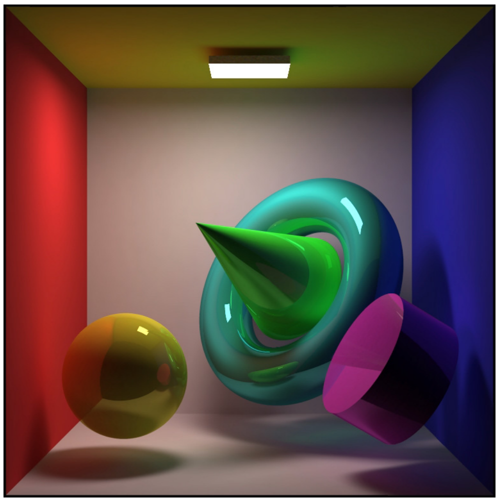
20.7 Screenshots

Auf der Seite `http://www.graphics.cornell.edu/online/research` gibt es einige Beispiele für Radiosity-Bilder.

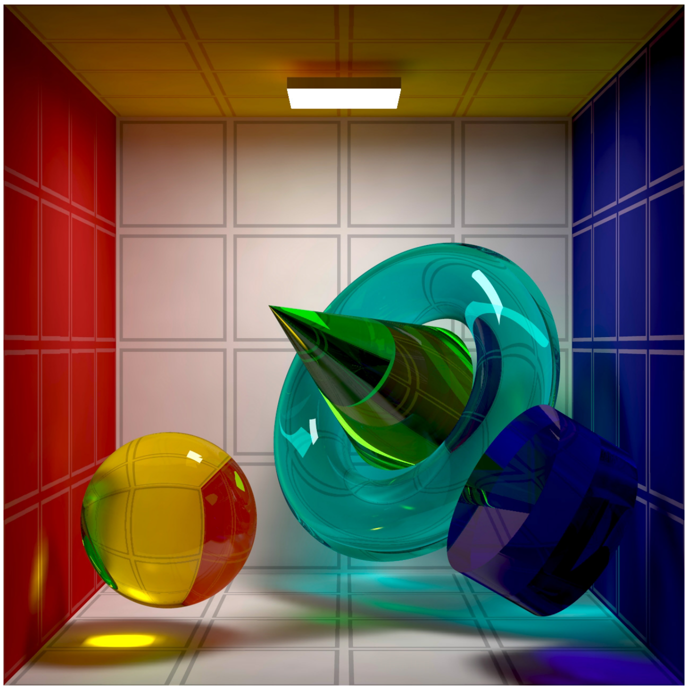
Die nächsten drei Bilder sind Screenshots vom Advanced Rendering Toolkit der Technischen Universität Wien (`http://www.cg.tuwien.ac.at/research/rendering/ART`). Hier wurde das klassische Radiosity-Verfahren um Ray-Tracing-Komponenten erweitert.



Diffuse Reflexion



Diffuse und spekulare Reflektion



Diffuse, spekulare und transparente Reflexion