



Hochschule RheinMain
Fachbereich Design Informatik Medien
Studiengang Informatik

Abschlussarbeit

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

Interaktives Physik-basiertes Rendering

Vorgelegt von Sebastian Dorn
am 30.04.2014
Referent Prof. Dr. Ulrich Schwanecke
Korreferent Prof. Dr. Steffen Reith

Erklärung gemäß ABPO § 6.4.3

Ich erkläre hiermit,

- dass ich die vorliegende Abschlussarbeit selbstständig angefertigt,
- keine anderen als die angegebenen Quellen benutzt,
- die wörtlich oder dem Inhalt nach aus fremden Arbeiten entnommenen Stellen, bildlichen Darstellungen und dergleichen als solche genau kenntlich gemacht und
- keine unerlaubte fremde Hilfe in Anspruch genommen habe.

Wiesbaden, 30.04.2014

Sebastian Dorn

Erklärung zur Verwendung der Master Thesis

Hiermit erkläre ich mein Einverständnis mit den im folgenden aufgeführten Verbreitungsformen dieser Abschlussarbeit:

Verbreitungsform	Ja	Nein
Einstellung der Arbeit in die Hochschulbibliothek mit Datenträger	×	
Veröffentlichung des Titels der Arbeit im Internet	×	
Veröffentlichung der Arbeit im Internet	×	

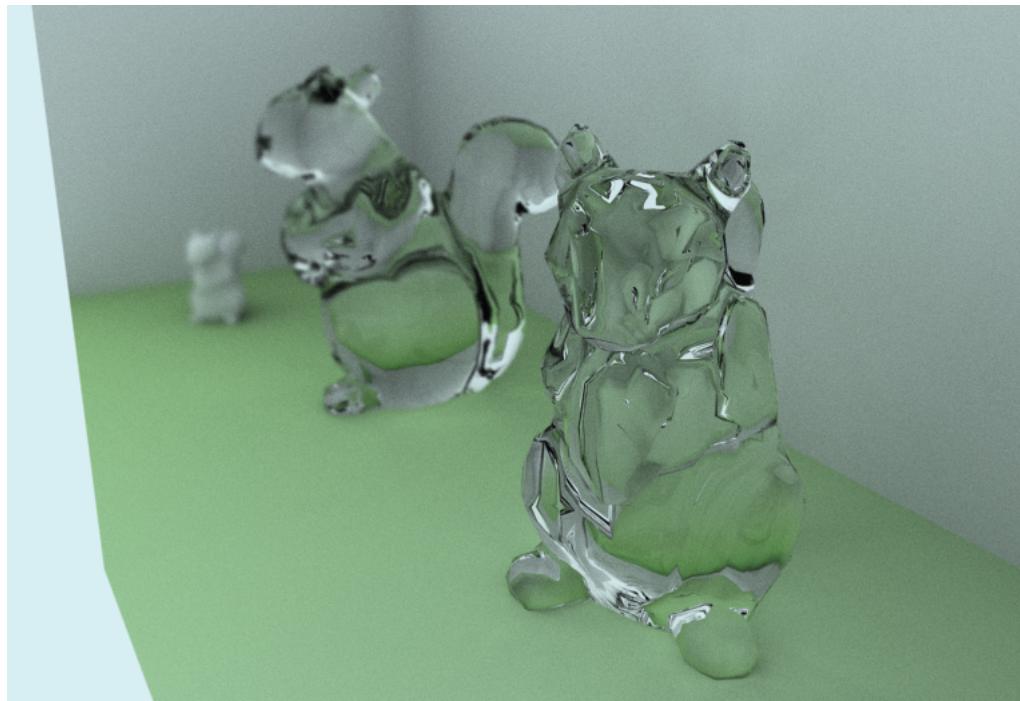
Wiesbaden, 30.04.2014

Sebastian Dorn

Zusammenfassung

Physik-basiertes Rendering beschäftigt sich mit globalen Lichtmodellen für möglichst realitätsgetreue 3D-Grafiken. Die Haupt-Anwendungsbereiche sind in der Architektur- und Filmbranche angesiedelt. In der Spielebranche ist dies noch nicht der Fall, da der Rechenaufwand für optisch ansprechende Bilder bei Erhalt von Interaktivität noch eine zu große Hürde darstellt. Dank technischer Weiterentwicklungen wird es innerhalb der nächsten Hardware-Generationen jedoch zusehends möglich werden.

Ziel dieser Arbeit ist es, einen Physik-basierten Renderer zu entwickeln, mit dem man sich interaktiv in einer 3D-Szene umschauen kann. Globale Beleuchtung wird über *Path Tracing* realisiert, bei dem von der Kamera aus Strahlen in die Szene geschossen werden, um so Lichtinformationen zu sammeln. Farben von Flächen und Licht werden als *Spectral Power Distributions* behandelt, was sie als mehrere Wellenlängen mit verschiedenen Energieanteilen beschreibt. Die Geometrie der Szene wird in einer *Bounding Volume Hierarchy* – einer Baumstruktur – untergebracht, um das Finden von Schnittpunkten der Strahlen mit der Szene zu beschleunigen. Für das Rendern auf der Grafikkarte wird das OpenCL-Framework verwendet.



Inhaltsverzeichnis

1 Einleitung	1
1.1 Einige Grundlagen in Vektorrechnung	2
1.2 Repräsentation von Objekten in der Computergrafik	4
1.3 Zielsetzung	6
2 Verwandte Arbeiten	9
3 Globale Beleuchtungsmodelle	11
4 Repräsentation von Licht	17
4.1 Die Spectral Power Distribution	17
4.2 Umwandlung von SPD zu RGB-Farben	18
5 Path Tracing	25
5.1 Initiale Strahlen vom Auge aus	28
5.2 Lichtreflektion und -brechung	30
6 Bidirectional Rendering Distribution Functions	35
6.1 Schlick	36
6.2 Shirley-Ashikhmin	40
7 Interaktives Rendering	43
8 Implementierung	45
8.1 Zufallszahlen	47
8.2 Implizite Pfade	49
8.3 Berechnung der Lichtpfade	49
8.4 Antialiasing	60
8.5 Datei-Formate der Modelle	61
8.6 Darstellung eines Frames mit OpenGL	63
8.7 Generelle Optimierungen für OpenCL	63
9 Evaluation	69
10 Zusammenfassung und Ausblick	79

1 Einleitung

Realistisches Rendern ist vor allem im Bereich Film schon länger von Bedeutung. Sei es, um Objekte nachträglich in eine aufgenommene Szene einzufügen oder ganze 3D-Filme in immer besserer Grafik zu präsentieren. Das Rendering findet allerdings nicht in Echtzeit statt, sondern die Berechnung einzelner Frames kann mitunter Stunden brauchen – was jedoch auch auf einen besonders hohen Anspruch an Bildqualität zurückzuführen ist. Dank Fortentwicklungen im Hardwarebereich – Grafikkarten im Besonderen –, werden realistische Beleuchtungstechniken (Global Illumination) nun auch für Echtzeitanwendungen interessant, wie beispielsweise Spiele. So verfügt die Game Engine *Unreal Engine 4* in ihrer PC-Version als einer der Vorreiter bereits über Global Illumination-Techniken [Mit12].

Der Wunsch nach immer besseren Beleuchtungstechniken stammt vor allem daher, dass glaubhafte Lichteffekte einer der Hauptfaktoren sind, um eine Szene realer erscheinen zu lassen. Bereits einfache lokale Licht- und Schatteneffekte – wie abgedunkelte Flächen auf einem Würfel – erzeugen einen Tiefeneffekt für die 3D-Raumwahrnehmung. Lokale Beleuchtungsmodelle haben den Vorteil, dass sie schnell berechnet werden können. Dafür lassen sie viele Lichteffekte missen, wie z.B. Color Bleeding (Farben strahlen auf andere Flächen ab), Kaustiken (Lichtbündelungen) oder Verdeckung anderer Objekte, so dass diese im Schatten liegen. Die gängigen Rasterisierungsverfahren sind aber nur bedingt für Globale Beleuchtung tauglich. Für immer realistischere Bildqualität werden immer komplexere und aufwendigere Techniken eingesetzt, was jedoch die Anforderungen an die Hardware steigen lässt. Mittlerweile erreicht die Entwicklung einen Punkt, wo es sinnvoller erscheint, auf richtige Globale Beleuchtungsmodelle wie Path Tracing umzusteigen.

Einen simplen Path Tracer zu entwickeln ist auch nicht übermäßig kompliziert, wie Projekte wie *smallpt*¹ beweisen. Ziel des Projekts ist es, mit möglichst wenig Code einen simplen Path Tracer umzusetzen. Daher bietet es einen guten Einstiegspunkt für Interessierte, die neben der Theorie auch gerne ein vollständiges Code-Beispiel sehen möchten.

Neben den Unterschieden von lokalen zu globalen Modellen und der Frage nach Echtzeit, spielt auch die Repräsentation des Lichts eine Rolle. Traditionell wird in der Computergrafik Licht als RGB-Wert behandelt, da dies auch der Darstellungsform von Monitoren und dem Speicherformat von Bilddateien entspricht. Für eine korrekte Handhabung von Lichteffekten ist es jedoch unabdingbar eine physikalisch korrekte Repräsentation zu wählen, was bedeutet, dass auch der Wellen-Aspekt von Licht betrachtet werden muss.

¹<http://www.kevinbeason.com/smallpt/>

Notation in dieser Arbeit. Den Sprachgebrauch und mathematische bzw. algorithmische Notationen betreffend gelten folgende Regeln:

- Matrizen sind groß geschrieben, kursiv und fettgedruckt, z.B. M
- Vektoren sind klein geschrieben, kursiv und fettgedruckt, z.B. v
- Normalisierte Vektoren haben zusätzlich ein Dach, z.B. \hat{n}
- Das Koordinatensystem ist wie in Abbildung 1.1 zu sehen rechtshändig ausgerichtet.

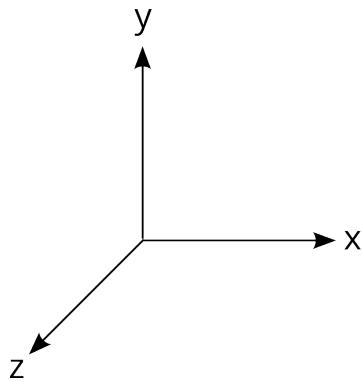


Abbildung 1.1: Ausrichtung der Achsen.

1.1 Einige Grundlagen in Vektorrechnung

Für ein besseres Verständnis der Arbeit ist es hilfreich, einige Grundlagen der Vektorrechnung aufzufrischen. Im folgenden wird nur der 3-dimensionale Fall betrachtet, da die Vektoren im Rest der Arbeit diese Dimension auch nicht übersteigen. Besonders wichtig ist es, sich einmal grafisch zu veranschaulichen, was diverse Operationen mit Vektoren bewirken.

Skalar-Produkt. Im Englischen wird das Skalar-Produkt auch als *dot product* bezeichnet, weshalb in Programmiersprachen die zugehörige Funktion auch meistens `dot(v, w)` lautet. Das Skalar-Produkt zweier Vektoren berechnet sich über die Summe der multiplizierten Komponenten:

$$(v \cdot w) = v_x w_x + v_y w_y + v_z w_z \quad (1.1)$$

Der Zahlenwert, der als Ergebnis entsteht, hat eine konkrete Bedeutung. Dieser entspricht dem Kosinus des Winkels zwischen den beiden Vektoren. Im Fall von Abbildung 1.2 gilt also $(v \cdot w) = \cos \theta$.

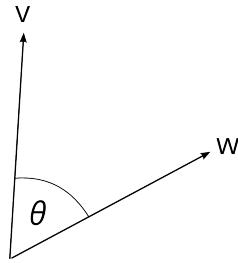


Abbildung 1.2: Das Skalarprodukt entspricht dem Kosinus des Winkels.

Kreuz-Produkt. Im Englischen heißt das Kreuz-Produkt auch *cross product* und Funktionen entsprechend häufig $\text{cross}(v, w)$. Das Kreuz-Produkt zweier Vektoren ist wieder ein Vektor, der definiert ist als:

$$\mathbf{v} \times \mathbf{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \times \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_y w_z - v_z w_y \\ v_z w_x - v_x w_z \\ v_x w_y - v_y w_x \end{pmatrix} \quad (1.2)$$

Der neue Vektor steht orthogonale auf der durch die beiden erzeugenden Vektoren aufgespannten Ebene, wie in Abbildung 1.3 dargestellt.

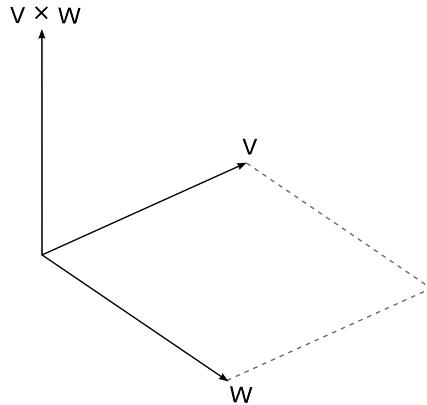


Abbildung 1.3: Das Kreuzprodukt ist ein Vektor, der orthogonale zu den erzeugenden Vektoren liegt. Die gestrichelten Linien sollen bei der räumlichen Vorstellung der Ebene helfen.

Halb-Vektor oder Winkelhalbierender Vektor. Für zwei gegebene Vektoren \hat{v}, \hat{w} existiert ein Vektor \hat{h} , sodass gilt $(\hat{h} \cdot \hat{v}) = (\hat{h} \cdot \hat{w})$ bzw. $\angle \hat{h}, \hat{v} = \angle \hat{h}, \hat{w}$. Den Vektor \hat{h} erhält man über:

$$\hat{h} = \frac{\hat{v} + \hat{w}}{|\hat{v} + \hat{w}|} \quad (1.3)$$

wobei $|u|$ die Notation für die Länge eines Vektors ist. Abbildung 1.4 zeigt eine räumliche Darstellung.

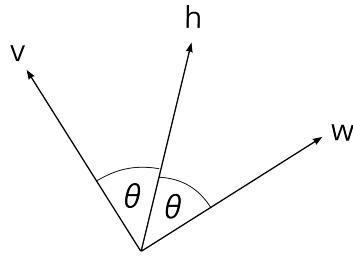


Abbildung 1.4: Winkelhalbierender Vektor.

1.2 Repräsentation von Objekten in der Computergrafik

Ziel der Computergrafik ist es, Objekte aus der Realwelt digital abzubilden. Die Geometrie eines Objektes kann auf verschiedene Arten repräsentiert werden. Als Standard hat es sich dabei durchgesetzt, die Fläche eines 3D-Modells in Dreiecke aufzuteilen. Ein Fläche, die aus zusammenhängenden Polygonen besteht, wird auch als Mesh bezeichnet. Als Polygon das Dreieck zu wählen, hat zum Vorteil, dass die meisten Verfahren – z.B. zum Verformen oder zum Testen von Kollisionen mit anderen Objekten – sich am effizientesten mit Dreiecken durchführen lassen. Ausschlaggebend ist hierbei, dass alle Eckpunkte eines Dreiecks in der selben Ebene liegen, was bei mehr als drei Eckpunkten nicht mehr als gegeben angenommen werden kann. Das Dreieck ist zudem die einfachste Form, die ein Polygon annehmen kann – ein Polygon mit weniger als drei Eckpunkten existiert nicht. Alle höhergradigen Polygone, wie Vierecke, Fünfecke usw., lassen sich in Dreiecke aufteilen. Es ist egal, aus welcher Art von Polygonen ein 3D-Modell besteht, es lässt sich in Dreiecke umwandeln. Abbildung 1.5 zeigt ein 3D-Modell.

Zur Speicherung von 3D-Daten gibt es zahlreiche Formate. Ein weit verbreitetes ist das OBJ-Format. Die Geometrie des Objektes wird dabei in Vertices, Vertex-Normalen und Faces aufgeteilt. Ein Vertex (pl. Vertices) ist ein Punkt im Raum – im Fall von 3D handelt es sich also um eine X-, Y-, und Z-Koordinate. Ein Face beschreibt ein Dreieck, indem es die drei Vertices auflistet, die zusammen die Eckpunkte des Dreiecks bilden. Schließlich gibt es noch die Vertex-Normalen, die die Ausrichtung eines Vertex angeben, woraus sich dann auch die Orientierung eines Face ableiten lässt – man kann also beschreiben, welches die Vorder- und welches die Rückseite eines Dreiecks ist.

Neben der Geometrie hat ein Objekt aber noch mehr Eigenschaften. *Ist die Oberfläche rau oder glatt? Welche Farbe hat es? Ist es transparent und bricht Licht? Zusammengefasst lautet die Frage: Aus was für einem Material besteht das Objekt?* Dererlei

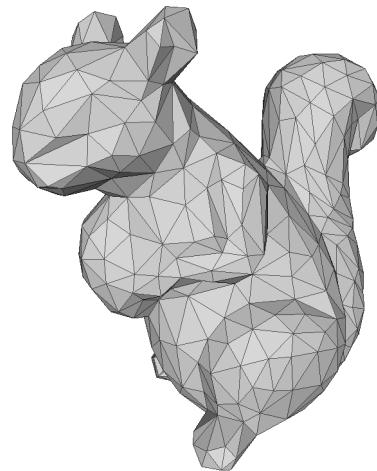


Abbildung 1.5: Eichhörnchenmodell in 3D aus Dreiecken.

Informationen werden in einer MTL-Datei gespeichert. Darin werden verschiedene Materialien jeweils mit einem eigenen Namen definiert und dann in der OBJ-Datei den gewünschten Faces zugeordnet. Listing 1.1 und 1.2 zeigen beispielhafte Auszüge aus einer zusammengehöriger OBJ- und MTL-Datei. In der OBJ-Datei wird ein Objekt squirrel1k-right definiert, dass aus dem Material Squirrel-right besteht. In der MTL-Datei wird das Material dann näher beschrieben – in diesem Fall ist es transparent (d 0.0) und bricht Licht (Ni 1.5).

Listing 1.1: Beispiel für einen Auszug aus einer OBJ-Datei.

```
1 o squirrel1k-right
2 v -0.595124 1.395281 0.432642
3 v -0.615749 1.421955 0.380705
4 ...
5 vn 0.343577 0.779595 0.523545
6 vn 0.327677 0.935118 -0.134770
7 ...
8 usemtl Squirrel-right
9 f 137//15 138//16 139//17
10 f 138//16 140//18 141//19
11 ...

---


```

Listing 1.2: Beispiel für einen Auszug aus einer MTL-Datei.

```
1 newmtl Squirrel-right
2     Ni 1.5
3     d 0.0

---


```

1.3 Zielsetzung

Ziel dieser Arbeit war die Entwicklung eines 3D-Renderers, der physikalische Eigenschaften berücksichtigt, um so möglichst korrekte Grafiken zu erzeugen. Die Anwendung soll dabei interaktiv bleiben – der Anwender soll sich in der Szene umschauen können, ohne erst auf das fertig gerenderte Bild warten zu müssen.

Offensichtlich ist es nicht möglich, alle physikalischen Eigenschaften der realen Welt getreu zu simulieren. Deshalb wird sich auf eine Auswahl der als am wichtigsten empfunden begrenzt. Die folgenden Effekte sollen dargestellt werden können.

Weiche Schatten. Weichen Schatten werden auch als Penumbra oder Halbschatten bezeichnet und sind Übergangsbereiche zwischen dem Vollschatten eines Objektes und der unverdeckten, beleuchteten Fläche. Wie es zu weichen Schatten kommt ist in Abbildung 1.6 zu sehen: Punkte ohne direkte Sichtlinie zur Lichtquelle werden nur durch indirekte Beleuchtung aufgehellt und sind dadurch dunkler als Punkte, die einen direkten Strahl zum Licht haben. Je weniger ein Punkt von einem Objekt verdeckt wird, desto mehr Strahlen können von diesem Punkt einen Punkt auf der Lichtquelle finden und werden dadurch stärker aufgehellt. So entsteht eine Abstufung in der Helligkeit, was in weichen Schatten resultiert [Wat12].

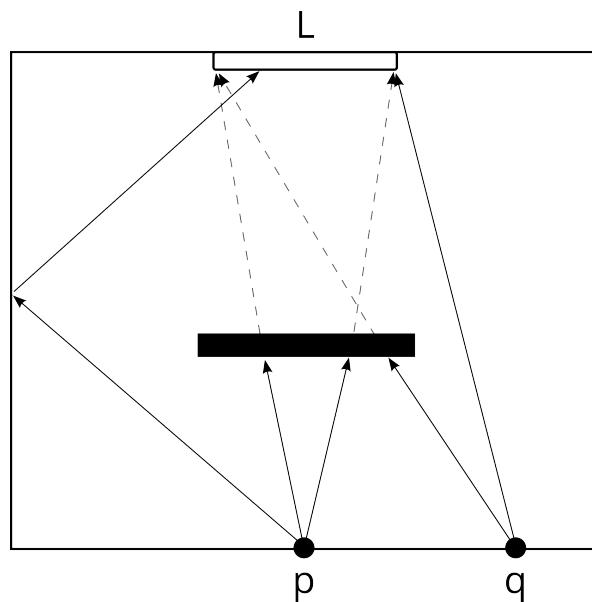


Abbildung 1.6: Punkt p ist verdeckt und erhält nur indirekte Beleuchtung. Punkt q hingegen wird teilweise von der Lichtquelle L beleuchtet.

Lichtbrechung. Ein Beispiel hierfür ist der helle Lichtkreis, der sich im Schatten einer Glaskugel bildet. Dieser Effekt wird auch als Kaustik bezeichnet. In Abbildung 9.7 treffen die Strahlen von der Lichtquelle auf die Glaskugel, werden beim Eintritt in das Material gebrochen, werden beim Austritt erneut gebrochen und landen so näher zusammen auf dem Untergrund, als es ohne die Glaskugel im Weg passiert wäre. Verschiedene Materialien brechen Licht zudem verschieden stark: Ein Diamant bricht Licht z.B. stärker als es bei Glas der Fall ist [GS04].

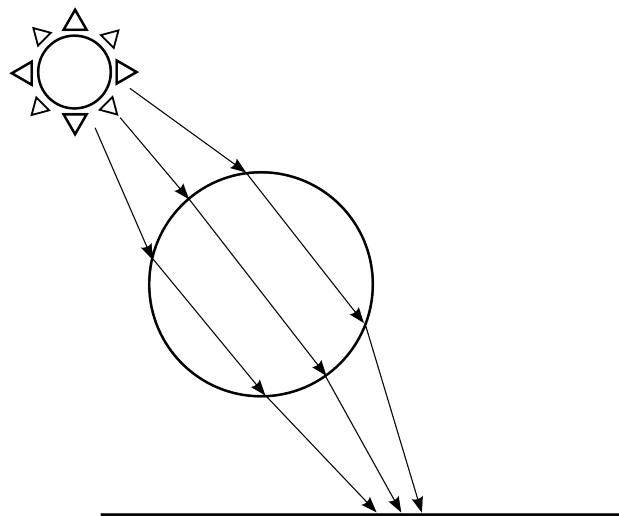


Abbildung 1.7: Strahlen von der Lichtquelle werden beim Eintritt in bzw. Austritt aus der Glaskugel gebrochen und so gebündelt.

Licht als Welle. Der Einfachheit halber wird Licht in der Computergrafik oft als ein Strahl behandelt, der einen bestimmten RGB-Farbwert hat und entsprechend Punkte im Bild einfärbt. In Wirklichkeit ist Licht eine Zusammensetzung verschiedener Wellen, die verschiedene Längen und Amplituden haben. Die Wellenlänge wird als der Farbton wahrgenommen und die Amplitude als die Helligkeit. Materialien erscheinen in einer bestimmten Farbe, indem sie Wellen zu verschiedenen Anteilen absorbieren und den Rest reflektieren [RBA09].

2 Verwandte Arbeiten

Die Idee, Rendering durch das Schießen von Strahlen in die Szene zu realisieren, kam bereits im Jahr 1968 auf [App68]. Seitdem hat sich am ursprünglichen Konzept, von einer Betrachterposition aus Strahlen durch eine Bildebene zu schießen, nichts mehr geändert. Damals war das Rendering auf Schattenwurf beschränkt, ohne indirekte Beleuchtung und unterschiedliche Materialeigenschaften zu berücksichtigen. In [Whi05] wurde Ray Tracing wieder aufgegriffen und erweitert – indirekte Beleuchtung sowie verschiedene Oberflächentypen wie diffus und spekular werden berücksichtigt. Selbst Lichtbrechung in durchsichtigen Objekten wie Glas wurden realisiert.

Mit Online-Ray Tracing beschäftigt sich [WSBW01], um zu erkunden, inwiefern Ray Tracing in Zukunft die bis dahin gängigen Rasterisierungsverfahren ablösen könnte. Dafür wurde ein auf der CPU laufender Ray Tracer entwickelt, der durch Parallelität dank SIMD²-Funktionalität und intelligentes Ausnutzen von Caching durch SSE³ interaktiv bedienbar blieb. Als Beschleunigungsstruktur wurde ein Binary Space Partitioning (BSP)-Baum gewählt, da dessen Durchwandern sowohl schnell möglich ist, als auch parallel für mehrere Strahlen durchgeführt werden kann. Die Resultate bestätigten die Annahme, dass es einen Break-Even-Point zwischen Ray Tracing und Rasterisierungsverfahren gibt, was die Komplexität der Modelle (Anzahl der Dreiecke) betrifft. Während der Rechenaufwand für Ray Tracing mit steigender Komplexität lediglich logarithmisch steigt – vorausgesetzt eine geeignete Datenstruktur zur Unterteilung der Geometrie wird verwendet –, entwickelt sich der Aufwand bei Rasterisierung linear.

Ein GPU-geeignetes Path Tracing Verfahren für Echtzeit-Anwendungen wird in [CNS⁺11] vorgestellt. Im *Voxel Cone Tracing* genannten Verfahren werden zunächst Informationen über Geometrie und direkte Beleuchtung in einem Octree gespeichert. Für das Rendern aus Kamerasicht wird das einfallende Licht über ein Annäherungsverfahren bestimmt: Anstatt Strahlen durch die Hemisphäre über einem Punkt auszusenden und so Lichtpfade zu erzeugen, werden kegelförmige Bereiche definiert, die Lichtinformationen aus dem Octree sammeln. Über Anzahl, Breite und Richtung dieser Trichter lassen sich diffuse, matte und spekulare Flächen simulieren. Beispielsweise lassen sich matte und spekulare Flächen mit wenigen schmalen Trichtern simulieren, die Lichtinformationen in Richtung der Reflexion sammeln. Für diffuse Flächen hingegen werden zum Sammeln mehrere breite Trichter verwendet, ohne eine bestimmte Richtung besonders zu bevorzugen. Verglichen mit einem reinem Ray Tracer produziert Voxel Cone Tracing weniger kor-

²Single Instruction, Multiple Data

³Streaming SIMD Extension

rekte Bilder, dafür bietet es aber einen guten Kompromiss zwischen Korrektheit und Ausführungsgeschwindigkeit für Echtzeit-Anwendungen. Zuletzt wurde die Technik in der Unreal Engine 4 – einer Game Engine – implementiert [Mit12].

Spectral Rendering ist der Fokus in [DCWP02]. Es wird dargelegt, wie sich das Spektrum von Licht als Daten modellieren und dann für die Ausgabe auf einem Monitor wieder RGB-Farbwerte umwandeln lässt. Verschiedene Effekte werden vorgestellt, die für realistische Grafiken berücksichtigt werden müssen, wie Dispersion in Medien (ein bekanntes Beispiel ist das Glas-Prisma, das weißes Licht in sein Farbspektrum aufteilt) und polarisiertes Licht, wie es z.B. für den stereoskopischen Effekt durch 3D-Brillen für 3D-Filme genutzt wird, indem jedes Glas der Brille eine andere Polarisierung herausfiltert. Der Hauptteil des Berichts stellt eine Reihe verschiedener Arbeiten auf dem Gebiet der Tone Reproduction – der korrekten Darstellung von Farben und Helligkeit – vor.

In [RBA09] wird sich speziell mit Rendering unter Verwendung des vollen Spektrums beschäftigt. Gemäß dem Monte Carlo-basierten Rendering-System wird Random Point Sampling zur Repräsentation des Spektrums vorgeschlagen. Es zeigt sich, dass der Mehraufwand durch Rendering mit einem vollem Spektrum gegenüber RGB-Werten desto geringer ausfällt, je komplexer die Szene ist. Gerade bei auf Path Tracing beruhenden Verfahren überschattet der Aufwand für die Pfadfindung die vom Umfang der Szene unabhängige Auswertung der Spektren.

3 Globale Beleuchtungsmodelle

Im Gegensatz zu lokalen Beleuchtungsmodellen berücksichtigen globale Beleuchtungsmodelle alle Gegebenheiten der Szene: Eine Lichtquelle kann durch ein Objekt verdeckt werden, wodurch ein Schatten auf ein anderes Objekt geworfen wird; Licht kann von Flächen reflektiert werden und dadurch indirekt ein Objekt beleuchten. Mathematisch beschrieben wird dieses korrekte Lichtverhalten durch die Render-Gleichung [Kaj86], welche das ausgehende Licht an einem Punkt als eine Kombination aus dem selbst ausgestrahlten Licht des Materials und dem reflektierten Licht allen eingehendes Lichtes beschreibt. Ein globales Beleuchtungsmodell hat nun zum Ziel, diese Gleichung auf irgendeine Art und Weise umzusetzen bzw. eine korrekte Lösung anzunähern. Die Render-Gleichung ist definiert als:

$$L_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out}) = E_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out}) + \int_{\hat{\mathbf{v}}_{in} \in V} R_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out}, \hat{\mathbf{v}}_{in}) L_\lambda(\mathbf{p}, -\hat{\mathbf{v}}_{in}) (\hat{\mathbf{n}} \cdot \hat{\mathbf{v}}_{in}) d\hat{\mathbf{v}}_{in} \quad (3.1)$$

wobei gilt:

- λ ist die Wellenlänge des Lichts.
- \mathbf{p} ist der gerade betrachtete Punkt.
- V ist die Menge aller möglichen Richtungen, aus denen Licht einfallen kann.
- $\hat{\mathbf{v}}_{in} \in V$ ist die Richtung, in die das Licht auf den Punkt einfällt.
- $\hat{\mathbf{v}}_{out}$ ist die Richtung, in die das Licht den Punkt verlässt.
- $\hat{\mathbf{n}}$ ist Normale der Fläche in Punkt \mathbf{p} .

$L_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out})$ ist die Energie der Wellenlänge λ , die den Punkt \mathbf{p} in Richtung $\hat{\mathbf{v}}_{out}$ verlässt. Diese setzt sich zusammen aus dem von Punkt \mathbf{p} selbst ausgestrahlten Licht $E_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out})$ und dem aus allen möglichen Richtungen V einfallenden Licht $L_\lambda(\mathbf{p}, -\hat{\mathbf{v}}_{in})$, wobei nicht jede der Richtungen gleich stark gewichtet einfließt.

Die Intensität des Lichtes wird reguliert durch die *Bidirectional Reflectance Distribution Function* (BRDF) $R_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out}, \hat{\mathbf{v}}_{in})$, die dazu da ist, das an einem Punkt ankommende Licht entsprechend der Material-Eigenschaften zu gewichten. Auf einer spekularen Fläche wird dann z.B. Licht stärker gewichtet, wenn es aus der gespiegelten Richtung des Einfallwinkels kommt. Auf einer perfekt diffusen Fläche würden alle Richtungen gleich gewichtet werden.

Das Lambertsche Gesetz ($\hat{\mathbf{n}} \cdot \hat{\mathbf{v}}_{in}$) besagt, dass die Strahlungsstärke eines Strahles vom Winkel abhängt, aus dem dieser abgestrahlt wird. Abbildung 3.1 zeigt den

Zusammenhang: Je geringer der Winkel zwischen der Normalen \hat{n} und dem einfallenden Licht \hat{v} , desto dunkler erscheint die Stelle. Beträgt der Winkel $\theta = 0^\circ$, findet keine Abschwächung statt. Das Lambertsches Gesetz gilt jedoch nicht für alle Oberflächen, sondern nur für perfekt diffuse. Sollte es sich nicht um eine perfekt diffuse Fläche handeln, liegt es an der BRDF in der Render-Gleichung die Lambert-Abschwächung wieder auszugleichen.

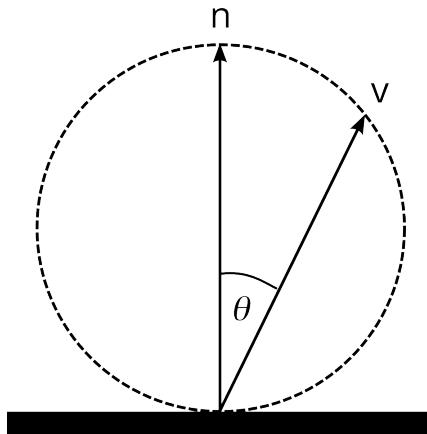


Abbildung 3.1: Abschwächung der Strahlungsstärke nach Lambert, wobei \hat{n} die Normale der Fläche und \hat{v} das einfallende Licht ist.

Eine physikalische Voraussetzung für realistisches Rendering ist zudem die Einhaltung des Energieerhaltungssatzes, der hier besagt, dass nicht mehr Energie einen Punkt verlassen darf, als an ihm ankommt [Sch94]:

$$\forall \hat{v}_{out} \in V : \int_{\hat{v}_{in} \in V} R_\lambda(\mathbf{p}, \hat{v}_{out}, \hat{v}_{in}) (\hat{n} \cdot \hat{v}_{in}) d\hat{v}_{in} \leq 1 \quad (3.2)$$

Während keine Energie unerwartet hinzukommen darf, kann weniger Energie den Punkt verlassen, als ankommt. Der Grund ist, dass Material Teile der Energie absorbiert kann. Ein anschauliches Beispiel ist schwarze Kleidung, die sich unter Sonneneinstrahlung erhitzt: Das einfallende Licht wird zum Großteil absorbiert und zu Hitze umgewandelt. Prinzipiell geht also keine Energie verloren, sondern einfach ausgedrückt wird aus der Helligkeit nur Hitze, was von der BRDF aber nicht mehr berücksichtigt wird. Die Temperatur eines Materials kann auch seine wahrgenommene Farbe verändern – ein extremes Beispiel ist glühendes Metall. Im Rendering wird dieser Effekt in der Regel der Einfachheit halber ignoriert, da sonst zusätzlich zur Verteilung der Lichtstrahlen auch noch Wärmetransfers verfolgt werden müssten.

Praktisch ist es für das Rendering nicht möglich das gesamte Integral auszuwerten, weshalb durch Monte Carlo-Integration das Ergebnis des Integrals angenähert

wird [Laf96]. Monte Carlo-Algorithmen nähern ein Integral dadurch an, dass sie nur eine gleichverteilt zufällige Untermenge an Werten berechnen, aufsummieren und dann anhand der Anzahl an Stichproben mitteln. Das linke Integral $S(f)$ im folgenden Beispiel wird so zu der angenäherten rechten Form von $S_m(f)$, wobei x_1, \dots, x_n eine Menge gleichverteilter Punkte ist:

$$S(f) = \int_0^1 f(x)dx \quad S_m(f) = \frac{1}{n} \sum_{i=1}^n f(x_i) \quad (3.3)$$

Da nicht mehr das gesamte Integral für die BRDF ausgewertet wird, sondern nur noch eine zufällige Unterauswahl, wird eine zusätzliche Gewichtung eingeführt. Dies wird als *Importance Sampling* bezeichnet, bei dem die einzelnen Ergebnisse der BRDF anhand einer Wahrscheinlichkeitsdichtefunktion gewichtet werden. Das Ziel von Importance Sampling ist es, die Varianz im Monte Carlo-Verfahren zu reduzieren. Dadurch verbessert sich die Bildqualität, da eine hohe Varianz als Rauschen im Bild wahrgenommen wird. Als Diskretisierung der Rendergleichung erhält man durch Monte Carlo Integration und die Dichtefunktion $p(\mathbf{p}, \hat{\mathbf{v}}_{out}, \hat{\mathbf{v}}_{in})$:

$$L_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out}) \approx E_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out}) + \frac{1}{n} \sum_{i=1}^n \frac{R_\lambda(\mathbf{p}, \hat{\mathbf{v}}_{out}, \hat{\mathbf{v}}_{in,i})}{p(\mathbf{p}, \hat{\mathbf{v}}_{out}, \hat{\mathbf{v}}_{in,i})} L_\lambda(\mathbf{p}, -\hat{\mathbf{v}}_{in,i}) (\hat{\mathbf{n}} \cdot \hat{\mathbf{v}}_{in,i}) \quad (3.4)$$

Es wird nun also durch die Dichtefunktion geteilt, was bedeutet, dass einige Strahlen zusätzlich stärker gewichtet werden. Die Dichtefunktion soll gezielt die Gewichtung jener Strahlen erhöhen, die in einem Bereich von hoher Bedeutung liegen. Dadurch soll ausgeglichen werden, dass durch das Monte Carlo-Verfahren nur eine begrenzte Menge an Strahlen aus dem Integral berücksichtigt werden.

Nachfolgend sollen einige bekannte und aktuelle Verfahren für Globale Beleuchtung vorgestellt werden. Verschiedene Verfahren haben auch verschiedene Vor- und Nachteile: Manche sind vergleichsweise schnell, dafür aber in ihren Fähigkeiten eingeschränkt; andere führen Verbesserungen für ein existierendes Verfahren ein, sind dafür aber schwerer zu implementieren.

Radiosity berechnet die Verteilung der Lichtstrahlen in der Szene unter Beachtung des Energieerhaltungsgesetzes. Das Verfahren berücksichtigt nicht die Position des Beobachters, was zum Vorteil hat, dass die Lichtverteilung für eine Szene nur einmalig berechnet werden muss. Radiosity ermöglicht zwar indirekte Beleuchtung, ist in seinen Fähigkeiten verschiedene Materialien darzustellen aber auf perfekt diffuse Flächen eingeschränkt [RPJV93].

Ray Tracing funktioniert über das Verfolgen von Lichtstrahlen. Üblicherweise beginnen die Strahlen dabei am Kameraauge und enden an einer Lichtquelle. Ray Tracing wurde im Laufe der Zeit durch verschiedene Techniken erweitert, um mehr Lichteffekte zu berücksichtigen. In seiner Original-Form stoppt der Algorithmus einen Lichtstrahl jedoch bei Erreichen einer diffusen Fläche und sucht abschließend eine Verbindung zu einer Lichtquelle. Dadurch fließt keine indirekte Beleuchtung durch diffuse Flächen in das Bild ein [Whi05].

Photon Mapping ist eine Erweiterung des Ray Tracings. In einem Pre-Processing-Schritt werden Lichtstrahlen (in diesem Kontext: Photonen) von den Lichtquellen aus geschossen. An Oberflächen folgen sie physikalischen Gesetzen wie Reflektion, Brechung und Absorption. Trifft ein Photon auf eine diffuse Fläche, wird es in der sogenannten Photon Map gespeichert. Anhand dieser Photon Map können während des Ray Tracings Informationen über die Lichtenergie an diversen Punkten abgeleitet werden [Jen01].

Path Tracing ist eine verbesserte Variante des traditionellen Ray Tracings. An jeder getroffenen Fläche werden Sekundärstrahlen ausgesendet, wodurch neben den bisher schon möglichen Effekten, wie Reflektion und Brechung, auch indirekte Beleuchtung berücksichtigt wird [Kaj86, LW93].

Bidirektionales Path Tracing ist eine Erweiterung des Path Tracings, bei dem Lichtstrahlen auch von den Lichtquellen ausgehen und mit den erzeugten Pfaden des Auges verknüpft werden. Dies verbessert die Bildqualität von Szenen, in denen die Lichtquellen teilweise verdeckt sind, denn es ist wahrscheinlicher, dass ein Lichtstrahl aus dem verdeckten Bereich der Lichtquelle herauskommt, als dass ein Pfad den Weg am Hindernis vorbei findet [LW93]. Abbildung 3.2 soll den Unterschied aufzeigen. Im linken Bild gehen Strahlen nur vom Auge aus, weshalb es schwer fällt, Pfade zur teilweise verdeckten Lichtquelle zu finden. Die gestrichelten Linien sind Strahlen in Richtung der Lichtquelle, die grauen schaffen es aber nicht, eine Verbindung herzustellen. Im rechten Bild gehen Strahlen auch von der Lichtquelle aus und werden mit den Strahlen vom Auge aus verknüpft, wodurch mehr Pfade zur Lichtquelle zustandekommen. Anstatt zu versuchen, die Lichtquelle selbst zu treffen, werden nun Punkte angezielt, die von der Lichtquelle aus angestrahlt werden. So kommt der blaue Pfad zustande.

Metropolis Light Transport ist ähnlich wie Bidirektionales Path Tracing eine Technik zur Verbesserung der Bildqualität in dunklen Szenen. Das Verfahren basiert

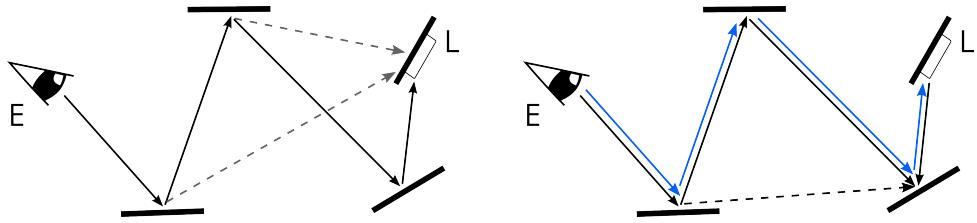


Abbildung 3.2: *Links:* Normales Path Tracing. *Rechts:* Bidirektionales Path Tracing. Die blauen Linien zeigen den zustandegekommen Pfad zur Lichtquelle. Gestrichelte Linien sind indirekte Pfade zur Lichtquelle bzw. in der rechten Grafik ein indirekter Pfad zum abgestrahlten Licht.

darauf, einmal erzeugte Pfade in der nächsten Iteration in geringem Maße zu mutieren und so Variationen des Pfades zu generieren. Pfade, die dabei viel Energie (Licht) führen, werden für den weiteren Verlauf ausgewählt [VG97, CE05].

Energy Redistribution Path Tracing (ERPT) ist ein noch relativ neues Verfahren, das ähnlich wie Metropolis Light Transport Pfade mutiert und auswählt. Die Energie der einzelnen Bildpunkte wird anschließend über die Bildebene verteilt. Der Vorteil gegenüber Metropolis Light Transport ist, dass ERPT eine geringere Varianz erzeugt, was als Bildrauschen wahrgenommen wird, und für direkte Beleuchtung bessere Pfade findet [CTE05, Bat05].

4 Repräsentation von Licht

Bereits im normalen Sprachgebrauch spricht man von „Lichtstrahlen“. Auch in der Computergrafik wird Licht in der Regel als ein Strahl behandelt, der sich durch den Raum bewegt. Für die Bewegung von Fläche zu Fläche wird auch hier dieses Strahlenmodell aus der geometrischen Optik verwendet. Auf die Geschwindigkeit von Licht wird jedoch keine Rücksicht genommen. In der Simulation wird Licht daher so behandelt, als bewege es sich instantan von einem Punkt zum anderen.

Zur Auswertung der Farbe an getroffenen Oberflächen und der Farbe einer Lichtquelle selbst wird das Licht hingegen als Wellen betrachtet. Das für den Menschen sichtbare Spektrum liegt ungefähr im Bereich von 380 bis 780 nm. Wie in Abbildung 4.1 zu sehen, existiert nicht zu jeder vom Menschen wahrnehmbaren Farbe auch eine entsprechende Wellenlänge. Zum Beispiel ist weißes Licht keine einzelne Welle, sondern eine Ansammlung aller Wellen im Spektrum – genauso wie in der Computergrafik ein RGB-Wert von (255, 255, 255) ein Weiß voller Helligkeit ist. Licht muss also als eine Kombination verschiedener Wellen verschiedener Intensität modelliert werden.

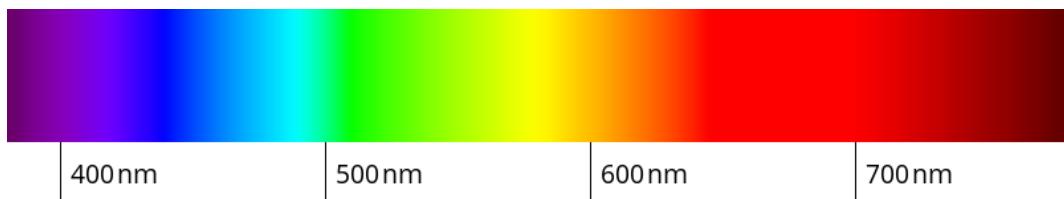


Abbildung 4.1: Das sichtbare Lichtspektrum.

4.1 Die Spectral Power Distribution

Ein Modell, das Licht als Kombination verschiedener Wellen repräsentiert, ist die *Spectral Power Distribution* (SPD). Den Wellenlängen werden dabei je ein Wert zwischen 0 (keine Intensität) bis 1 (volle Intensität) zugewiesen. Wie viele Wellenlängen man in die SPD aufnimmt, ist dabei der Abwägung zwischen Genauigkeit und Berechnungszeit überlassen. Typische Schritte für eine hohe Genauigkeit sind 5 bis 10 nm, während im Interesse der Geschwindigkeit auch bis zu 100 nm gewählt werden können. Die SPD kann nicht nur zur Beschreibung der Farbe und Helligkeit einer Lichtquelle verwendet werden, sondern auch für die Absorptionseigenschaften von Materialien – welche Wellenlängen wie stark absorbiert bzw. reflektiert werden.

Abbildung 4.2 zeigt ein SPD-Diagramm für die Lichtquelle „CIE Illuminant D65“. Die Internationale Beleuchtungskommission (CIE) hat die Werte für diesen Illumi-

nant veröffentlicht, der dem Tageslicht eines Mittaghimmels entspricht. SPDs für reale Materialien sind leider nur schwer aufzufinden, da Messungen sehr aufwendig sind und Spezialwerkzeug erfordern. Erschwerend kommt hinzu, dass Materialien als gleichfarbig wahrgenommen werden können, in Wirklichkeit aber verschiedene SPDs haben. Beispielsweise kann Orange über die Wellenlänge 610 nm als auch aus einer Kombination aus Gelb und Rot definiert werden.

Neben der Repräsentation einer SPD als Auflistung von Wellenlängen mit zugehöriger Energie (*Point Sampling*), gibt es auch noch andere Formen. So kann eine SPD auch als polynomiale Funktion beschrieben werden. Dadurch wird zwar weniger Speicher benötigt und Werte können für beliebige Wellenlängen bestimmt werden, dafür können SPDs jedoch weniger genau beschrieben werden. Besonders plötzliche Ausschläge bei bestimmten Wellenlängen können nur schwer korrekt abgebildet werden. Um diesen Schwachpunkt zu eliminieren, existiert noch ein hybrides Verfahren. Hierbei wird die Grundlinie eines SPDs als Funktion beschrieben, während Stellen, die stark abweichen, über Point Sampling hinzugefügt werden [DCWP02].

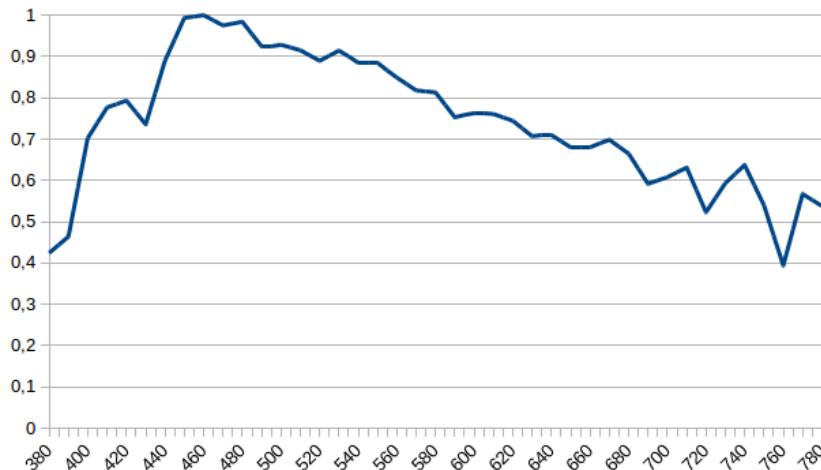


Abbildung 4.2: Die SPD für Illuminant D65. Die X-Achse repräsentiert die Wellenlänge und die Y-Achse den zugehörigen, normierten Energie-Wert. Werte aus [Int07].

4.2 Umwandlung von SPD zu RGB-Farben

Zur Berechnung der Farben wird Licht als Spectral Power Distribution repräsentiert. In diesem Abschnitt wird nun dargelegt, wie aus einer SPD wieder ein RGB-Wert für das zu erzeugende Bild wird.

Von SPD zu XYZ. Während des Renderings wird Licht als Wellen behandelt. Für die Ausgabe auf dem Monitor wird jedoch ein RGB-Farbwert benötigt. Die SPD, die letzten Endes an einem Pixel ankommt, muss daher umgewandelt werden. Dies geschieht in mehreren Schritten. So wird die SPD zuerst in den CIE-XYZ-Farbraum übertragen (siehe Abbildung 4.3) [Wal96].

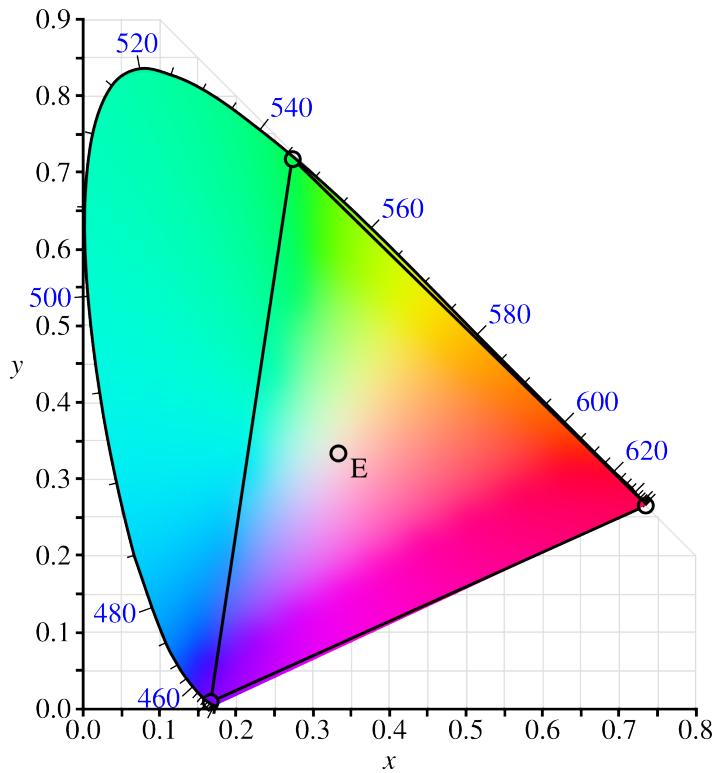


Abbildung 4.3: Das CIE-Farbsegel. Innerhalb des Dreiecks sind die darstellbaren Farben. Der Punkt E ist der Weißpunkt im Farbsystem. Quelle: ⁴

Der CIE-XYZ-Farbraum umfasst alle vom Menschen wahrnehmbaren Farben und stellt eine Beziehung zwischen der Farbwahrnehmung und den physikalischen Ursachen der Farbreize her – sprich: den Längen und Intensitäten der Lichtwellen.

Die Funktionen $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ und $\bar{z}(\lambda)$ sind die CIE-Beobachter-Funktionen, die für eine gegebene Wellenlänge λ den entsprechenden Gegenwert für den X-, Y- bzw. Z-Wert zurückgeben. Abbildung 4.4 zeigt Graphen der Funktionen. Die Funktion $I(\lambda)$ gibt die Intensität aus der SPD für die gegebene Wellenlänge zurück. Formal ergeben sich die XYZ-Werte aus dem Integral über alle Wellenlängen:

$$X = \int_{\lambda} \bar{x}(\lambda) I(\lambda) d\lambda \quad Y = \int_{\lambda} \bar{y}(\lambda) I(\lambda) d\lambda \quad Z = \int_{\lambda} \bar{z}(\lambda) I(\lambda) d\lambda \quad (4.1)$$

⁴https://en.wikipedia.org/wiki/File:CIE1931xy_CIERGB.svg

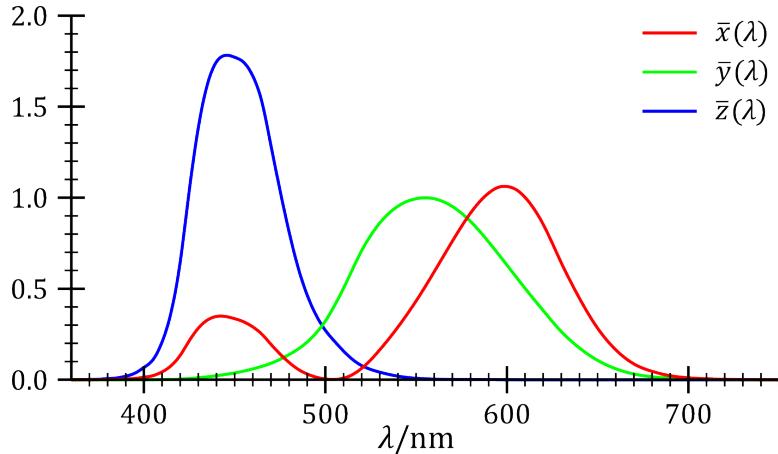


Abbildung 4.4: Die Beobachterfunktionen $\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$. Quelle:⁵

In der Regel berechnet man für die Beobachterfunktionen jedoch nicht das Integral, sondern nimmt die Werte aus einer LookUp-Tabelle, die vom CIE veröffentlicht wurde [Int07]. Vereinfacht wird aus den Gleichungen 4.1 somit eine Aufsummierung der Werte:

$$X = \sum_{\lambda} \bar{x}(\lambda) I(\lambda) \quad Y = \sum_{\lambda} \bar{y}(\lambda) I(\lambda) \quad Z = \sum_{\lambda} \bar{z}(\lambda) I(\lambda) \quad (4.2)$$

Für die weiteren Berechnungen müssen die Werte nun in das Intervall $[0,1]$ skaliert werden. Mit $n = X + Y + Z$ erhält man:

$$X_s = X/n \quad Y_s = Y/n \quad Z_s = Z/n = 1 - X_s - Y_s \quad (4.3)$$

Zu einem späteren Zeitpunkt wird auch noch die Intensität I der Farbe benötigt. Diese berechnet man bereits an dieser Stelle, wobei $m = \sum_{\lambda} \bar{x}(\lambda) + \sum_{\lambda} \bar{y}(\lambda) + \sum_{\lambda} \bar{z}(\lambda)$ (maximaler Wert für $X + Y + Z$) und n der aktuelle Wert für die SPD aus Gleichung 4.3 ist:

$$I = 1 - \frac{m - n}{m}, \quad I \in [0,1] \quad (4.4)$$

Von XYZ zu RGB. Im nächsten Schritt werden die skalierten XYZ-Werte nun in RGB-Werte umgewandelt und anschließend ihre Helligkeit angepasst. Die Umwandlung zu RGB geschieht über [Lin11]:

⁵https://en.wikipedia.org/wiki/File:CIE_1931_XYZ_Colour_Matching_Functions.svg

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \mathbf{M}^{-1} \begin{pmatrix} X_s \\ Y_s \\ Z_s \end{pmatrix} \quad (4.5)$$

Die Matrix \mathbf{M} ist zusammengesetzt aus den Farb-Koordinaten des verwendeten RGB-Systems und dessen Weißpunkt. Dabei gibt es verschiedene Farbsysteme je nach Monitor, zum Beispiel NTSC für analoge Fernseher. Ein solches Farbsystem beschreibt die darstellbaren Farben – was in Abbildung 4.3 dem Dreiecksbereich innerhalb des Farbsegels entspricht – und welcher Punkt darin der Farbe Weiß entspricht. Das hier verwendete ist ohne Berücksichtigung einer speziellen Monitorart und weist die folgenden Werte auf [Wal96]:

$$\begin{pmatrix} x_r \\ x_g \\ x_b \\ x_w \end{pmatrix} = \begin{pmatrix} 0,7355 \\ 0,2658 \\ 0,1669 \\ \frac{1}{3} \end{pmatrix} \quad \begin{pmatrix} y_r \\ y_g \\ y_b \\ y_w \end{pmatrix} = \begin{pmatrix} 0,2645 \\ 0,7243 \\ 0,0085 \\ \frac{1}{3} \end{pmatrix} \quad (4.6)$$

Die Werte x_w und y_w geben zusammen den Weißpunkt im Farbsystem an. Die Matrix \mathbf{M} selbst ist wie folgt aufgebaut:

$$\mathbf{M} = \begin{pmatrix} S_r X_r & S_g X_g & S_b X_b \\ S_r Y_r & S_g Y_g & S_b Y_b \\ S_r Z_r & S_g Z_g & S_b Z_b \end{pmatrix} \quad (4.7)$$

mit

$$\begin{aligned} X_r &= \frac{x_r}{y_r} & X_g &= \frac{x_g}{y_g} & X_b &= \frac{x_b}{y_b} \\ Y_r &= Y_g = Y_b = 1 \\ Z_r &= \frac{(1 - x_r - y_r)}{y_r} & Z_g &= \frac{(1 - x_g - y_g)}{y_g} & Z_b &= \frac{(1 - x_b - y_b)}{y_b} \end{aligned} \quad (4.8)$$

und

$$\begin{pmatrix} S_r \\ S_g \\ S_b \end{pmatrix} = \mathbf{N}^{-1} \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} \quad \mathbf{N} = \begin{pmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{pmatrix} \quad (4.9)$$

wobei $z_w = 1 - x_w - y_w$. Ausführlicher aufgeschrieben und unter Berücksichtigung von $Y_r = Y_g = Y_b = 1$ (siehe Gleichung 4.8) gilt somit:

$$\begin{aligned} S_r &= \frac{1}{|\mathbf{N}|} ((Z_b - Z_g)x_w + (X_b Z_g - X_g Z_b)y_w + (X_g - X_b)z_w) \\ S_g &= \frac{1}{|\mathbf{N}|} ((Z_r - Z_b)x_w + (X_r Z_b - X_b Z_r)y_w + (X_b - X_r)z_w) \\ S_b &= \frac{1}{|\mathbf{N}|} ((Z_g - Z_r)x_w + (X_g Z_r - X_r Z_g)y_w + (X_r - X_g)z_w) \end{aligned} \quad (4.10)$$

wobei \mathbf{N} die zu invertierende Matrix aus Gleichung 4.9 ist und es gilt Determinante $|\mathbf{N}| = X_r(Z_b - Z_g) - X_g(Z_b - Z_r) + X_b(Z_g - Z_r)$. Für die Ausgangsgleichung 4.5 ergibt sich somit:

$$\begin{aligned} R &= \frac{1}{|\mathbf{M}|} S_g S_b ((Z_b - Z_g)X_s + (X_b Z_g - X_g Z_b)Y_s + (X_g - X_b)Z_s) \\ G &= \frac{1}{|\mathbf{M}|} S_b S_r ((Z_r - Z_b)X_s + (X_r Z_b - X_b Z_r)Y_s + (X_b - X_r)Z_s) \\ B &= \frac{1}{|\mathbf{M}|} S_r S_g ((Z_g - Z_r)X_s + (X_g Z_r - X_r Z_g)Y_s + (X_r - X_g)Z_s) \end{aligned} \quad (4.11)$$

mit Determinante $|\mathbf{M}| = S_r S_g S_b (X_r(Z_b - Z_g) - X_g(Z_b - Z_r) + X_b(Z_g - Z_r))$.

Für die richtige Helligkeit müssen die RGB-Werte dann nur noch mit der in Gleichung 4.4 ermittelten Intensität I multipliziert werden:

$$\begin{pmatrix} R_s \\ G_s \\ B_s \end{pmatrix} = I \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (4.12)$$

Die so erhaltenen RGB-Komponenten liegen jeweils im Interval $[0, 1]$.

Beispiel. Die Berechnung des RGB-Wertes aus einer SPD soll nun am Beispiel vom Illuminant D65 vorgeführt werden. Die Werte für die SPD von Illuminant D65 finden sich in [Int07], wobei das Beispiel mit 40 Proben – also 40 Wellenlängen – rechnen wird. Der erste Wert ist von der Wellenlänge 380 nm, dann wird in Schritten von 10 nm weitergemacht bis einschließlich 770 nm. Die für das Beispiel relevanten 40 Werte von Wellenlängen zu Intensitäten von Illuminant D65 stehen in Tabelle 4.1.

Tabelle 4.1: SPD-Werte für Illuminant D65 bei 40 Proben.

λ	Intensität	λ	Intensität	λ	Intensität
380	0,444027773062	390	0,583145180457	400	0,739486639731
410	0,784800359895	420	0,764412793264	430	0,812935863919
440	0,941635826571	450	0,996587783927	460	0,987471564866
470	0,979458798764	480	0,953782297219	490	0,925898889757
500	0,921620887516	510	0,902251044036	520	0,901767222354
530	0,900137507215	540	0,884672189590	550	0,865981394085
560	0,833252130513	570	0,815376192578	580	0,782915152956
590	0,758376905578	600	0,762253420704	610	0,752460700098
620	0,725678199165	630	0,708704546226	640	0,694861304451
650	0,680072488371	660	0,689625844566	670	0,681433130751
680	0,628142294503	690	0,599813261807	700	0,619453026856
710	0,576991308186	720	0,558048416121	730	0,615270940142
740	0,588563134485	750	0,466891318372	760	0,480526601704
770	0,552525209656				

Tabelle 4.2: LookUp-Tabelle für die Beobachterfunktionen.

λ	$x(\lambda)$	$y(\lambda)$	$z(\lambda)$	λ	$x(\lambda)$	$y(\lambda)$	$z(\lambda)$
380	2,6899e-3	2,0000e-4	1,2260e-2	390	1,0781e-2	8,0000e-4	4,9250e-2
400	3,7981e-2	2,8000e-3	1,7409e-1	410	9,9941e-2	7,4000e-3	4,6053e-1
420	2,2948e-1	1,7500e-2	1,0658e+0	430	3,1095e-1	2,7300e-2	1,4672e+0
440	3,3336e-1	3,7900e-2	1,6166e+0	450	2,8882e-1	4,6800e-2	1,4717e+0
460	2,3276e-1	6,0000e-2	1,2917e+0	470	1,7476e-1	9,0980e-2	1,1138e+0
480	9,1944e-2	1,3902e-1	7,5596e-1	490	3,1731e-2	2,0802e-1	4,4669e-1
500	4,8491e-3	3,2300e-1	2,6437e-1	510	9,2899e-3	5,0300e-1	1,5445e-1
520	6,3791e-2	7,1000e-1	7,6585e-2	530	1,6692e-1	8,6200e-1	4,1366e-2
540	2,9269e-1	9,5400e-1	2,0042e-2	550	4,3635e-1	9,9495e-1	8,7823e-3
560	5,9748e-1	9,9500e-1	4,0493e-3	570	7,6425e-1	9,5200e-1	2,2771e-3
580	9,1635e-1	8,7000e-1	1,8066e-3	590	1,0230e+0	7,5700e-1	1,2348e-3
600	1,0550e+0	6,3100e-1	9,0564e-4	610	9,9239e-1	5,0300e-1	4,2885e-4
620	8,4346e-1	3,8100e-1	2,5598e-4	630	6,3289e-1	2,6500e-1	9,7694e-5
640	4,4062e-1	1,7500e-1	5,1165e-5	650	2,7862e-1	1,0700e-1	2,4238e-5
660	1,6161e-1	6,1000e-2	1,1906e-5	670	8,5753e-2	3,2000e-2	5,6006e-6
680	4,5834e-2	1,7000e-2	2,7912e-6	690	2,2187e-2	8,2100e-3	1,3135e-6
700	1,1098e-2	4,1020e-3	6,4767e-7	710	5,6531e-3	2,0910e-3	3,3304e-7
720	2,8253e-3	1,0470e-3	1,7026e-7	730	1,3994e-3	5,2000e-4	8,7107e-8
740	6,6847e-4	2,4920e-4	4,3162e-8	750	3,2073e-4	1,2000e-4	2,1554e-8
760	1,5973e-4	6,0000e-5	1,1204e-8	770	7,9513e-5	3,0000e-5	5,8340e-9

Multipliziert man die Intensitäten der Wellenlängen für D65 mit den jeweiligen Werten aus Tabelle 4.2 für die Beobachterfunktionen, ergibt sich für X, Y und Z:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 8,4661 \\ 8,93622 \\ 9,5453 \end{pmatrix} \quad \begin{pmatrix} X_s \\ Y_s \\ Z_s \end{pmatrix} = \frac{1}{n} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0,314169 \\ 0,331614 \\ 0,354217 \end{pmatrix} \quad (4.13)$$

mit $n = X + Y + Z = 26,9476$. Der Maximalwert liegt bei $m = 31,9512$. Damit liegt die Intensität bei $I = 0,843401$ (siehe Gleichung 4.4). Für die Umwandlung von XYZ zu RGB werden nun die Farb-Koordinaten aus Gleichung 4.6 verwendet. Dadurch ergibt sich:

$$\begin{pmatrix} X_r \\ X_g \\ X_b \end{pmatrix} = \begin{pmatrix} 2,780718 \\ 0,366975 \\ 19,635294 \end{pmatrix} \quad \begin{pmatrix} Y_r \\ Y_g \\ Y_b \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} Z_r \\ Z_g \\ Z_b \end{pmatrix} = \begin{pmatrix} 0 \\ 0,013944 \\ 97,011764 \end{pmatrix} \quad (4.14)$$

Als nächstes werden S_r, S_g, S_b aus Gleichung 4.9 berechnet. Die benötigten Werte liegen bereits alle vor und müssen nur noch eingesetzt werden.

$$S_r = \frac{1}{|N|} \cdot 14,134134 \quad S_g = \frac{1}{|N|} \cdot 63,201723 \quad S_b = \frac{1}{|N|} \cdot 0,796304 \quad (4.15)$$

mit $|N| = 234,396486$. Die Transformationsmatrix M sieht folgendermaßen aus:

$$M = \begin{pmatrix} 0,167677 & 0,098949 & 0,066706 \\ 0,0603 & 0,269635 & 0,003397 \\ 0 & 0,003759 & 0,329573 \end{pmatrix} \quad (4.16)$$

Damit lassen sich nun die RGB-Werte berechnen:

$$R = \frac{1}{|M|} \cdot 0,010931 \quad G = \frac{1}{|M|} \cdot 0,013305 \quad B = \frac{1}{|M|} \cdot 0,013763 \quad (4.17)$$

mit $|M| = 0,012947$. Somit lauten die Endwerte für R, G, B und für R_s, G_s, B_s mit angepasster Helligkeit:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 0,844309 \\ 1,027647 \\ 1,06305 \end{pmatrix} \quad \begin{pmatrix} R_s \\ G_s \\ B_s \end{pmatrix} = \begin{pmatrix} 0,712091 \\ 0,866718 \\ 0,896577 \end{pmatrix} \quad (4.18)$$

Dies entspricht einem sehr hellen Blauton, was ein korrektes Ergebnis zu sein scheint, da Illuminant D65 auch die Farbe des Mittagshimmels repräsentieren soll.

5 Path Tracing

Path Tracing ist ein globales Beleuchtungsmodell, mit dem sich realistische Bilder erzeugen lassen. Der wichtigste Effekt dabei ist indirekte Beleuchtung, d.h. die Farbe einer Fläche wird nicht nur durch direkte Lichteinstrahlung bestimmt, sondern auch durch Licht, dass von anderen Flächen reflektiert wird. Dadurch kommt es auch zu Effekten wie Color Bleeding – dem Abfärben einer Fläche auf eine andere durch das reflektierte Licht. Abbildung 5.1 zeigt diesen Effekt. Die orange-rote Wand links und die blaue Wand rechts im Bild färben durch Lichtreflektion auf die benachbarten Wände und das Objekt in der Mitte des Raumes ab. Alle Flächen in der Szene sind perfekt diffus, es handelt sich also nicht um Spiegelungen.

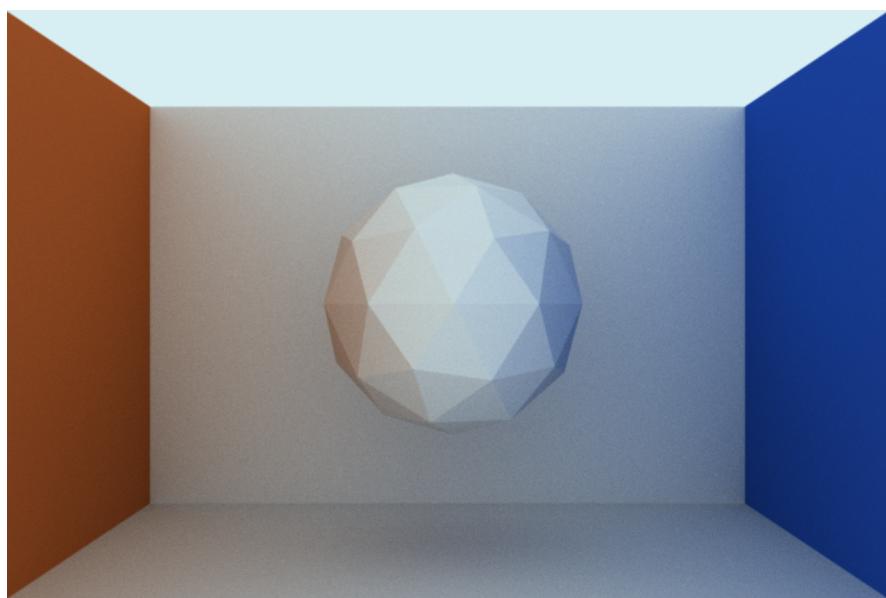


Abbildung 5.1: Color Bleeding.

Aus der Computergrafik kennt man üblicherweise vier verschiedene Lichtquellentypen: Richtungslichter, die aus einer bestimmten Richtung auf die Szene strahlen (z.B. eine Sonne); Punktlichter, die von einem einzigen Punkt aus gleichmäßig Licht ausstrahlen; Spotlights, die in einem bestimmten Winkel Licht ausstrahlen (z.B. Scheinwerfer); und Flächenlichter, bei denen eine ganze Fläche Licht abstrahlt. Flächenlichter sind dabei die realistischste Art von Lichtquelle, da alle Lichtquellen in der Natur ebenfalls eine Fläche haben – auch wenn diese u.U. sehr klein ist, wie z.B. die Oberfläche eines Glühdrahtes in einer Glühbirne oder eine LED. Entsprechend wirken Szenen mit Flächenlichtern auch realistischer und werden bevorzugt in der globalen Beleuchtung eingesetzt. Abbildung 5.2 führt noch einmal die vier Lichtquellentypen und ihre Strahlungsrichtungen auf.

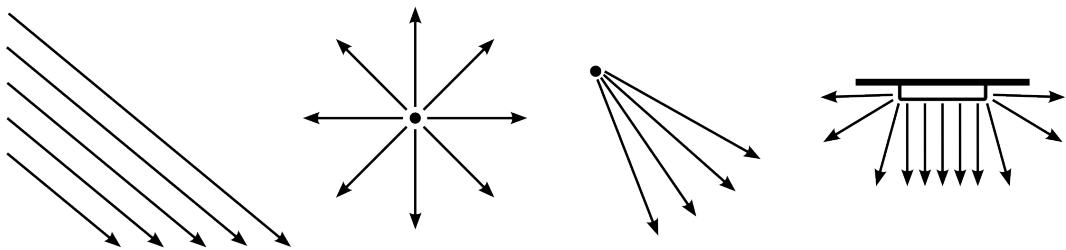


Abbildung 5.2: Richtungslicht, Punktlicht, Spotlight und Flächenlicht (v.l.n.r.).

Path Tracing beginnt mit Strahlen, die vom Kameraauge aus in die Szene geschossen werden. Für jede getroffene Oberfläche wird gemäß den Material-Eigenschaften eine neue Richtung für den Strahl bestimmt. Eine solche Aneinanderreihung von Strahlen wird als ein *Pfad* bezeichnet.

In [Hec90] führt Heckbert eine eigene Notation für Pfade ein, die Ähnlichkeit mit regulären Ausdrücken hat. Dabei steht ein E für das Auge, D für diffuse Flächen, S für spekulare Flächen und L für eine Lichtquelle – siehe auch Abbildung 5.4. Ein Beispiel für eine solche Pfadnotation ist $E[D \mid S]^*L$, was einen Pfad vom Auge zu einer Lichtquelle mit beliebig vielen diffusen und spekularen Flächen dazwischen beschreibt. Eine Notation für matte Flächen – Übergänge zwischen diffus und spekular – existiert allerdings nicht.

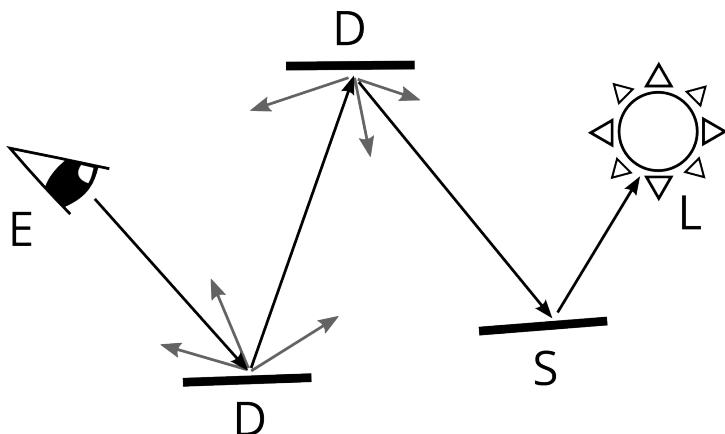


Abbildung 5.3: Ein Pfad vom Auge (E) aus zu einer Lichtquelle (L). Dabei trifft er diffuse Flächen (D) und eine spekulare (S).

Gültige Pfade – jene, die Energie von einer Lichtquelle transportieren und dadurch Farbinformationen beitragen – können auf zwei Arten zustandekommen: Als explizite und implizite Pfade.

Explizite Pfade sind Pfade, die von selbst eine Lichtquelle gefunden haben. Für jede getroffene Fläche wird ein neuer Strahl erzeugt und wenn dieser eine Lichtquelle trifft, ist der Pfad komplett. Kommen nur wenige gültige Pfade zustande, ist nur

leider auch das Bild sehr dunkel und wird aufgrund mangelnder Helligkeits- und Farbinformationen als „verrauschter“ empfunden.

Das andere sind *implizite* Pfade, bei denen bewusst die Wahrscheinlichkeit erhöht wird, dass Strahlen eine Lichtquelle treffen. Dabei werden an jeder getroffenen Oberfläche zwei Strahlen ausgesendet: Ein Strahl entsprechend dem Material, wie auch bei einem expliziten Pfad; und ein Strahl gezielt zu einer Lichtquelle. Trifft dieser zweite Strahl die Lichtquelle, fließt dieser Unterpfad ebenfalls mit ein. Dadurch wird das Bild insgesamt heller und weniger „verrauscht“, dafür ist auch der Rechenaufwand für die zusätzlichen Strahlen höher. Hinzu kommt auch noch ein höher Speicheraufwand, da Informationen über Lichtquellen vorhanden sein müssen. Aus den Lichtquellen muss zufällig eine gewählt und ebenfalls zufällig ein Punkt auf ihrer Fläche ausgewählt werden. Je nach Szene kann es aber auch sinnvoll sein, manche Quellen zu bevorzugen – z.B. den Himmel gegenüber einer kleinen Glühbirne. Eine sinnvolle Herangehensweise kann es daher sein, nur eine sehr geringe Anzahl besonders prominenter Lichtquellen anzusteuern, wie z.B. nur den Himmel und die Sonne.

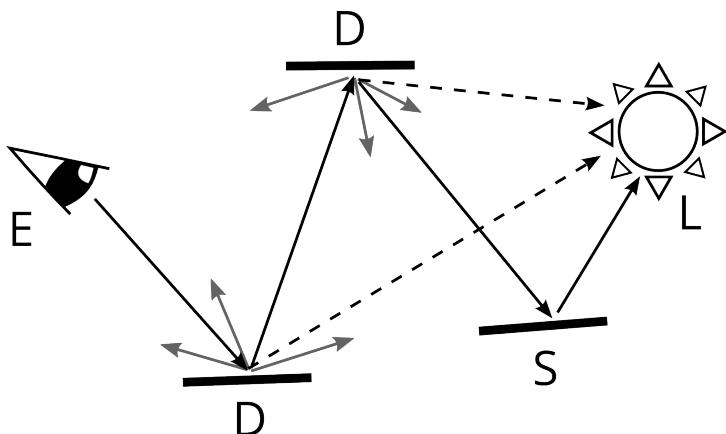


Abbildung 5.4: Von den diffusen Flächen gehen implizite Pfade zur Lichtquelle.

Ein Pfad terminiert entweder, wenn eine Lichtquelle getroffen wird, oder er eine vorgegebene Maximallänge erreicht. Eine zusätzliche Option ist eine Russisch-Roulette-Terminierung, bei der Pfade ab Erreichen einer vorgegebenen Mindestlänge zufällig abgebrochen werden.

In bestimmten Situationen sollten Pfade auch über ihre Maximallänge hinweg verlängert werden. Solche Fälle sind z.B. transparente oder spiegelnde Flächen. Angenommen ein Pfad trifft auf eine spiegelnde Fläche und würde nach der Reflexion beendet werden. Dann würden in der Spiegelung nur Lichtquellen zu sehen sein, da für andere Flächen ein weiterer Strahl notwendig wäre. Hier ist es sinnvoll, die maximale Pfadlänge zu erweitern.

5.1 Initiale Strahlen vom Auge aus

Ziel ist es, für jedes Pixel einen Farbwert zu errechnen. Dafür wird von jedem Pixel aus mindestens ein Pfad in die Szene begonnen, um getroffene Flächen zu finden und deren Farben zu kombinieren. Dieser Abschnitt erklärt, wie man diese initialen Vektoren je Pixelposition bestimmt [Bor03].

Die Kamera wird durch drei Vektoren beschrieben (siehe Abbildung 5.5), deren Werte bekannt sind:

- 3D-Vektor *eye*, der die aktuelle Position des Kameraauges beschreibt.
- 3D-Vektor *center*, der die Blickrichtung von *eye* angibt.
- 3D-Vektor *up*, der die Orientierung der Kamera nach oben angibt.

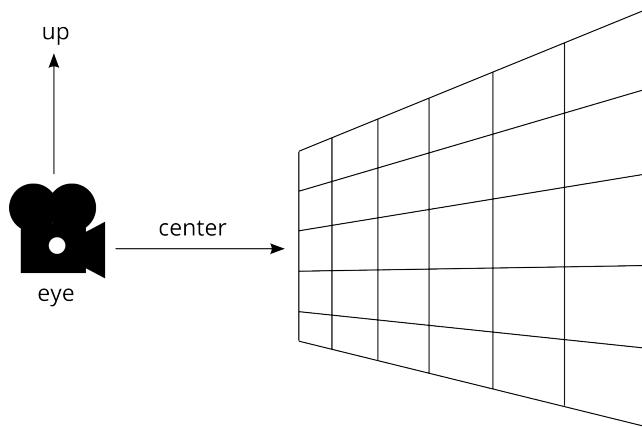


Abbildung 5.5: Veranschaulichung der Kamera-Vektoren.

Anhand dieser Vektoren wird als nächstes ein lokales Koordinatensystem für die Blickrichtung der Kamera bestimmt. Dieses wird beschrieben durch die Blickrichtung \hat{w} auf die Szene, sowie die beiden zu \hat{w} und gegenseitig orthogonalen Vektoren \hat{u} und \hat{v} . Dann gilt:

$$\begin{aligned}\hat{w} &= \frac{\text{center} - \text{eye}}{\|\text{center} - \text{eye}\|} \\ \hat{u} &= \frac{\hat{w} \times \text{up}}{\|\hat{w} \times \text{up}\|} \\ \hat{v} &= \frac{\hat{u} \times \hat{w}}{\|\hat{u} \times \hat{w}\|}\end{aligned}\tag{5.1}$$

Da \hat{w} direkt auf die Bildfläche zeigt und die drei Vektoren eine orthogonale Basis bilden, lässt sich das Problem ins 2-dimensionale projizieren, wie in Abbildung 5.6

dargestellt. Die Dimension von Vektor \hat{w} entfällt dabei, während Vektor \hat{u} parallel zur x-Achse der Bildfläche und \hat{v} parallel zur y-Achse verläuft.

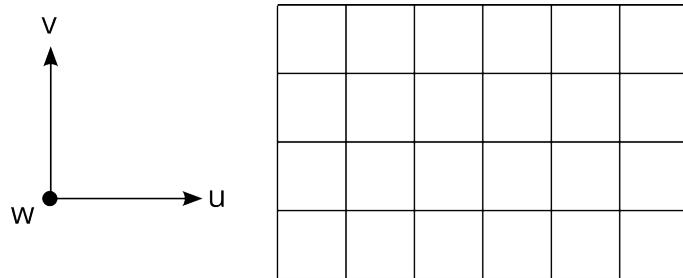


Abbildung 5.6: Das lokale Koordinatensystem ins 2-dimensionale reduziert.

Das zu erzeugende Bild habe nun die Dimensionen img_w für Breite und img_h für Höhe gemessen in Pixeln. Die Fläche eines Pixels wiederum habe die Breite und Höhe px . Der Vektor $r_{0,0}$ für den linken, oberen Punkt auf dem Raster der Bildfläche ist eingezeichnet in Abbildung 5.7 und berechnet sich wie folgt:

$$r_{0,0} = eye + \hat{w} - \frac{img_w}{2} \cdot px \cdot \hat{u} + \frac{img_h}{2} \cdot px \cdot \hat{v} \quad (5.2)$$

Die Größe eines Pixels ist zu diesem Zeitpunkt jedoch noch nicht bekannt. Sei fov das Sichtfeld der Kamera angegeben in Radianen, dann gilt:

$$px = aspect \cdot f \cdot \frac{1}{img_w} \quad (5.3)$$

mit

$$aspect = \frac{img_w}{img_h} \quad f = 2 \cdot \tan \frac{fov}{2} \quad (5.4)$$

Nun soll der Strahl aber durch die Mitte des Pixels verlaufen und nicht – wie jetzt noch – durch den Rasterschnittpunkt. Dafür muss die bisherige Position noch um eine halbe Pixelbreite und -höhe verschoben werden, was in Abbildung 5.7 zu sehen ist und zu der folgenden Gleichung führt:

$$r_{0,0} = eye + \hat{w} - \frac{img_w}{2} \cdot px \cdot \hat{u} + \frac{img_h}{2} \cdot px \cdot \hat{v} + \frac{px}{2} \cdot \hat{u} - \frac{px}{2} \cdot \hat{v} \quad (5.5)$$

Für die restlichen Pixel auf dem Bildraster muss gemäß der Position um weitere Pixelbreiten und -höhen fortlaufend verschoben werden. Sei pos_x der Index eines Pixels entlang der x-Achse und pos_y der Index eines Pixels entlang der y-Achse. Damit gelangt man zu einer verallgemeinerten Gleichung für einen Vektor r :

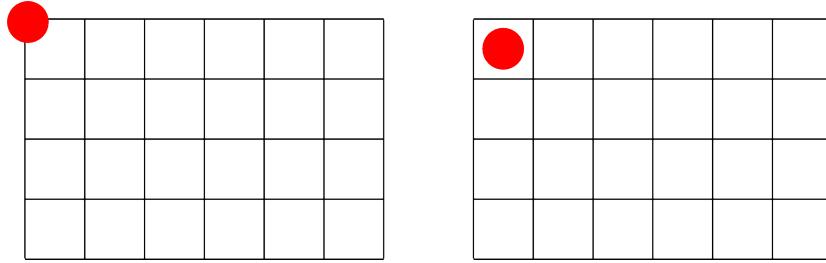


Abbildung 5.7: Links: $r_{0,0}$ ohne Anpassung, direkt auf dem Gitter liegend.
Rechts: $r_{0,0}$ im Pixel zentriert.

$$\begin{aligned} \mathbf{r} = \mathbf{eye} + \hat{\mathbf{w}} - \frac{img_w}{2} \cdot px \cdot \hat{\mathbf{u}} + \frac{img_h}{2} \cdot px \cdot \hat{\mathbf{v}} + \frac{px}{2} \cdot \hat{\mathbf{u}} - \frac{px}{2} \cdot \hat{\mathbf{v}} \\ + pos_x \cdot px \cdot \hat{\mathbf{u}} - pos_y \cdot px \cdot \hat{\mathbf{v}} \end{aligned} \quad (5.6)$$

Anschließend muss der Vektor \mathbf{r} noch normalisiert werden.

5.2 Lichtreflektion und -brechung

Im Path Tracing müssen Strahlen entsprechend den getroffenen Flächen in eine neue Richtung geschickt werden. Grundlegend unterscheidet man zunächst zwischen diffusen (z.B. Kreide) und spekularen Materialien (z.B. Spiegel). Darüber hinaus gibt es matte Abstufungen zwischen diesen beiden Typen, lichtdurchlässige Materialien und Oberflächen, deren Erscheinen von ihrem Betrachtungswinkel abhängen. Abbildung 5.8 zeigt ein Beispiel für diffuse, matte und spekular Flächen.

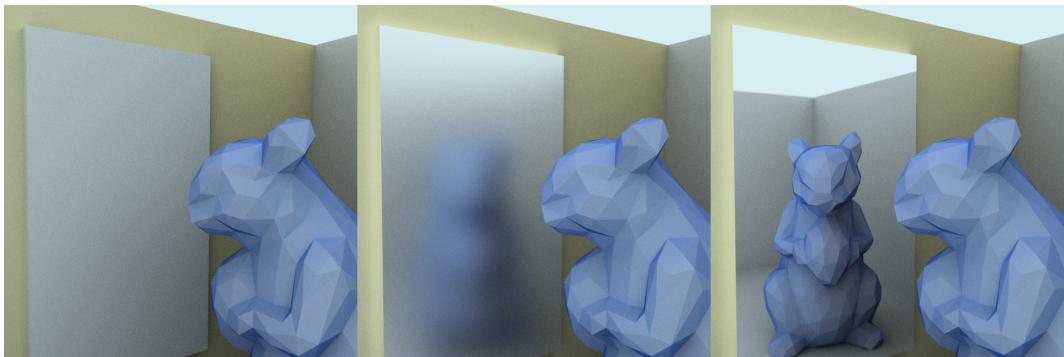


Abbildung 5.8: Die selbe Fläche einmal in diffus, matt und spekular (v.l.n.r.).

Diffuse Oberflächen. Auf perfekt diffusen Oberflächen werden die Strahlen in zufällige Richtungen reflektiert. Dafür wird anhand der Flächennormale $\hat{\mathbf{n}}$ eine

Hemisphäre über dem getroffenen Punkt p gebildet (siehe Abbildung 5.9) und ein neuer, zufälliger Strahl erzeugt, der die Hemisphäre schneidet [SC97].

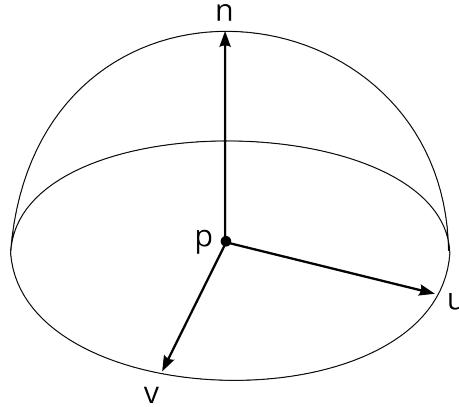


Abbildung 5.9: Hemisphäre über dem Punkt p .

Zunächst erstellt man ein lokales Koordinatensystem am Punkt p . Sei \hat{n} die Normale von p , dann werden zwei weitere Vektoren \hat{u}, \hat{v} benötigt, die zusammen mit \hat{n} eine orthonormale Basis bilden:

$$\hat{n}' = \begin{pmatrix} n_y \\ n_z \\ n_x \end{pmatrix} \quad \hat{u} = \frac{\hat{n}' \times \hat{n}}{\|\hat{n}' \times \hat{n}\|} \quad \hat{v} = \frac{\hat{n} \times \hat{u}}{\|\hat{n} \times \hat{u}\|} \quad (5.7)$$

Hierfür wird zunächst ein neuer Vektor \hat{n}' erzeugt, der eine Permutation der Koordinaten von \hat{n} ist, wobei jede Koordinate mit einer anderen vertauscht wurde. Dadurch ist sichergestellt, dass der Vektor \hat{n}' weder parallel noch entgegengesetzt zu \hat{n} verläuft. So lässt sich dann über das Kreuzprodukt der erste orthogonale Vektor \hat{u} aus dem Kreuzprodukt von \hat{n} und \hat{n}' bilden, was nicht möglich wäre, verliefen die Vektoren parallel bzw. entgegengesetzt. Der zweite Vektor \hat{v} muss nun sowohl zu \hat{n} als auch \hat{u} orthogonal sein, weshalb aus diesen beiden Vektoren das Kreuzprodukt gebildet wird. Durch Normalisieren der Vektoren liegt schließlich eine orthonormale Basis vor. Einen zufälligen Vektor innerhalb der durch die orthonormale Basis beschriebene Hemisphäre erhält man nun über:

$$\mathbf{d} = (\hat{u} \cos \varphi + \hat{v} \sin \varphi) \cdot c + \hat{n} \cdot d \quad (5.8)$$

Seien $a, b \in [0, 1]$ Zufallszahlen, dann gilt für die noch fehlenden Faktoren:

$$c = \sqrt{a} \quad d = \sqrt{1 - a} \quad \varphi = 2 \pi b \quad (5.9)$$

Spiegelnde Oberflächen. Für perfekt spiegelnde Oberflächen gilt der Satz, dass der Einfallswinkel des Strahls gleich dem ausgehenden Winkel ist (siehe Abbildung 5.10). Seien \hat{i} der einfallende Strahl und \hat{n} die Normale der Fläche, dann gilt für die Reflektion \hat{r} [DG06]:

$$\hat{r} = \hat{i} - 2(\hat{n} \cdot \hat{i})\hat{n} \quad (5.10)$$

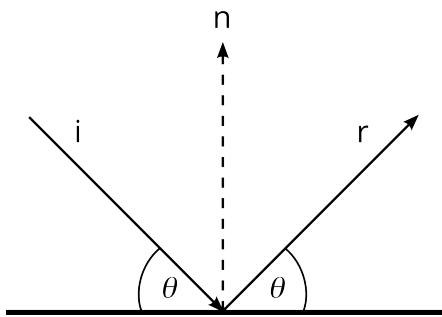


Abbildung 5.10: Vektor \hat{i} wird an der Flächennormale \hat{n} gespiegelt und wird so zu Vektor \hat{r} .

Matte Oberflächen. Matte Oberflächen können als ein Zustand zwischen perfekt diffus und perfekt spekular betrachtet werden. Definiert man wie in [Sch94] vorgeschlagen für Oberflächen einen Grobheitsfaktor $r \in [0,1]$, lässt sich die Lichtreflektion \hat{m} bestimmen als eine gewichtete Kombination aus dem diffusen Vektor \hat{d} und dem spekularen Vektor \hat{r} :

$$\hat{m} = r \cdot \hat{d} + (1 - r) \cdot \hat{r} \quad (5.11)$$

Hierbei handelt es sich aber nur um einen Vorschlag, wie sich matte Oberflächen ohne BRDF realisieren lassen. BRDFs führen i.d.R. eigene Funktionen ein, um neue Strahlen gemäß den Materialeigenschaften zu erzeugen. Dies wird in Kapitel 6 vor gestellt.

Lichtbrechung. Trifft ein Lichtstrahl auf eine durchsichtige Oberfläche, wie z.B. Glas, tritt er in das Objekt ein. Verschiedene Materialien haben verschiedene Bre chungsindizes, welche angeben, wie stark ein Lichtstrahl von seiner bisherigen Richtung abgelenkt wird. Für ein Vakuum liegt dieser Brechungsindex bei 1 und für Glas bei ungefähr 1,5 [DG06]. Im Internet finden sich zudem Datenbanken⁶ für Brechungsindizes verschiedener Materialien.

⁶z.B. <http://refractiveindex.info/>

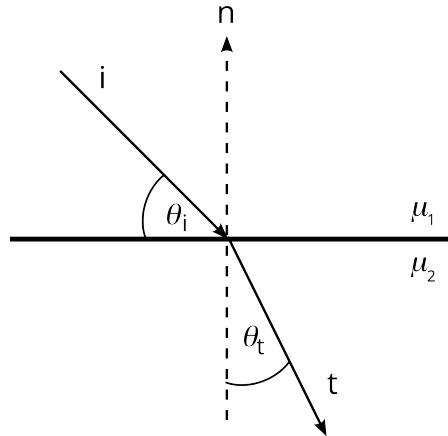


Abbildung 5.11: Strahl \hat{i} wird an der Oberfläche gebrochen.

Sei μ_1 das Material durch das sich der Strahl aktuell bewegt und μ_2 das Material, in das der Strahl eindringt. Dann gilt für den an der Oberfläche gebrochenen Strahl \hat{t} :

$$\hat{t} = \frac{\mu_1}{\mu_2} \cdot \hat{i} + \left(\frac{\mu_1}{\mu_2} \cdot \cos \theta_i - \sqrt{1 - \sin^2 \theta_t} \right) \quad (5.12)$$

mit

$$\cos \theta_i = -\hat{i} \cdot \hat{n} \quad \sin^2 \theta_t = \left(\frac{\mu_1}{\mu_2} \right)^2 (1 - \cos^2 \theta_i) \quad (5.13)$$

wobei \hat{i} wieder die Richtung des einfallenden Strahles ist und \hat{n} die Normale der Oberfläche. In Abbildung 5.11 sind die beteiligten Vektoren und Winkel zu sehen.

Wie in Gleichung 5.12 zu sehen ist, gilt die Bedingung $\sin^2 \theta_t \leq 1$. Wird diese nicht erfüllt, wird der Strahl stattdessen reflektiert wie in Gleichung 5.11 beschrieben. Dies wird als *totale interne Reflektion* (TIR) bezeichnet.

Treffen Lichtstrahlen auf eine durchlässige Oberfläche wird ein Anteil T durchgelassen und im Material gebrochen, während der andere Anteil R reflektiert wird. Dabei gilt $T + R = 1$. Der Anteil an reflektiertem Licht berechnet sich abhängig von Einfalls- und Brechungswinkel, sowie den Brechungsindizes. Dies wird in den Fresnelschen Gleichungen beschrieben:

$$R_{\perp} = \left(\frac{\mu_1 \cos \theta_i - \mu_2 \cos \theta_t}{\mu_1 \cos \theta_i + \mu_2 \cos \theta_t} \right)^2 \quad (5.14)$$

$$R_{\parallel} = \left(\frac{\mu_2 \cos \theta_i + \mu_1 \cos \theta_t}{\mu_2 \cos \theta_i - \mu_1 \cos \theta_t} \right)^2$$

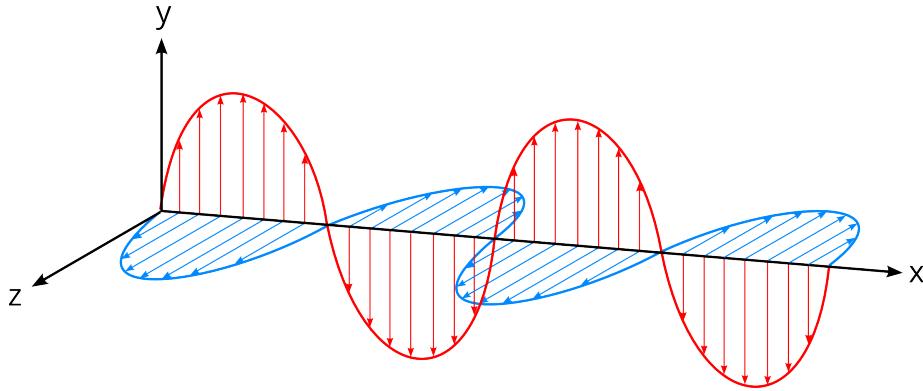


Abbildung 5.12: Polarisiertes Licht mit zwei Wellen in unterschiedlichen Lagen.

Fresnel berücksichtigt durch die Aufteilung in R_{\perp} und R_{\parallel} die beiden möglichen Polarisierungen von Licht. Diese Aufteilung wird in Abbildung 5.12 gezeigt, in der sich ein Lichtstrahl entlang der y-Achse ausbreitet. Die rote Kurve entspricht hierbei R_{\perp} und die blaue Kurve R_{\parallel} . Dieser Effekt kann im Rendering berücksichtigt werden, bedeutet aber zusätzlichen Rechenaufwand. In der Regel verzichten Ray Tracer darauf und nutzen die durch Schlicks Annäherungsverfahren [Sch94] vereinfachte Formel:

$$R(\theta_i) = \begin{cases} R_0 + (1 - R_0)(1 - \cos \theta_i)^5 & \Leftrightarrow \mu_1 \leq \mu_2 \\ R_0 + (1 - R_0)(1 - \cos \theta_t)^5 & \Leftrightarrow \mu_1 > \mu_2 \wedge \neg \text{TIR} \\ 1 & \Leftrightarrow \mu_1 > \mu_2 \wedge \text{TIR} \end{cases} \quad (5.15)$$

mit

$$R_0 = \left(\frac{\mu_1 - \mu_2}{\mu_1 + \mu_2} \right)^2 \quad (5.16)$$

Für den durchgelassenen Anteil gilt dann entsprechend $T(\theta_i) = 1 - R(\theta_i)$.

Anisotropische Oberflächen. Materialien können feine, gerichtete Rillen in ihrer Oberfläche haben. Dies hat zur Folge, dass Licht abhängig von der Position des Betrachters auf eine bestimmte Art und Weise reflektiert wird. Man spricht hierbei von einem anisotropischen Material. Das Gegenteil ist ein isotropisches Material. Beispiele für anisotropische Oberflächen sind beispielsweise gebürstetes Metall, Samt oder ein fließendes Gewässer [PF90].

6 Bidirectional Rendering Distribution Functions

Beim Einsatz im Path Tracing findet die BRDF an zwei Stellen Verwendung: Um eine neue Richtung für einen Strahl zu bestimmen und zur Gewichtung, wie viel Energie dieser zuliefert. In Abbildung 6.1 sind die Vektoren und Winkel eingezeichnet, auf die sich die folgenden Abschnitte beziehen werden. Dabei ist \hat{n} die Normale der Fläche, \hat{k}_1 der Vektor des einfallenden Lichtes, \hat{k}_2 der Vektor des ausgehenden Lichts, \hat{h} der Winkelhalbierende Vektor zwischen \hat{k}_1 und \hat{k}_2 , \bar{h} die Projektion von \hat{h} auf \hat{n} mit $\bar{h} \perp \hat{n}$, und \hat{u}, \hat{v} die zusammen mit \hat{n} eine orthonormale Basis bilden. Der Winkel φ gibt den Azimut-Winkel (Horizontalwinkel) und θ den Zenit-Winkel (Höhenwinkel) an. Wie in der Abbildung zu sehen, stehen beide Winkel in Zusammenhang mit dem Vektor \hat{h} , dessen Richtung sich auch nur über diese beiden Winkel angeben ließe.

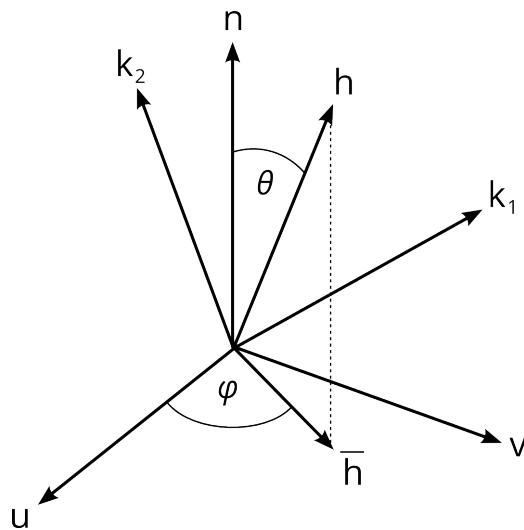


Abbildung 6.1: Vektoren und Winkel für die BRDF. Nach [Sch94, AS00].

Darüber hinaus seien die folgenden Variablen definiert:

$$\begin{aligned}
 t &= \hat{h} \cdot \hat{n} & \theta &= \arccos t \\
 u &= \hat{h} \cdot \hat{k}_2 & \\
 v_1 &= \hat{k}_1 \cdot \hat{n} & (6.1) \\
 v_2 &= \hat{k}_2 \cdot \hat{n} \\
 w &= \hat{u} \cdot \bar{h} & \varphi &= \arccos w
 \end{aligned}$$

Im folgenden sollen zwei BRDFs vorgestellt werden: Jene von Schlick [Sch94] und die von Shirley und Ashikhmin [AS00]. Dabei besteht jede Vorstellung aus drei Ab-

schnitten: Der BRDF selbst, der Wahrscheinlichkeitsdichtefunktion für Importance Sampling, sowie dem Generieren neuer Strahlen.

6.1 Schlick

In [Sch94] stellt Schlick eine BRDF vor, die speziell für schnelles, Physik-basiertes Rendering geeignet sein soll. Sie unterstützt einen flüssigen Übergang zwischen perfekt spekular und perfekt diffus, sowie Anisotropie. Über eine Technik namens Rational Fraction Approximation wurden aufwendige Formeln zudem in eine vereinfachte Form gebracht. Dabei werden für eine Formel Kern-Bedingungen aufgestellt – z.B. für $f(x) = \sin x$ wäre eine Kern-Bedingung $f(0) = 0$ –, anhand derer dann eine vereinfachte Form aufgestellt wird. Ein weiterer Vorteil ist, dass die Parameter zur Beschreibung eines Materials auf eine geringe Anzahl beschränkt sind, die zudem intuitiv verständlich sind:

- $C_\lambda \in [0, 1]$ – Reflektion des Lichtanteils mit der Wellenlänge λ .
- $r \in [0, 1]$ – Rauheit der Fläche, wobei $r = 0$ perfekt spekular, und $r = 1$ perfekt diffus ist.
- $p \in [0, 1]$ – Isotropie der Fläche, wobei $p = 0$ perfekt anisotrop (Abhängigkeit vom Betrachtungswinkel), und $p = 1$ perfekt isotrop (keine Abhängigkeit vom Betrachtungswinkel) ist.

Gewichtung des Lichteinfalls. Sei R_λ die Gewichtung des einfallenden Lichtes je nach Wellenlänge λ . Dann setzt sich diese Gewichtung zusammen aus dem spektralen Faktor S_λ und dem Richungsfaktor D .

$$R_\lambda(t, u, v_1, v_2, w) = S_\lambda(u) \cdot D(t, v_1, v_2, w) \quad (6.2)$$

Für den spektralen Faktor verwendet Schlick eine Approximation von Fresnels Gesetz, die es zu einem Polygon 5. Grades reduziert:

$$S_\lambda(u) = C_\lambda + (1 - C_\lambda)(1 - u)^5 \quad (6.3)$$

Der Richungsfaktor ist eine Summe aus drei verschiedenen Reflektionsmodellen (Lambert, anisotropische Microfacetten, Fresnel), die über $a, b, c \in [0, 1]$ unterschiedlich gewichtet werden. Der Lambert-Anteil $\frac{a}{\pi}$ beschreibt hierbei perfekt diffuse Flächen; das Microfacetten-Modell beschreibt matte Flächen, welche durch Unebenheiten in einer spiegelnden Oberfläche produziert werden [WMLT07] (siehe Abbildung 6.2); und der Fresnel-Anteil wirkt sich auf spiegelnde Flächen aus.

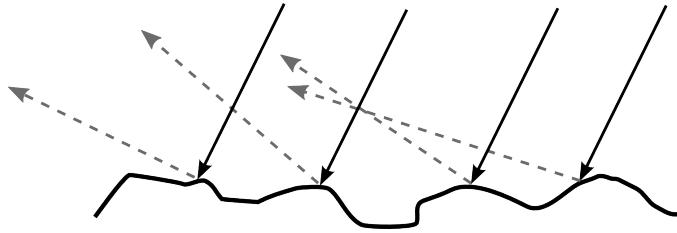


Abbildung 6.2: Oberfläche mit Micro-Facetten. Aufgrund der Unebenheiten werden einfallende Strahlen in unterschiedliche Richtungen reflektiert.

$$D(t, v_1, v_2, w) = \frac{a}{\pi} + b \cdot B(t, v_1, v_2, w) + \frac{c}{v_1} \quad \text{mit } a + b + c = 1 \quad (6.4)$$

Die Gewichtungen a, b, c können entweder vom Benutzer für jedes Material speziell definiert werden, oder abhängig von der Rauheit r automatisch bestimmt werden:

$$b = 4r(1-r) \quad a = \begin{cases} 0 & \text{für } r < \frac{1}{2} \\ 1-b & \text{sonst} \end{cases} \quad c = \begin{cases} 1-b & \text{für } r < \frac{1}{2} \\ 0 & \text{sonst} \end{cases} \quad (6.5)$$

Für sehr raue und somit diffuse Flächen ($r > \frac{1}{2}$) wird dann entsprechend der spekulare Fresnel-Anteil durch eine Gewichtung von $c = 0$ vernachlässigt, während umgekehrt für sehr glatte und somit spiegelnde Flächen ($r < \frac{1}{2}$) der Lambert-Anteil durch eine Gewichtung von $a = 0$ entfällt. Die Gewichtungen werden annähernd so gewählt, dass ein fließender Übergang von perfekt diffusen zu matten zu perfekt spekularen Flächen entsteht.

Für die Funktion $B(t, v, v', w)$ stellt Schlick zwei verschiedene Formeln zur Auswahl:

$$\begin{aligned} B_1(t, v_2, v_1, w) &= \frac{1}{4\pi v_1 v_2} Z(t) A(w) \\ B_2(t, v_2, v_1, w) &= \frac{G(v_1)G(v_2)}{4\pi v_1 v_2} Z(t) A(w) + \frac{1 - G(v_1)G(v_2)}{4\pi v_1 v_2} \end{aligned} \quad (6.6)$$

Bei $G(v_1)$ und $G(v_2)$ handelt es sich um einen geometrischen Faktor, der Selbst-Verdeckungen des Materials berücksichtigt:

$$G(v_1) = \frac{v_1}{r - r v_1 + v_1} \quad G(v_2) = \frac{v_2}{r - r v_2 + v_2} \quad (6.7)$$

Wie in Abbildung 6.3 zu sehen, treffen einfallende Lichtstrahlen auf die une ebene Fläche. Der mittlere Strahl wird jedoch so reflektiert, dass er durch einen Vorsprung verdeckt wird. Dies wird durch $G(v_1)G(v_2)$ ausgedrückt. Licht, das in einen verdeckten Bereich eintritt, wird jedoch auch wieder ausgestrahlt, was mit $1 - G(v_1)G(v_2)$ berücksichtigt wird.

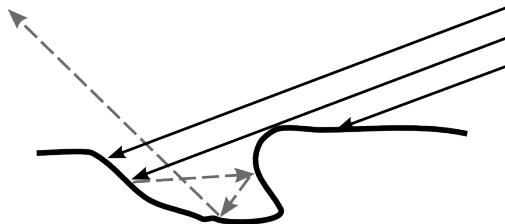


Abbildung 6.3: Selbst-Verdeckung einer rauen Oberfläche. Gestrichelte Linien sind reflektierte Strahlen.

Entscheidet man sich z.B. für die Funktion $B_1(t, v_2, v_1, w)$ aus Gleichung 6.6, sieht das Schlick-BRDF wie folgt aus:

$$D(t, v_1, v_2, w) = \frac{a}{\pi} + \frac{b}{4\pi v_1 v_2} Z(t) A(w) + \frac{c}{v_1} \quad (6.8)$$

Wahrscheinlichkeitsdichtefunktion. Als Dichtefunktion wird folgende Formel verwendet:

$$p(\hat{k}_1, \hat{k}_2) = \frac{1}{4\pi u} \quad (6.9)$$

Bestimmung der neuen Strahlrichtung. Die neue Richtung für den ausgehenden Strahl von Punkt p aus beruht auf dem Zenit-Winkel θ und dem Azimut-Winkel φ . Seien $a, b \in [0, 1]$ Zufallszahlen, dann gilt:

$$\theta = \arccos \sqrt{\frac{a}{r - a}} \quad \varphi = \frac{\pi}{2} \sqrt{\frac{p^2 b^2}{1 - b^2 + b^2 p^2}} \quad (6.10)$$

Mit der Gleichung für φ wird bisher jedoch nur ein Quadrant abgedeckt, ein Strahl muss aber in alle Richtungen der Hemisphäre ausgestrahlt werden können.

$$b' = \begin{cases} 1 - 4(0,25 - b) & \text{für } b < 0,25 \\ 1 - 4(0,5 - b) & \text{für } b < 0,5 \\ 1 - 4(0,75 - b) & \text{für } b < 0,75 \\ 1 - 4(1 - b) & \text{für } b < 1 \end{cases} \quad (6.11)$$

Daraufhin berechnet man den Winkel φ mit b' und erhält den Winkel φ' für den kompletten Bereich über:

$$\varphi' = \begin{cases} \varphi & \text{für } b < 0,25 \\ \pi - \varphi & \text{für } b < 0,5 \\ \pi + \varphi & \text{für } b < 0,75 \\ 2\pi - \varphi & \text{für } b < 1 \end{cases} \quad (6.12)$$

Als nächstes erstellt man ein lokales Koordinatensystem wie schon für diffuse Oberflächen in Gleichung 5.7. Daraufhin erhält man den Winkelhalbierenden Vektor \hat{h} von \hat{k}_1 und \hat{k}_2 :

$$\hat{h} = (\hat{u} \cos \varphi' + \hat{v} \sin \varphi') \cdot \sin \theta + \hat{n} \cdot \cos \theta \quad (6.13)$$

Der Vektor \hat{h} stammt aus einem kegelförmigen Bereich um die Normale \hat{n} herum. Dieser Bereich kommt durch die Kombination zweier Vektoren zustande. Zunächst wird mit $(\hat{u} \cos \varphi' + \hat{v} \sin \varphi')$ ein zur Fläche tangenter Vektor erzeugt, dessen Richtung innerhalb der durch \hat{u} und \hat{v} aufgespannten Ebene über den Azimut-Winkel φ bestimmt wird. Dann wird dieser Vektor über den Zenit-Winkel θ – bzw. dessen Sinus und Kosinus – mit der Normalen kombiniert. Durch die Zufallszahlen a, b kann dieser Bereich je nach Material variieren, was durch den kegelförmigen Bereich um die Normale herum in Abbildung 6.4 veranschaulicht werden soll.

Für eine perfekt diffuse Oberfläche wäre dieser Bereich identisch zur gesamten Hemisphäre, da bei Rauheit $r = 1$ der Zenit-Winkel $\theta = \arccos \sqrt{a}$ ist und somit $\theta \in [0,1]$, was den gesamten Zenit abdeckt. Bei einer perfekt spekularen Fläche entspricht \hat{h} genau der Normalen, denn für $r = 0$ wird $\theta = 0$, somit entfällt durch $\sin 0 = 0$ der Einfluss des Azimut-Winkels und übrig bleibt $\hat{n} \cdot \cos 0 = \hat{n}$. Mit zunehmender Rauheit r der Fläche wächst also auch der Bereich. Anisotropische Eigenschaften durch Werte $p < 1$ würden zudem eine Neigung dieses Bereiches bewirken.

Im letzten Schritt muss der Blickvektor \hat{k}_2 nur noch an \hat{h} gespiegelt werden:

$$\hat{k}_1 = \hat{k}_2 - 2(\hat{h} \cdot \hat{k}_2)\hat{h} \quad (6.14)$$

Während \hat{h} in der Hemisphäre liegt, kann es jedoch passieren, dass der gespiegelte Strahl \hat{k}_1 unterhalb die Hemisphäre zeigt. Je diffuser die Oberfläche, desto größer die Gefahr. Das Problem ist, dass diese Strahlen nichts zum Pfad beitragen, weshalb man sie vermeiden möchte. Dieser Fall tritt hauptsächlich bei diffusen Flächen auf,

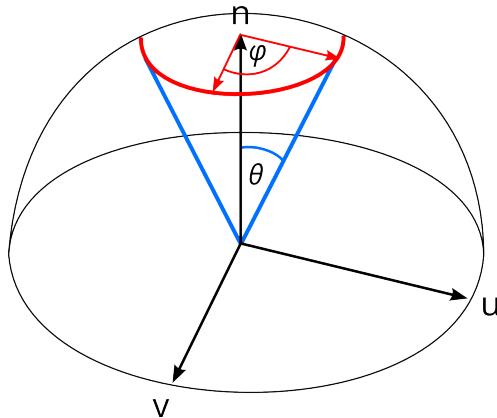


Abbildung 6.4: Der Bereich der Hemisphäre, in dem der neue Winkel-halbierende Vektor liegen kann, festgelegt durch den Azimut-Winkel φ (rot) und den Zenit-Winkel θ (blau).

weshalb sich folgendes anbietet: Gilt $(\hat{k}_1 \cdot \hat{n}) < 0$, dann erzeugt man einen neuen Strahl für \hat{k}_1 mit dem in Kapitel 5.2 vorgestellten Verfahren für diffuse Flächen.

6.2 Shirley-Ashikhmin

Die BRDF von Shirley und Ashikhmin unterstützt spekulare, matte, diffuse und anisotropische Flächen, wobei es das Energieerhaltungsgesetz einhält [AS00]. Die Oberflächeneigenschaften eines Materials werden über vier Parameter kontrolliert:

- $n_u, n_v \in [0, \infty)$ – Zwei Werte ähnlich den Exponenten im Phong-Modell, die die Form der Spiegelung kontrollieren.
- $R_s \in [0, 1]$ – Faktor für die spekulare Reflektierung.
- $R_d \in [0, 1]$ – Faktor für die diffuse Reflektierung.

Im Gegensatz zur Schlick-BRDF existieren keine getrennten Parameter für Rauheit und Isotropie, sondern diese Effekte werden durch das Parameter-Paar n_u, n_v kontrolliert. Je kleiner die Werte für n_u, n_v , desto diffuser bzw. je größer die Werte desto spekularer erscheint eine Fläche. Eine Fläche erscheint nahezu perfekt spekular ab Werten jenseits den 10.000. Sind die Werte für n_u, n_v verschieden, treten anisotropische Effekte auf. Welcher der beiden Werte der größere ist, hat zudem Auswirkungen auf die Form der Reflektion.

Anders als in der Schlick-BRDF ist zudem der Lambert-Faktor $(\hat{n} \cdot \hat{v}; n)$ aus Gleichung 3.1 kein Bestandteil der BRDF. Stattdessen wird der diffuse Anteil einer Fläche über eine eigene Komponente ρ_d berücksichtigt, um so besser den Energieerhalt sicher zu stellen.

Gewichtung des Lichteinfalls. Die BRDF von Shirley und Ashikhmin besteht aus der Summe eines spekularen Anteils ρ_s und eines diffusen Anteils ρ_d :

$$\rho(\hat{k}_1, \hat{k}_2) = \rho_s(\hat{k}_1, \hat{k}_2) + \rho_d(\hat{k}_1, \hat{k}_2) \quad (6.15)$$

Die Formel für den spekularen Anteil ist dabei gegeben als:

$$\rho_s(\hat{k}_1, \hat{k}_2) = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{8\pi} \cdot \frac{t^a}{u \cdot \max(v_1, v_2)} \cdot f \quad (6.16)$$

Der Exponent a kann auf zwei Weisen ausgedrückt werden:

$$\begin{aligned} a &= n_u \cos^2 \varphi + n_v \sin^2 \varphi \\ a &= \frac{n_u (\hat{h} \cdot \hat{u})^2 + n_v (\hat{h} \cdot \hat{v})^2}{1 - t^2} \end{aligned} \quad (6.17)$$

Die zweite Variante für a verwendet keine trigonometrischen Funktionen und ist dadurch schneller zu berechnen, weshalb diese auch für die Implementierung verwendet wird. Der Faktor f ist der durch Schlick angenäherte Fresnel-Term:

$$f = R_s + (1 - R_s)(1 - u)^5 \quad (6.18)$$

Der diffuse Anteil ergibt sich aus der Gleichung:

$$\rho_d(\hat{k}_1, \hat{k}_2) = \frac{28 R_d}{23 \pi} (1 - R_s) \left(1 - \left(1 - \frac{v_1}{2}\right)^5\right) \left(1 - \left(1 - \frac{v_2}{2}\right)^5\right) \quad (6.19)$$

Im Gegensatz zum spekularen Anteil haben die Parameter n_u, n_v keinen Einfluss. Dafür wird der diffuse Anteil umso schwächer gewichtet, je stärker die spekulare Reflektanz gegeben durch den Parameter R_s ist. Die Konstante $\frac{28}{23}$ wird zur Erfüllung des Energieerhaltungssatzes benötigt.

Wahrscheinlichkeitsdichtefunktion. Die Dichtefunktion für den Lichtstrahl beruht auf einer zusätzlichen Dichtefunktion $p_h(\hat{h})$ eigens für den Halb-Vektor \hat{h} :

$$p_h(\hat{h}) = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{2\pi} \cdot t^{(n_u \cos^2 \phi + n_v \sin^2 \phi)} \quad (6.20)$$

Bei $p_h(\hat{h})$ handelt es sich zudem um eine echte Dichtefunktion – das Integral der Dichtefunktion über alle möglichen Richtungen \hat{h} der Hemisphäre beträgt 1. Die

Funktion $p_h(\hat{h})$ findet sich dann wie folgt in der Dichtefunktion für den Lichtstrahl wieder:

$$p(\hat{k}_1, \hat{k}_2) = \frac{p_h(\hat{h})}{4u} \quad (6.21)$$

Bestimmung der neuen Strahlrichtung. Seien $a, b \in [0, 1]$ Zufallszahlen. Dann gilt für den Azimut-Winkel φ und Zenit-Winkel θ :

$$\varphi = \arctan \left(\sqrt{\frac{n_u + 1}{n_v + 1}} \tan \frac{\pi a}{2} \right) \quad (6.22)$$

$$\theta = \arccos \left((1 - b)^{1/(n_u \cos^2 \varphi + n_v \sin^2 \varphi + 1)} \right) \quad (6.23)$$

Wie bei der Schlickschen BRDF aus Kapitel 6.1 deckt φ zunächst nur einen Quadranten ab. Daher müssen auch hier die gleichen Anpassungen wie bei Schlick, beginnend bei Gleichung 6.11, vorgenommen werden. Den Halb-Vektor \hat{h} und daraus folgend die neue Richtung \hat{k}_1 erhält man ebenfalls wie bei Schlick.

7 Interaktives Rendering

Wie in Kapitel 6 beschrieben, kann das Integral der Render-Gleichung über das Monte Carlo-Verfahren angenähert werden. Bei Offline-Rendering würde man eine Anzahl von n Stichproben vorgegeben, wobei eine Stichprobe einem erzeugten Pfad für jedes Pixel entspricht. Die errechneten Pixelfarben der Pfade würden je Pixel addiert und mit der Anzahl n gemittelt.

Das Rendering soll jedoch interaktiv geschehen, weshalb es nicht möglich ist, für jeden Frame erst auf die Auswertung aller n Stichproben zu warten. Stattdessen ist gewünscht, dass unmittelbar nach Start der Anwendung oder Änderung der Kameraposition ein Bild zu sehen ist. Die Anpassung für interaktives Rendering besteht darin, dass bereits das Ergebnis der ersten Stichprobe als Frame angezeigt wird. Alle darauffolgend generierten Frames werden mit ihren Vorgängern kombiniert, sodass das Gesamtbild sich mit fortschreitender Zeit dem Ergebnis eines vergleichbaren Offline-Renderers angleicht.

Sei g das aus n Stichproben gerenderte Bild und f_i eine erzeugte Stichprobe. Dann ist die Gleichung für die Monte Carlo-Integration:

$$g = \frac{1}{n} \sum_{i=1}^n f_i \quad (7.1)$$

Das Problem lautet nun, dass hieraus eine Gleichung werden muss, bei der die Anzahl n der Stichproben einen nicht konstanten Wert haben darf. Dies wird möglich, indem man die Gleichung als Rekursion aufstellt [HGNR⁺12]. Sei i ein Zähler für die Anzahl an bereits generierten Frames und g_i das aktuell anzuzeigende Bild. Dann ist die Gewichtung eines neu-erzeugten Frames f_i mit seinem Vorgänger f_{i-1} :

$$\begin{aligned} g_i &= f_{i-1} \cdot \frac{i}{i+1} + f_i \cdot \left(1 - \frac{i}{i+1}\right) \\ &= f_i + (f_{i-1} - f_i) \cdot \frac{i}{i+1} \end{aligned} \quad (7.2)$$

Für $i = 0$ sei der Vorgänger als $f_{-1} = 0$ (schwarzer Frame) definiert. Für den ersten Frame f_0 besteht das angezeigte Bild somit nur aus diesem ersten Frame, da $\frac{0}{1} = 0$. Bei Berechnung des zweiten Frames f_1 liegt das Gewicht bei $\frac{1}{2}$ – die beiden bisher erzeugten Frames werden im Verhältnis 1:1 gemischt. Der dritte Frame f_2 fließt nur noch zu $\frac{1}{3}$ ein. Mit steigender Anzahl der generierten Frames verlagert sich die Gewichtung immer stärker zu Gunsten des Vorgänger-Frames:

$$\lim_{i \rightarrow \infty} \frac{i}{i+1} = 1 \quad (7.3)$$

Während also theoretisch unendlich viele Stichproben in das Bild einfließen können, wird sich ab einer bestimmten Anzahl praktisch kein optischer Unterschied mehr feststellen lassen. Ein optischer Vergleich eines im Laufe der Zeit entstehenden Bildes ist in den Abbildungen 7.1 und 7.2 zu sehen.

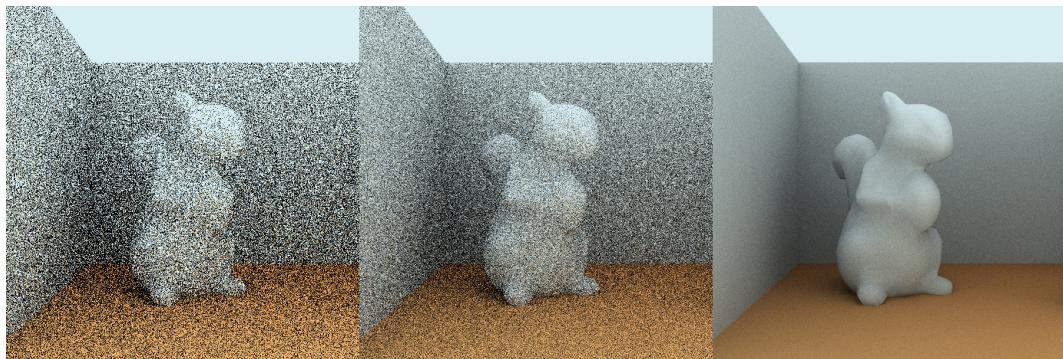


Abbildung 7.1: Nach dem ersten Durchgang, drei Durchgängen und nach ca. 30 Sekunden bei 10 FPS (v.l.n.r.).

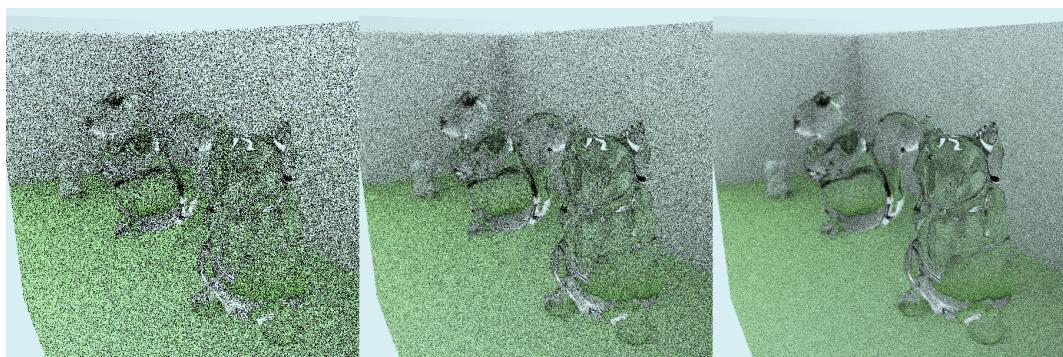


Abbildung 7.2: Nach dem ersten Durchgang, drei Durchgängen und zehn Durchgängen (v.l.n.r.).

8 Implementierung

Das komplette Path Tracing und Auswerten der Farben findet auf der Grafikkarte statt. Zu diesem Zweck wird *OpenCL*⁷ als Schnittstelle verwendet. Die Hauptanwendung selbst läuft auf der CPU, dank dem OpenCL-Framework können aber Berechnungen parallel auf der Grafikkarte ausgeführt werden. Einstiegspunkte für die Berechnungen sind sogenannte Kernel-Funktionen, bei denen es sich um Funktionen handelt, die man über das Stichwort `kernel` speziell als Kernel-Funktionen definiert hat. Kernel können auf der GPU parallel ausgeführt werden, was ideal für Path Tracing ist, da die einzelnen Pfade unabhängig voneinander entstehen.

Alle Threads, die parallel auf der GPU ausgeführt werden sollen, werden von OpenCL einer Workgroup zugeordnet. Die maximale Größe lässt sich mit der Funktion `clGetDeviceInfo()` und dem Flag `CL_DEVICE_MAX_WORK_ITEM_SIZES` abfragen – ein Resultat wäre z.B. (1024, 1024, 64). In diesem Fall würde bei einem größeren Bild als 1024×1024 Pixeln mehr als eine Workgroup angelegt. Die gewünschte Workgroup-Größe lässt sich aber auch vom Benutzer angeben und sollte deutlich unter den Maximalwerten liegen. In diesem Projekt hat sich eine Workgroup-Größe von 8×8 als am performantesten erwiesen. Einschränkungen für die Größe sind, dass jede Dimension ein Vielfaches von 2 sein muss und dass die Dimensionen (z.B. Breite und Höhe des Frames) durch die Workgroup-Größe restlos teilbar sein müssen. Während also eine Workgroup-Größe von 8×8 für einen 800×600 -Frame in Ordnung ist, kann eine Größe von 16×16 nicht verwendet werden, da $\frac{600}{16} = 37,5$.

Alle Threads, die parallel auf der GPU ausgeführt werden, können auf einen gemeinsamen globalen Speicherbereich zugreifen und so Informationen austauschen. Zugriffe auf diesen Speicher erfolgen jedoch nur sehr langsam. Threads, die zudem der gleichen Workgroup angehören, teilen sich auch einen lokalen Speicherbereich, auf den sich schneller zugreifen lässt als auf den globalen.

Abbildung 8.1 zeigt eine grobe Übersicht über den Programmablauf: Nach Start der Anwendung wählt man eine OBJ-Datei zum Rendern aus. Diese und die MTL- und SPEC-Dateien werden dann eingelesen, wobei es sich bei der SPEC-Datei um ein eigenes Format in diesem Projekt handelt, in dem SPDs gespeichert sind. Aus den geometrischen Daten der OBJ-Datei wird eine Datenstruktur (hier eine Baumstruktur) für die späteren Schnittpunkttests auf der Grafikkarte generiert. Danach wird OpenCL vorbereitet – der OpenCL-Code wird eingelesen, kompiliert, Buffer erzeugt und gefüllt. Anschließend beginnt das Rendering indem der OpenCL-Kernel ausgeführt wird. Jeder erzeugte Frame wird in eine 2D-Textur geschrieben und an eine OpenGL-View übergeben, die die Textur darstellt.

⁷<http://www.khronos.org/opencl/>

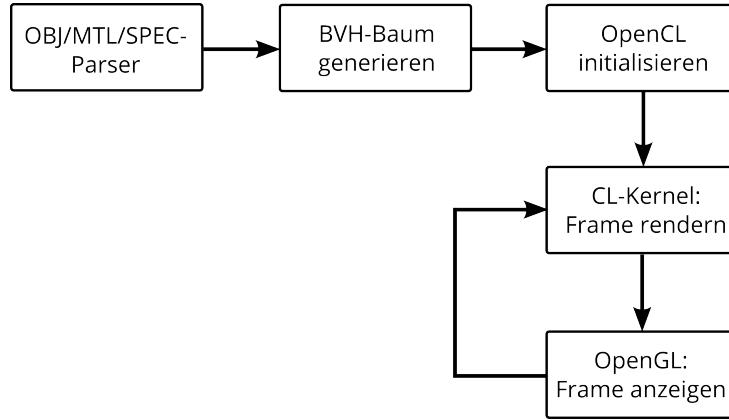


Abbildung 8.1: Übersicht über den Programmablauf.

Es wird OpenCL 1.1 mit der Implementierung des NVIDIA-Treibers 319.60 verwendet. Dabei kommt nur ein einziger Kernel zum Einsatz, dessen Ablauf in Abbildung 8.2 zu sehen ist. Zuerst wird der initiale Strahl in die Szene berechnet. Daraufhin wird getestet, welche Fläche der Strahl trifft und entsprechend dem Material der Fläche die SPD aktualisiert sowie ein neuer Strahl erzeugt. Wurde eine Lichtquelle getroffen, wird die endgültige SPD berechnet und als RGB-Wert für das Pixel gesetzt. Solange keine Lichtquelle getroffen wurde und die maximale Pfadlänge nicht überschritten wurde, wird das Path Tracing fortgesetzt. Die Anzeige des Bildes als 2D-Textur geschieht dann über einen OpenGL 3.3 Shader.

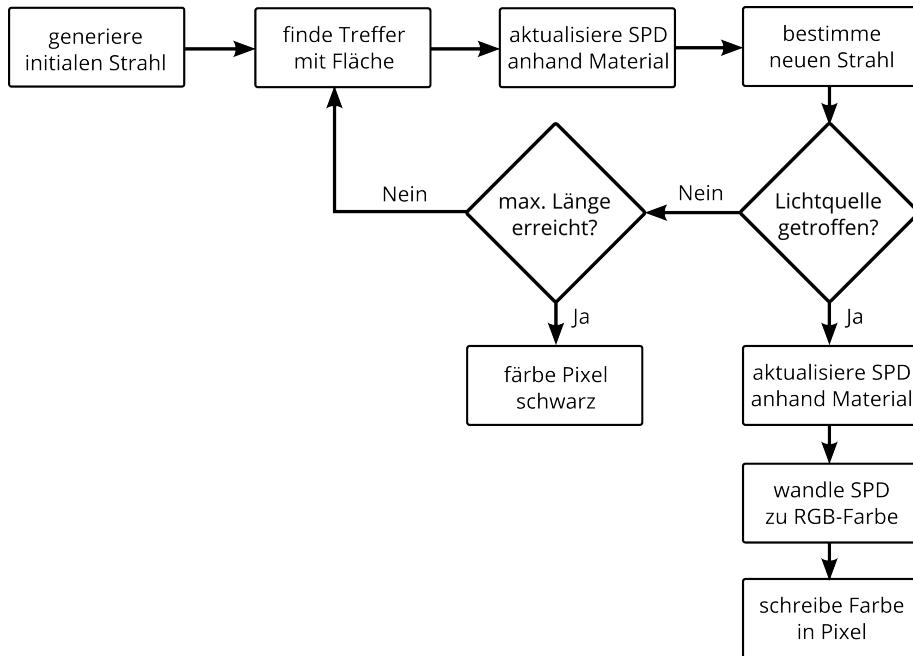


Abbildung 8.2: Übersicht über den OpenCL-Kernel-Ablauf.

8.1 Zufallszahlen

An mehreren Stellen im OpenCL-Kernel werden Zufallszahlen benötigt. Es müssen dabei keine echten Zufallszahlen sein, sondern Pseudo-Zufallszahlen sind bereits ausreichend. Pseudo-Zufallszahlen sind Zahlen, die für einen Betrachter zufällig erscheinen, sich aber berechnen lassen. Im Gegensatz dazu werden echte Zufallszahlen durch physikalische Einwirkungen erzeugt, wie z.B. das Rauschen elektrischer Bauteile.

OpenCL stellt für (Pseudo-)Zufallszahlen leider keine Funktionen zur Verfügung, weshalb diese Werte selbst erzeugt werden müssen. Hierzu wird der Code in Listing 8.1 verwendet. Die Grundlage für die Pseudo-Zufallszahl ist hierbei das Erzeugen von Aliasing [Bal12]. Aliasing wird in der Signalanalyse definiert als Fehler, die auftreten, wenn im Eingangssignal Werte erscheinen, die höher sind als die halbe Abtastfrequenz. Als solch eine Abtastfunktion wird hier die Sinus-Funktion verwendet.

Listing 8.1: Funktion für Zufallszahlen in OpenCL.

```

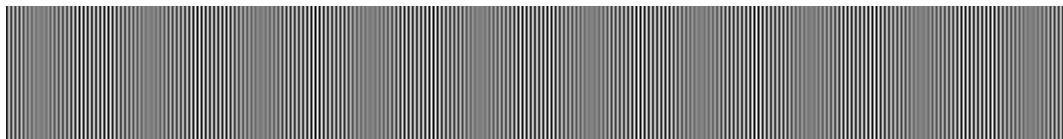
1 inline float rand( float* seed ) {
2     float i;
3     *seed += 1.0f;
4     return fract( native_sin( *seed ) * 43758.5453123f, &i );
5 }
```

Der Seed wird vor Beginn der Kernel-Ausführung auf die bisher verstrichene Zeit gesetzt, damit der Wert für jeden Frame verschieden ist. Bei jedem Aufruf von `rand(&seed)` wird der Seed dann um Eins erhöht, damit beim nächsten Aufruf eine neue Zufallszahl erzeugt wird. Da die Zeit in Millisekunden angegeben wird, nimmt der Seed sehr schnell, sehr hohe Werte an, was zu Aliasing führt. Dies allein reicht jedoch nicht, da die so erzeugten Zahlen immer noch ein sich wiederholendes Muster ergeben. Abbildung 8.3a veranschaulicht dies anhand einer erzeugten Grafik – die erzeugten Zufallszahlen wurden dabei als Grauton verwendet und die Grafik zur besseren Erkennung in der Höhe gestreckt. Dieser Wert wird nun an die Funktion `fract` übergeben, die nur die Nachkommastellen einer Zahl zurückgibt:

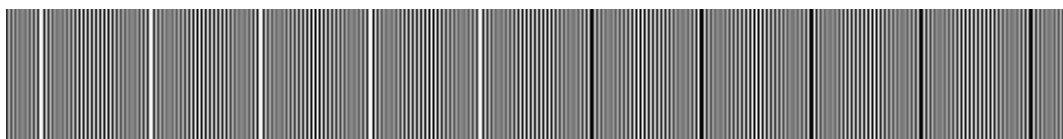
$$\text{fract}(x) = x - \lfloor x \rfloor \quad (8.1)$$

Die so erzeugten Werte liegen dadurch im Intervall $[0, 1]$. Wie in Abbildung 8.3b zu sehen, treten weiterhin Aliasing-Effekte und bereits schwaches Rauschen auf. Um das Rauschen zu verstärken, wird das Ergebnis der Sinus-Funktion zusätzlich mit einem großen Wert multipliziert. (In vielen Implementierungen vergleichbarer

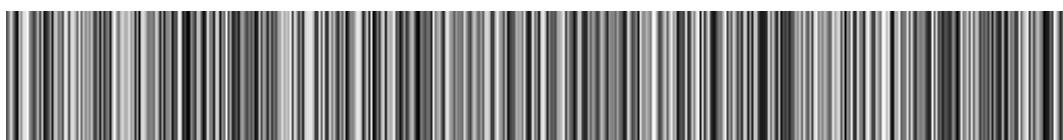
Zufallszahlen-Generatoren wird die Zahl 43.758,5453123 verwendet. Es ließ sich jedoch keine Quelle finden, die die Wahl dieses Wertes begründet.) Als Signal betrachtet bedeutet dies, dass die Amplitude des Signals erhöht wird. Die Funktion `fract` liefert nun hinreichend zufällige Pseudo-Zufallszahlen, wie Abbildung 8.3c veranschaulicht.



(a) Der Sinus sehr hoher Zahlenwerte. Aliasing-Effekte treten auf, ein Muster bleibt aber zu erkennen.



(b) Der Sinus-Wert wird zusätzlich an die Funktion `fract` übergeben. Neben Aliasing tritt nun auch leichtes Rauschen auf.



(c) Der an `fract` übergebene Sinus-Wert wird stark erhöht. Wie gewünscht tritt ein starkes Rauschen auf.

Abbildung 8.3: Grafische Veranschaulichung erzeugter Zahlen. Aus [Bal12].

Nun bleibt aber noch ein Problem: Der Anfangswert für den Seed ist für jedes Pixel in einem Frame zu Beginn identisch, was bedeutet, dass auch alle Folgewerte identisch sein werden. Um dies zu vermeiden, wird auf den Seed nach jeder getroffenen Fläche noch der t -Faktor aus der Strahlengleichung $r = o + t \cdot \hat{d}$ aufaddiert, da dieser von Strahl zu Strahl variiert und so die Zufallsreihen von einander abweichen lässt. Andernfalls passiert, was in Abbildung 8.4 zu sehen ist: Alle Strahlen, die Flächen mit der selben Normale treffen, werden in die selbe Richtung reflektiert. Da der anfängliche Strahl für jedes Pixel auch mit dem gleichen Seed beginnt, werden alle Folgestrahlen auch anhand des gleichen Seeds erzeugt. Die einzigen weiteren Faktoren für einen neuen Strahl sind die Eigenschaften des Materials und die Normale des getroffenen Punktes. Lässt man nicht noch einen weiteren Faktor einfließen, über den sich Strahlen voneinander unterscheiden lassen, zeigen alle Folgestrahlen solcher sich ähnelnder Flächen in die selbe Richtung. Deswegen wird noch der t -Faktor miteinbezogen.

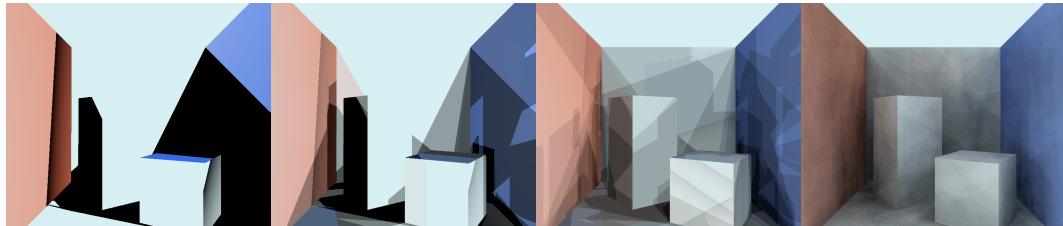


Abbildung 8.4: Aufgrund des selben Seeds werden Strahlen bei der selben Normale in die selbe Richtung reflektiert. Die selbe Szene nach 1 Frame, 3 Frames, 10 Frames und einigen Sekunden bei 10 FPS (*v.l.n.r.*).

8.2 Implizite Pfade

Während ein expliziter Pfad gebildet wird, wird an jeder unterwegs getroffenen, diffusen Fläche versucht, einen impliziten Pfad zu bilden. An diesen Punkten wird also gezielt ein Strahl in Richtung einer Lichtquelle geschickt. Wird keine Lichtquelle getroffen, passiert nichts weiter. Wird hingegen eine Lichtquelle getroffen, wird diese Helligkeit zum Gesamtergebnis hinzugerechnet.

Anhand von Listing 8.2 soll das näher erläutert werden. Das Array `spd` akkumuliert für jede Wellenlänge die Abschwächung der Energie, zu der es durch die getroffenen Materialien und die Auswertung durch die BRDF kommt. Entsteht schließlich ein expliziter Pfad, wird die SPD der Lichtquelle mit den Faktoren aus dem Array `spd` multipliziert und so die finale SPD für den expliziten Pfad gebildet. Hinzu kommen noch die impliziten Pfade. An jedem getroffenen Punkt – außer der Lichtquelle – wird ein Strahl in Richtung einer Lichtquelle geschickt. Wird eine Lichtquelle getroffen, wird deren SPD dem Gesamtergebnis hinzugefügt.

8.3 Berechnung der Lichtpfade

Dieses Kapitel beschäftigt sich mit dem Finden der Lichtpfade im Modell. Um das Finden der von Strahlen getroffenen Oberflächenpunkte zu beschleunigen, wird eine Bounding Volume Hierarchy als Datenstruktur verwendet.

8.3.1 Beschleunigungsstruktur

Als Beschleunigungsstruktur wird eine Bounding Volume Hierarchy (BVH) in Form eines Binärbaumes verwendet. Eine Szene besteht aus einem oder mehreren 3D-Objekten – z.B. Wände, Säulen, Eichhörnchen usw. Für jedes Objekt wird ein BVH-Baum erzeugt. Diese Bäume werden wiederum selbst in einer BVH untergebracht. Zum Schluss liegt nur noch ein einziger Baum vor, der alle Objekte beinhaltet.

Listing 8.2: Beitrag eines impliziten Pfades.

```

1 if( implicitLight ) { // Implicit path (light source hit)
2     brdf = BRDF( implicitRay, normal, mtl )
3         * lambert( normal, implicitRay.dir );
4
5     for( int i = 0; i < SPEC; i++ ) {
6         spdTotal[i] += spd[i] * specPowerDists[indexLight + i]
7             * specPowerDists[index + i] * brdf;
8     }
9 }
10
11 // Explicit path (no light source hit)
12 brdf = BRDF( ray, normal, mtl ) * lambert( normal, ray.dir );
13 for( int i = 0; i < SPEC; i++ ) {
14     spd[i] *= specPowerDists[index + i] * brdf;
15 }
```

Erzeugen einer BVH. Algorithmus 8.1 zeigt den Ablauf. Die Ausgangssituation ist eine Bounding Box für das gesamtes Objekt, die anhand aller Faces des Modells berechnet wurde. Diese Bounding Box ist der erste Knoten der BVH – der Root-Knoten. Im nächsten Schritt wird der Knoten anhand eines Kriteriums in zwei Kindknoten aufgeteilt.

Algorithmus 8.1 Erzeugen einer BVH.

```

1:  $F_{all}$                                          ▷ all faces of the object
2:  $maxFaces \leftarrow 4$                          ▷ maximal number of faces per leaf node
3: function BUILDTREE( $F$ )
4:    $N \leftarrow \text{CREATENODE}(F)$            ▷  $N_{bb}$  is the bounding box
5:   if  $|F| \leq maxFaces$  then
6:     return  $N$ 
7:   end if
8:    $split \leftarrow \text{SAH or midpoint of longest } N_{bb} \text{ axis}$ 
9:    $F_{left} \leftarrow \text{all faces left of } split$ 
10:   $F_{right} \leftarrow \text{all faces right of } split$ 
11:   $N_{left} \leftarrow \text{BUILDTREE}(F_{left})$           ▷ left child node
12:   $N_{right} \leftarrow \text{BUILDTREE}(F_{right})$         ▷ right child node
13:  return  $N$ 
14: end function
15:  $R \leftarrow \text{BUILDTREE}(F_{all})$                   ▷ root node
```

Wie ein Knoten geteilt wird, wird durch eine *Surface Area Heuristic* (SAH) bestimmt. Die SAH trifft eine grobe Aussage darüber, wie aufwendig das Durchwandern und Testen eines Knotens ist. Je nachdem, wie man einen Knoten unterteilt, sind die Faces unterschiedlich verteilt und haben die Kindknoten unterschiedlich große Bounding Boxen. Entsprechend variiert auch der Aufwand, einen Knoten zu testen, denn Bounding Boxen mit einer großen Oberfläche werden wahrscheinlicher von einem Strahl getroffen. Je mehr Faces eine Bounding Box beinhaltet, desto mehr Schnittpunkttests müssen durchgeführt werden. Man möchte einen Knoten also so unterteilen, dass die SAH einen möglichst kleinen Aufwand ergibt.

Der Algorithmus 8.2 beschreibt das Vorgehen. Die SAH wird entlang jeder Achse separat angewendet. Beginnt man mit der X-Achse, sortiert man die Faces des Knotens also zuerst nach ihrer X-Koordinate. Hierfür bietet es sich an, den Face-Schwerpunkt zu nehmen. Danach erzeugt man eine Reihe von Bounding Boxen, entsprechend der möglichen Aufteilungen in zwei Kindknoten. Man beginnt also von der linken Seite aus mit einer Bounding Box, die nur das erste Face beinhaltet, während man von rechts aus eine Bounding Box hat, die alle Faces außer dem ersten beinhaltet. Im nächsten Schritt wird das zweite Face aus der rechten Bounding Box entfernt und der linken Bounding Box hinzugefügt. Dabei wird die Größe der Bounding Boxen neu berechnet. Dies wiederholt man so lange, bis in der rechten Bounding Box nur noch ein Face übrig bleibt. Abbildung 8.5 zeigt noch einmal exemplarisch das Unterteilen des Knotens anhand der Faces. Wie zu sehen ist, kann es dabei auch zu Überlappungen der Bounding Boxen kommen.

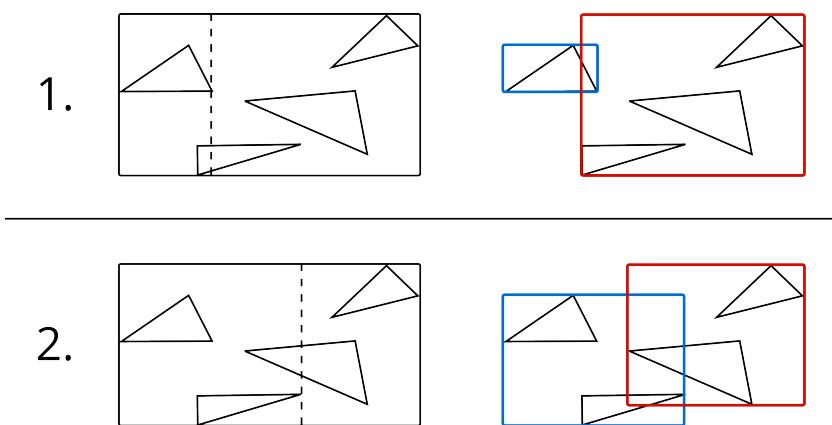


Abbildung 8.5: Schrittweises Aufteilen in Kindknoten für die SAH.

Anschließend nutzt man die immer paarweise erzeugten linken und rechten Bounding Boxen zur Berechnung der SAH. Wurde dies für alle Achsen durchgeführt, nimmt man den günstigsten SAH-Wert und teilt die Faces entsprechend in die entstehenden Kindknoten auf. Diese Art der Aufteilung wird auch als Objekt-Teilung bezeichnet, da Faces eindeutig einem Kindknoten zugeordnet werden. Im Gegen-

Algorithmus 8.2 Knoten-Teilung anhand einer Surface Area Heuristic (SAH).

```

1:  $F_{node} \leftarrow$  faces in this node
2:  $SA_{node} \leftarrow$  surface area of the current node
3:  $SAH_{min} \leftarrow \infty$ 

4: for each axis do
5:    $F_{node} \leftarrow$  sort faces by axis using the centroid
6:    $i \leftarrow 1$                                  $\triangleright$  grow split combinations from the left side
7:    $BB_{left,0} \leftarrow \text{BOUNDINGBOX}(F_{node}[0])$ 
8:   while  $i < |F_{node}| - 1$  do
9:      $BB_{left,i} \leftarrow \text{BOUNDINGBOX}(BB_{left,i-1}, F_{node}[i])$ 
10:     $i \leftarrow i + 1$ 
11:   end while

12:   $i \leftarrow |F_{node}| - 2$                    $\triangleright$  grow split combinations from the right side
13:   $BB_{right,i} \leftarrow \text{BOUNDINGBOX}(F_{node}[i + 1])$ 
14:  while  $i \geq 0$  do
15:     $BB_{right,i} \leftarrow \text{BOUNDINGBOX}(BB_{right,i+1}, F_{node}[i + 1])$ 
16:     $i \leftarrow i - 1$ 
17:  end while

18:   $i \leftarrow 0$                                  $\triangleright$  compare split possibilities
19:  while  $i < |F_{node}| - 1$  do
20:     $SA_{left} \leftarrow$  surface area of  $BB_{left,i}$ 
21:     $SA_{right} \leftarrow$  surface area of  $BB_{right,i}$ 
22:     $F_{left} \leftarrow$  faces in  $BB_{left,i}$ 
23:     $F_{right} \leftarrow$  faces in  $BB_{right,i}$ 
24:     $SAH \leftarrow 1/SA_{node} (SA_{left} \cdot |F_{left}| + SA_{right} \cdot |F_{right}|)$ 
25:    update  $SAH_{min}$ 
26:     $i \leftarrow i + 1$ 
27:  end while
28: end for

29: split node where the split has  $SAH_{min}$ 

```

satz dazu existiert noch die räumliche Teilung, bei der Faces auch zu beiden Kindknoten gehören können, aber unter Umständen an den Grenzen der Bounding Box abgeschnitten werden.

Da die SAH ein recht aufwendiges Verfahren ist, wird für Knoten mit vielen Faces – z.B. mehr als 1000 – optional auf ein schnelleres Verfahren ausgewichen, das dafür aber auch einen weniger optimalen Baum erzeugt. Hierbei wird der Knoten einfach am Mittelpunkt der längsten Achse geteilt. Ist also die Bounding Box entlang der X-Achse am längsten, liegt der Trennpunkt so, dass er die Bounding Box auf der X-Achse halbiert. Alle Faces, deren Schwerpunkt unterhalb dieses Punktes liegen,

werden dem linken Kindknoten zugeordnet; alle mit dem Schwerpunkt überhalb werden dem rechten Kindknoten zugeordnet. Für die so erzeugten Kindknoten wird anhand der Faces wieder die Bounding Box bestimmt. Abbildung 8.6 zeigt eine solche Teilung am Mittelpunkt.

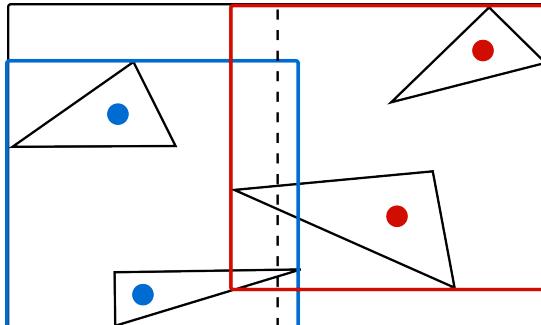


Abbildung 8.6: Kindknoten (links blau und rechts rot umrandet) mit neuen Bounding Boxen anhand ihrer Faces.

Die Knoten werden so lange weiter unterteilt, bis ein Abschlusskriterium erreicht wird. Üblicherweise ist dies eine maximale Baumtiefe oder eine Maximalanzahl von Faces in einem Knoten, die unterschritten wird. Hier wird eine Maximalanzahl von vier Faces je Knoten verwendet, bei der ein Blattknoten generiert wird. Generell sollten sich nicht zu viele Faces in den Blattknoten befinden, da die Schnittpunkttests von einem Strahl mit einer Box weniger rechenintensiv sind, als jene von Strahl mit Dreieck. Eine Anzahl von 2–16 Faces pro Blatt scheint eine sinnvolle Größenordnung zu sein. Abbildung 8.7 zeigt die Blattknoten einer erstellten BVH für ein Eichhörnchenmodell zusammen mit der Silhouette des Modells zum Vergleich.

Durchwandern einer BVH auf der GPU. Um den Baum auf der GPU zu durchlaufen ist es notwendig, selbst einen Stack zu verwalten. Zwar wird in [HDW⁺11] ein Algorithmus zur Stack-losen Durchwanderung vorgeschlagen, dieser liegt in der Ausführungsgeschwindigkeit jedoch hinter vergleichbaren Stack-basierten Verfahren zurück. Ohnehin muss im Stack nicht eine komplette Knotenstruktur abgelegt werden, sondern es reicht, sich den Index zu merken, an dem der jeweilige Knoten im globalen Speicher liegt. Die benötigte maximale Stackgröße entspricht der maximal erreichten Tiefe des Binärbaumes. Hierfür wird beim Erzeugen des Baumes die maximale Tiefe in einer Variable gespeichert.

Der Ablauf für das Durchwandern ist in Algorithmus 8.3 beschrieben. Einstiegspunkt ist in jedem Fall der Root-Knoten des Baumes. Trifft der Strahl dessen Bounding Box, werden die beiden Kindknoten auf die gleiche Weise getestet. Getroffene Kindknoten werden auf den Stack gesetzt. Sollen beide Kindknoten auf dem Stack abgelegt werden, wird zuerst der Knoten hinzugefügt, dessen Schnittpunkt mit

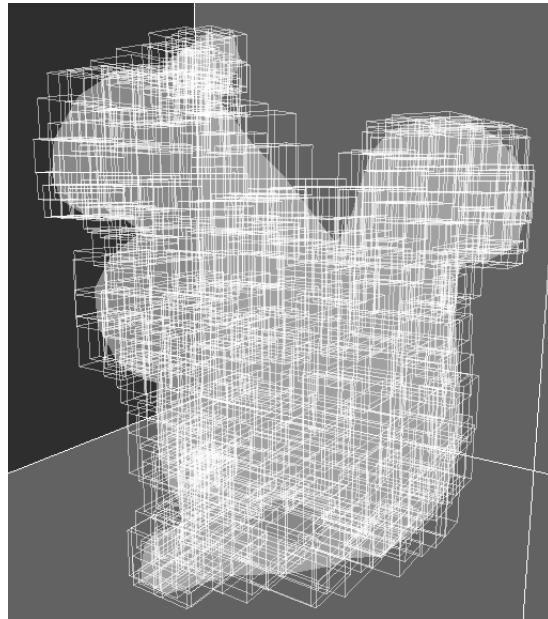


Abbildung 8.7: BVH-Blattknoten eines Eichhörnchen-Modells.

dem Strahl weiter vom Strahlenursprung entfernt ist. Dadurch sitzt der näherliegende Knoten ganz oben auf dem Stack und wird in der nächsten Iteration zuerst getestet. Abbildung 8.8 zeigt einen beispielhaften Zustand beim Prüfen der BVH, nachdem bereits ein paar Knoten getestet wurden und andere Knoten sich nun auf dem Stack befinden.

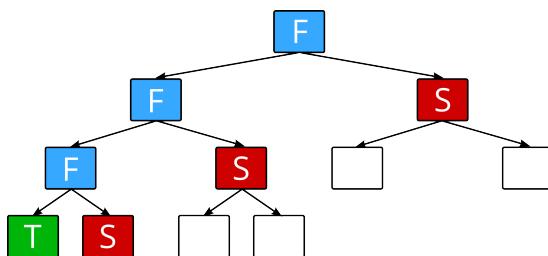


Abbildung 8.8: Blau markierte Knoten (F) wurden fertig geprüft, der grün markierte (T) wird aktuell getestet, rot markierte (S) liegen auf dem Stack.

Nach dem ersten Durchgang können Knoten auch schon abgelehnt werden, wenn ihr Schnittpunkt mit dem Strahl weiter entfernt ist als der bisher naheliegendste, gefundene Treffer mit einem Face. Auf Schnittpunkte mit Faces wird immer dann geprüft, wenn es sich bei einem Knoten um einen Blattknoten handelt.

Algorithmus 8.3 Durchwandern der BVH.

```

1:  $R$                                      ▷ ray into the scene
2:  $N \leftarrow N_{root}$                    ▷ BVH node
3:  $I_{best} \leftarrow \text{NULL}$            ▷ closest intersection of ray with a node
4: push  $N$  on stack

5: while stack is not empty do
6:    $N \leftarrow \text{pop from stack}$ 
7:   if INTERSECT( $R, N$ ) and  $N$  is leaf node then
8:      $I \leftarrow \text{INTERSECTFACES}(R, N)$ 
9:     if  $I$  is closer than  $I_{best}$  then
10:     $I_{best} \leftarrow I$ 
11:   end if
12: end if

13:   $I_{left} \leftarrow \text{INTERSECT}(R, N_{left})$ 
14:   $I_{right} \leftarrow \text{INTERSECT}(R, N_{right})$ 
15:  if  $I_{left}$  is closer than  $I_{right}$  then
16:    push  $N_{right}$  on stack
17:    push  $N_{left}$  on stack
18:  else
19:    push  $N_{left}$  on stack
20:    push  $N_{right}$  on stack
21:  end if
22: end while

23: update  $R$  with  $I_{best}$ 

```

8.3.2 Schnittpunkttests

Tests, ob ein Strahl sich mit einem anderen Objekt, wie einer Box oder einem Dreieck, schneidet, sind die am häufigsten aufgerufenen Funktionen während dem Path Tracing. Entsprechend müssen sie besonders optimiert sein.

Test von Strahl mit Bounding Box. Für die Schnittpunkttests der Strahlen mit den BVH-Knoten wird hier das Verfahren aus [WBMS05] mit einer Optimierung von [onl10] verwendet (Listing 8.3.) Voraussetzung dafür ist, dass es sich um eine Axis-Aligned Bounding Box (AABB) handelt.

Die AABB ist gegeben durch den Minimal- und Maximalpunkt. Der Strahl ist gegeben als $r = o + t \cdot \hat{d}$, wobei o der Ursprung des Strahls, \hat{d} der Richtungsvektor und t der Längen-Faktor für die Richtung ist. Für ein gegebenes t erhält man einen Punkt auf dem Strahl. Ziel des Algorithmus ist es, zwei Werte für t zu bestimmen: Den Eintrittspunkt in die Bounding Box und den Austrittspunkt – insofern der Strahl

Listing 8.3: Strahl-Box-Schnittpunkttest nach [WBMS05, onl10].

```

1 bool intersectBox(
2     ray4* ray, float4 bbMin, float4 bbMax, float* tNear, float* tFar
3 ) {
4     float4 invDir = native_recip( ray->dir ); // 1.0f / ray->dir;
5
6     float4 t1 = ( bbMin - ray->origin ) * invDir;
7     float4 tMax = ( bbMax - ray->origin ) * invDir;
8     float4 tMin = fmin( t1, tMax );
9     tMax = fmax( t1, tMax );
10
11    *tNear = fmax( fmax( tMin.x, tMin.y ), tMin.z );
12    *tFar = fmin( fmin( tMax.x, tMax.y ), tMax.z );
13
14    return ( *tNear <= *tFar );
15 }
```

die AABB denn schneidet. Diese Bestimmungen finden in 2D statt, denn jede Seite der Bounding Box kann als Ebene parallel zu einer der Achsen betrachtet werden. Damit hat man sechs Schnittpunkttests – zwei für jede Seite der AABB. Dies lässt sich effizient in zwei Berechnungen erledigen:

$$\mathbf{u} = (\mathbf{b}_{min} - \mathbf{o}) \cdot \frac{1}{\hat{d}} \quad \mathbf{v} = (\mathbf{b}_{max} - \mathbf{o}) \cdot \frac{1}{\hat{d}} \quad (8.2)$$

wobei \mathbf{b}_{min} der Minimalpunkt und \mathbf{b}_{max} der Maximalpunkt der AABB ist. Bei \mathbf{u}, \mathbf{v} handelt es sich um Vektoren. Als nächstes gruppiert man die kleinsten bzw. größten Werte jeder Koordinate in Vektoren:

$$\mathbf{t}_{min} = \begin{pmatrix} \min(\mathbf{u}_x, \mathbf{v}_x) \\ \min(\mathbf{u}_y, \mathbf{v}_y) \\ \min(\mathbf{u}_z, \mathbf{v}_z) \end{pmatrix} \quad \mathbf{t}_{max} = \begin{pmatrix} \max(\mathbf{u}_x, \mathbf{v}_x) \\ \max(\mathbf{u}_y, \mathbf{v}_y) \\ \max(\mathbf{u}_z, \mathbf{v}_z) \end{pmatrix} \quad (8.3)$$

Aus der Gruppe der kleinsten Werte \mathbf{t}_{min} wählt man den größten Wert für t_{near} , während aus der Gruppe der größten Werte \mathbf{t}_{max} wiederum der kleinste Wert für t_{far} genommen wird:

$$t_{near} = \max \mathbf{t}_{min} \quad t_{far} = \max \mathbf{t}_{max} \quad (8.4)$$

Dieser Zusammenhang wird klarer, wenn man sich Abbildung 8.9 anschaut. Zusätzlich muss gelten, dass $t_{near} \leq t_{far}$. Wird diese Bedingung verletzt, schneidet

der Strahl nicht die AABB.

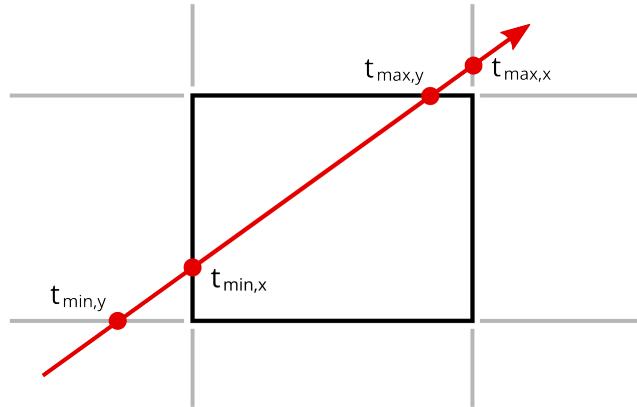


Abbildung 8.9: Schnittpunkttest von Strahl mit Box in 2D.

Test von Strahl mit Dreieck. Listing 8.4 ist der Schnittpunkttest eines Strahls mit einem Dreieck im 3-dimensionalen Raum nach [MT05]. Zuerst wird ein Schnittpunkt des Strahls mit der Ebene gesucht, in der das Dreieck liegt. Findet sich keiner Schnittpunkt – z.B. weil die beiden parallel zueinander liegen – kann der Test bereits beendet werden. Andernfalls muss als nächstes geprüft werden, ob der gefundene Schnittpunkt auch innerhalb des Dreiecks liegt. Hierfür werden baryzentrische Koordinaten genutzt: u und v im Code, die dritte Komponente w ist implizit durch die anderen beiden gegeben mit $w = 1 - u - v$. Diese beschreiben den Punkt relativ zu den drei Eckpunkten. Des Weiteren gilt, dass $0 \leq u, v, w \leq 1$ und $u + v < 1$. Werden diese Bedingungen nicht erfüllt, liegt der Schnittpunkt außerhalb des Dreiecks.

Smooth-Shading mit Per-Vertex-Normalen. Um aus Dreiecken bestehende Modelle glatter erscheinen zu lassen, gibt es zwei Möglichkeiten: Entweder, man erhöht drastisch die Anzahl an Faces, oder gibt den Vertices unterschiedliche Normalen, je nachdem, welches Face sie gerade beschreiben. Für eine geglättete Fläche werden die Vertex-Normalen zu einem Face dann gemäß den anliegenden Faces ausgerichtet, sodass die Übergänge zwischen den Faces glatter erscheinen. Trifft ein Strahl ein Face, wird die Normale dann speziell für die Trefferposition auf dem Face berechnet. Dafür wird die Trefferposition in Form von baryzentrischen Koordinaten benötigt, wie sie im Strahl-Dreieck-Test aus Listing 8.4 in der Variable tuv gespeichert werden. Dann erhält man die Normale wie in Listing 8.5 beschrieben.

Dabei bleibt das Problem, dass der Strahl-Dreiecks-Test nicht die Normalen je Vertex verwenden kann, da die Trefferposition für die Gewichtung noch nicht bekannt

Listing 8.4: Strahl-Dreieck-Schnittpunkttest nach Möller und Trumbore [MT05].

```

1  bool checkFaceIntersection(
2      ray4* ray, face_t face, float3* tuv, float tNear, float tFar
3  ) {
4      float3 edge1 = face.b - face.a;
5      float3 edge2 = face.c - face.a;
6      float3 tVec = ray->origin - face.a;
7      float3 pVec = cross( ray->dir, edge2 );
8      float3 qVec = cross( tVec, edge1 );
9      float invDet = native_recip( dot( edge1, pVec ) );
10
11     tuv->x = dot( edge2, qVec ) * invDet;
12
13     if( tuv->x < EPSILON ||
14         fmax( tuv->x - tFar, tNear - tuv->x ) > EPSILON
15     ) {
16         tuv->x = -2.0f;
17         return false;
18     }
19
20     tuv->y = dot( tVec, pVec ) * invDet;
21     tuv->z = dot( ray->dir.xyz, qVec ) * invDet;
22     if( fmin( tuv->y, tuv->z ) < 0.0f || tuv->y > 1.0f ||
23         tuv->y + tuv->z > 1.0f
24     ) {
25         tuv->x = -2.0f;
26         return false;
27     }
28
29     return true;
30 }
```

Listing 8.5: Berechnung der Normale anhand Per-Vertex-Normalen.

```

1  float3 getTriangleNormal(
2      float3 tuv, float3 an, float3 bn, float3 cn
3  ) {
4      float w = 1.0f - tuv.y - tuv.z;
5      return fast_normalize( w * an + tuv.y * bn + tuv.z * cn );
6 }
```

ist. Wenn ein Strahl nun einen Bereich trifft, der durch die Glättung nicht zu sehen sein dürfte, resultiert dies in einer falsch ausgerichteten Normalen. Typische

Erscheinungsmerkmale im Bild sind schwarze Bereiche entlang der Silhouette eines Objektes. Um dies zu vermeiden ist eine Möglichkeit, den Schnittpunkttest mit der berechneten Normalen zu wiederholen. Effizienter ist es aber, die Ausrichtung der Normalen mit der des Strahles zu vergleichen und je nach Resultat dann den gefundenen Schnittpunkt zu verwerfen.

Sei \hat{n}_f die berechnete Normale für das getroffene Face und \hat{d} die Richtung des Strahls. Dann wird ein Schnittpunkt abgelehnt, wenn $(\hat{n}_f \cdot -\hat{d}) < 0$ gilt. Die Normale beschreibt eine Hemisphäre über dem Punkt. Eine Fläche kann nur genau dann von einem Strahl getroffen worden sein, wenn der Strahl durch jene Hemisphäre auf die Trefferposition zeigt. Dies wird über das Skalarprodukt ausgedrückt, welches dem Kosinus des Winkels zwischen zwei Vektoren entspricht. Ist der Wert negativ, dann schneidet der Strahl das Face von der Rückseite aus – oder in diesem Fall bedeutet dies, die Normale zeigt in die falsche Richtung. Ein Bild-Vergleich von vor und nach dem Normalen-Test ist in Abbildung 8.10 zu sehen.

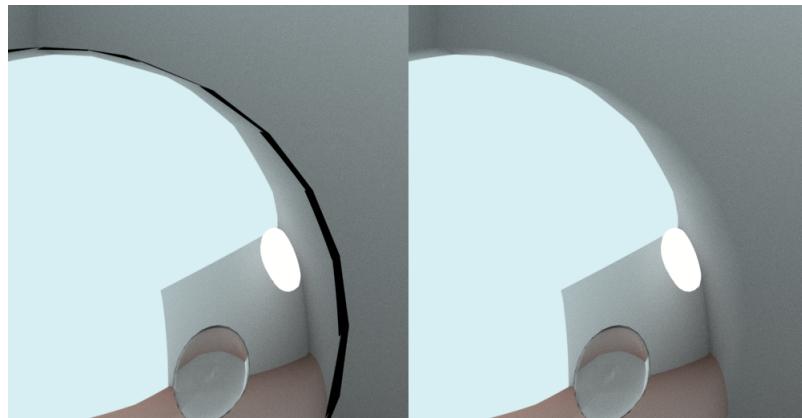


Abbildung 8.10: *Links: Geglättete Kugel mit schwarzen Bereichen am Rand. Rechts: Schwarze Bereiche behoben durch Test der Normalen.*

Ein Nebeneffekt dieses Verfahrens zur Treffer-Ablehnung ist, dass nun auf die gesamte Szene *Backface Culling* angewendet wird. Backface Culling ist eigentlich eine Technik zur Optimierung aus dem Bereich der Rasterisierungsverfahren, bei der die Rückseite von Faces nicht gerendert werden, was Rechenaufwand reduziert. Im Path Tracing beschleunigt Backface Culling allerdings nicht die Ausführungs geschwindigkeit, da für die Normale an einem Punkt in jedem Fall zuerst der Schnittpunkttest von Strahl mit Dreieck durchgeführt werden muss und erst anschließend der Test zur Ablehnung durchgeführt werden kann.

Es werden also alle Flächen ignoriert, deren Normalen nicht in Richtung des aktuellen Strahls zeigen. Dies ist jedoch problematisch für transparente Materialien, wie z.B. eine Glaskugel. Beim Austritt aus dem Objekt kommt der Strahl aus

dem Inneren, während die Normalen nach außen zeigen. Dies würde den Lichtbrechungseffekt verfälschen, da der Austritt aus dem Material fehlen würde. Als Workaround wird daher zusätzlich abgefragt, ob das getroffene Material transparent ist, und in diesem Fall auf den Normalen-Test verzichtet. Die Silhouette von geglätteten, transparenten Objekten weiß somit leider wieder Fehler in Form von dunklen Bereichen auf. Möchte man auf den Workaroung verzichten und auch transparente Objekte glätten, müsste man sich zusätzlich für einen Strahl merken, ob er an der letzten getroffenen Fläche in ein transparentes Objekt eingetreten ist. Dies bedeutet einen größeren Speicheraufwand und berücksichtigt zudem keine nicht-transparenten Objekte innerhalb von einem transparenten Objekt, wie z.B. Schneekugeln oder Steine in einem Fluss.

8.3.3 Abbruch mit Russisch Roulette

Bisher wird ein Pfad solange verlängert, bis ein Strahl entweder nichts trifft, eine Lichtquelle trifft, oder die Pfadlänge das vorgegebene Maximum erreicht. Es kann aber sinnvoll sein, Pfade frühzeitig zu beenden, wenn abzusehen ist, dass selbst bei Erreichen einer Lichtquelle kaum Energie zum Bildpunkt beitragen werden. Dies ist der Fall, wenn der Pfad Oberflächen getroffen hat, die viel Energie absorbieren.

Zu diesem Zweck wird eine Russisch-Roulette-Terminierung eingeführt. Ab einer gewählten Pfad-Mindestlänge (hier wurde zwei gewählt), werden Pfade mit einer gewissen Wahrscheinlichkeit zufällig beendet. Als Kriterium für den Abbruch wird die aktuelle Energie mit einem Zufallswert $r \in [0, 1]$ verglichen. Für den Energiewert wird der maximale Wert aus der aktuellen SPD verwendet. Die Implementierung ist unkompliziert und ist in Listing 8.6 zu sehen.

Listing 8.6: Russisch Roulette zur frühzeitigen Terminierung eines Pfades.

```

1 float maxValSpd = 0.0f;
2 for( int i = 0; i < SPD_SAMPLES; i++ ) {
3     maxValSpd = fmax( spd[i], maxValSpd );
4 }
5 if( depth > MIN_DEPTH && maxValSpd < rand( &seed ) ) {
6     break; // end this path
7 }
```

8.4 Antialiasing

Als Antialiasing werden Verfahren zur Kantenglättung in Grafiken bezeichnet. Im Path Tracing ist dies ein sehr simpler Vorgang, der praktisch kaum mehr Rechen-

leistung benötigt. Lediglich der Richtungsvektor \hat{d} des ersten Strahles eines jeden Pixels muss geringfügig verwackelt werden. Dafür erzeugt man einen zufälligen Richtungsvektor \hat{r} wie schon für diffuse Flächen in Kapitel 5.2 mit \hat{d} als Normale. Diesen Zufallsvektor addiert man schließlich mit einer sehr geringen Gewichtung auf \hat{d} . Als Gewichtung lässt sich die Pixelgröße px nehmen, die in Kapitel 5.1 bestimmt wurde. Üblicherweise liegt dieser Wert im Bereich von 0,001. Der neue Richtungsvektor muss anschließend wieder normalisiert werden. Der Unterschied, den Antialiasing macht, ist in Abbildung 8.11 zu sehen.

$$\hat{d} = \hat{d} + \hat{r} \cdot g \quad (8.5)$$

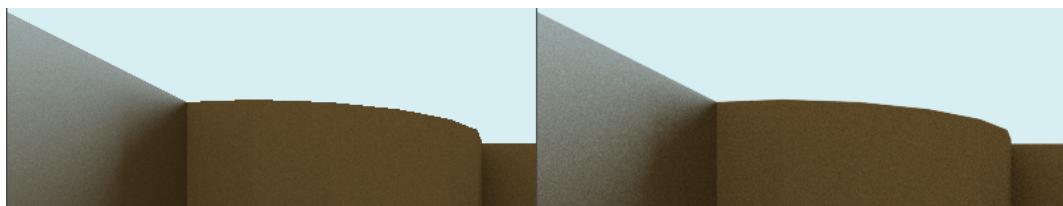


Abbildung 8.11: Treppenbildung ohne Antialiasing (links) und geklättet mit einem Faktor von $g = 0,001$ (rechts).

8.5 Datei-Formate der Modelle

Zu jedem Testmodell gehören drei Dateien: OBJ, MTL und SPEC. Die OBJ- und MTL-Datei wurden von der Modellierungssoftware Blender⁸ exportiert. Die SPEC-Datei ist ein selbst definiertes Format zur Beschreibung der verwendeten SPDs.

Die OBJ-Datei beschreibt die Geometrie der Szene. Eine Zeile beginnt mit einem Schlüsselbegriff wie v für einen Vertex, gefolgt von einem dazu passenden Wert bzw. Werten. Tabelle 8.1 führt die wichtigsten Inhalte einer OBJ-Datei auf. Für den Schlüssel o ist dabei die Position in der Datei wichtig, da er den Beginn eines Objektes einleitet, wie sie in Kapitel 8.3.1 im Zusammenhang mit der BVH erwähnt wurden. Alle folgenden Faces gehören speziell zu diesem Objekt bis ein neues o kommt oder die Datei zu Ende ist. Ebenso ist die Position von usemtl von Bedeutung, da alle nachfolgenden Faces das benannte Material verwenden.

Die MTL-Datei definiert die Materialien, die einer Fläche zugeordnet werden können. Tabelle 8.2 listet im ersten Abschnitt die relevanten Schlüssel auf und

⁸<http://blender.org/>

Tabelle 8.1: OBJ-Schlüsselbegriffe und Werte relevant in diesem Projekt.

Schlüssel	Wert	Beispiel
f	3 Vertices und Normalen	f 76//67 65//57 75//63
o	Name eines Unter-Objektes	o Sphere1
usemtl	Material der Faces	usemtl this_is_a_mirror
v	Vertex-Koordinaten (X, Y, Z)	v 0.177385 0.792878 1.931103
vn	Normalen-Koordinaten (X, Y, Z)	vn 0.0 0.49086 -0.871212

nennt im zweiten Abschnitt die für dieses Projekt hinzugefügten Erweiterungen. Jede BRDF hat für ihre Parameter eigene Einträge erhalten.

Tabelle 8.2: Verwendete MTL-Schlüsselbegriffe und Werte.

Schlüssel	Wert	Beispiel
d	Transparenz [0,1]	d 0.5
light	Lichtquelle ja (1)/nein (0)	light 1
newmtl	Materialname	newmtl mirror
Ni	Brechungsindex [0,∞)	Ni 1.5
<i>Schlick</i>		
p rough	Isotropie-Faktor [0,1] Diffusität [0,1]	p 0.3 rough 0.8
<i>Shirley-Ashikhmin</i>		
nu	Phong-artiger Exponent für die Spekularität [0,∞)	nu 1000.0
nv	Phong-artiger Exponent für die Spekularität [0,∞)	nv 5.0
Rs	Faktor spekulare Reflektion [0,1]	Rs 0.05
Rd	Faktor diffuse Reflektion [0,1]	Rd 1.0

Die **SPEC-Datei** ist eine für dieses Projekt entworfene Datei für SPDs. Sie verwendet JSON⁹ als Notationsform. Listing 8.7 zeigt ein Beispiel für eine SPEC-Datei. Unter dem Schlüssel `materials` geschieht die Zuordnung von Material (aus der MTL-Datei) zu SPD, wobei zwischen einer SPD für diffuse Reflektion (`diff`) und einer für spekulare (`spec`) unterschieden werden kann. Der Schlüssel `sky` ist ein vorgegebenes Material und wird für Strahlen verwendet, die keine Fläche getroffen haben, da sie die Bounding Box der Szene verlassen haben. Es handelt sich somit um die Farbe der Skybox, die als Lichtquelle behandelt wird. Unter dem Schlüssel `spectra` werden schließlich die SPDs definiert. Ein SPD wird dabei als Wellenlängen von 380 nm bis 775 nm mit jeweiliger Intensität beschrieben.

⁹JavaScript Object Notation

Listing 8.7: SPEC-Beispiel.

```

1  {
2      "materials": {
3          "Metal": { "diff": "golden", "spec": "golden" }
4      },
5      "sky": "CIE_D65",
6      "spectra": {
7          "CIE_D65": {
8              "380": 0.444027773062, "385": 0.463859369164, ...
9              "770": 0.552525209656, "775": 0.537999524666
10         },
11         ...
12     }
13 }
```

8.6 Darstellung eines Frames mit OpenGL

Die einzelnen Frames werden als 2D-Textur behandelt und an einen Vertex-Shader (Listing 8.8) und Fragment-Shader (Listing 8.9) überreicht. Ein Vertex-Shader kann unter anderem die Position der Vertices verändern, z.B. um sie perspektivisch korrekt zu setzen. Dies geschieht hier jedoch bereits beim Schießen der Strahlen in die Szene. Dem Vertex-Shader wird nur noch das aktuelle Bild als 2D-Texture übergeben, welches dieser Pixel für Pixel ohne Veränderung übernimmt. Ein Fragment-Shader – oder auch Pixel-Shader genannt – kann auf die Farbe eines Pixel einwirken. Doch auch hier wird nur die Farbe jedes einzelnen Pixels des übergebenen Bildes als RGB-Wert ausgelesen und unverändert dargestellt.

Listing 8.8: Vertex-Shader.

```

1 layout( location = 0 ) in vec3 vertex;
2
3 void main( void ) {
4     gl_Position = vec4( vertex, 1.0 );
5 }
```

8.7 Generelle Optimierungen für OpenCL

Die Ausführungszeiten der einzelnen Kernel kann über ein eigens in OpenCL integriertes Profiling gemessen werden. Dafür muss beim Erstellen der Command-Queue der Parameter CL_QUEUE_PROFILING_ENABLE gesetzt werden. In Listing 8.10

Listing 8.9: Fragment-Shader.

```

1 uniform int width;
2 uniform int height;
3 uniform sampler2D texUnit;
4 out vec4 color;
5
6 void main( void ) {
7     vec2 tCoord = vec2(
8         gl_FragCoord.x / width, gl_FragCoord.y / height
9     );
10    vec3 texture = texture2D( texUnit, tCoord ).rgb;
11    color = vec4( texture, 1.0 );
12 }
```

ist zu sehen, dass dann für ein gegebenes Kernel-Event die Ausführungszeit des Kernels abgefragt werden kann. Voraussetzung dafür ist, dass der Kernel seine Aufgabe abgeschlossen haben muss. Dies stellt man über OpenCL-Events sicher. Dafür wird beim Aufruf eines Kernels eine Variable vom Typ `cl_event` mitgegeben (Zeile 5). Mit der Funktion `clWaitForEvents()` wird dann abgewartet, bis das zur Event-Variable zugehörige Event abgeschlossen ist. In dieser Variable werden diverse Informationen hinterlegt, u.a. wann Ausführungsbeginn und -ende des Kernels war. Aus dem Zeitabstand lässt sich dann die vergangene Zeit berechnen.

Listing 8.10: Messen der Ausführungszeit eines OpenCL-Kernels.

```

1 cl_ulong timeEnd, timeStart;
2 cl_event kernelEvent;
3
4 // Execute the kernel
5 clEnqueueNDRangeKernel( mCommandQueue, kernel, 3, NULL,
6     globalWorkSize, NULL, 0, NULL, &kernelEvent );
7 // Wait for kernel to finish
8 clWaitForEvents( 1, &kernelEvent );
9
10 // Get the start and end time from the kernel event
11 clGetEventProfilingInfo( kernelEvent, CL_PROFILING_COMMAND_START,
12     sizeof( timeStart ), &timeStart, NULL );
13 clGetEventProfilingInfo( kernelEvent, CL_PROFILING_COMMAND_END,
14     sizeof( timeEnd ), &timeEnd, NULL );
15
16 // Convert to milliseconds
17 double kernelTimeMs = (double) ( timeEnd - timeStart ) / 1000000.0f;
```

Die meisten der folgenden Optimierungshinweise finden sich im „NVIDIA OpenCL Best Practices Guide“ [NVI09]. Hier erfolgte eine Auswahl der wichtigsten Tipps zur Verbesserung, auf die bei der Entwicklung geachtet werden sollte.

Speicherzugriffe. Bei der Verwendung des GPU-Speichers sollten die verschiedenen Hierarchien bedacht werden. Globaler Speicher ist am größten, Zugriffe aber auch am langsamsten. In abnehmender Größe, aber zunehmender Geschwindigkeit folgen der lokale Speicher einer Workgroup, dann der private Speicher eines einzelnen Threads. Zusätzlich gibt es den konstanten Speicher – ein kleiner, auf Geschwindigkeit optimierter read-only Bereich im globalen Speicher. Abbildung 8.12 gibt noch einmal eine Übersicht über die Speicher-Hierarchie.

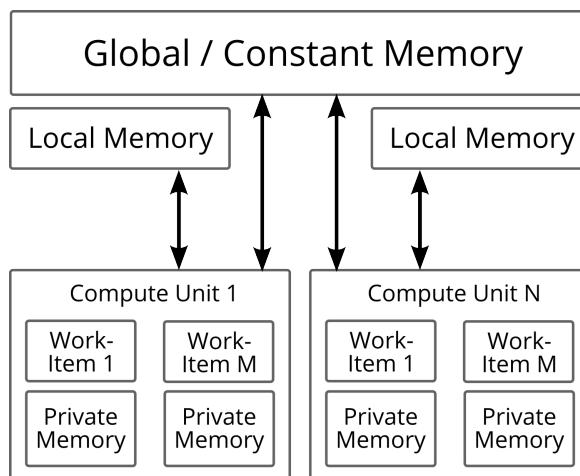


Abbildung 8.12: Die Speicher-Hierarchie in OpenCL.

Wird auf eine Listen-Struktur im Speicher zugegriffen – insbesondere der globale oder lokale –, dann sollten die Zugriffe geordnet erfolgen. Daten, die in einer Liste aufeinanderfolgende Indizes haben, liegen i.d.R. auch im Speicher nebeneinander. Wird nun auf ein Element zugegriffen, wird nicht nur dieses eine aus dem Speicher gelesen, sondern gleich mehrere und in einem schnellen Zwischenspeicher als sogenannte Cache Line abgelegt. Hierdurch erfolgen Zugriffe auf einige nachfolgende Elemente schneller.

Sparsam mit privatem Speicher umgehen. Der private Speicher eines Threads ist schnell, aber nur sehr klein. Nehmen Variablen zu viel Platz ein, werden Daten in langsamere Speicherbereiche ausgelagert. Um dies zu vermeiden, sollten möglichst wenig Variablen verwendet und insbesondere größere Structs vermieden werden. Listing 8.11 zeigt ein Beispiel, in dem eine private Variable `tri` des selbst-definierten Datentyps `myTriangle` eingespart wird. Dies wird erreicht, in-

dem das Resultat des Array-Zugriffes `triangles[node->triangleIndex]` direkt an die Funktionen übergeben wird. Dadurch findet zwar ein weiterer Array-Zugriff statt, dafür belegt der Wert keinen Platz im privaten Speicher.

Listing 8.11: Beispiel für sparsameren Umgang mit dem privaten Speicher.

```

1 // Bad
2 void intersect(
3     myRay* ray, myNode* node, global myTriangle* triangles
4 ) {
5     myTriangle tri = triangles[node->triangleIndex];
6     float4 tuv;
7     bool hit = intersectRayTriangle( ray, tri, &tuv );
8
9     if( hit ) { ray->normal = getTriangleNormal( tri, tuv ); }
10 }
11
12 // Good
13 void intersect(
14     myRay* ray, myNode* node, global myTriangle* triangles
15 ) {
16     float4 tuv;
17     #define TRI triangles[node->triangleIndex]
18     bool hit = intersectRayTriangle( ray, TRI, &tuv );
19
20     if( hit ) { ray->normal = getTriangleNormal( TRI, tuv ); }
21     #undef TRI
22 }
```

Branching. Durch Bedingungen wie `if` verzweigt der Code-Ablauf, was dem Parallelismus entgegenwirkt und die Anwendung ausbremsst. Vermieden werden sollten insbesondere frühe Verzweigungen die den weiteren Verlauf in sehr verschiedene Richtungen lenken. Auch können sich tief verschachtelte Bedingungen negativ auswirken. Listing 8.12 zeigt ein Minimal-Beispiel.

Native-Funktionen. Für einige mathematischen Funktionen existieren Alternativen mit `native_` als Präfix im Namen, z.B. `native_divide(float a, float b)` zum Dividieren oder `native_cos(float a)` für den Kosinus. Diese Funktionen verwenden die Implementierung der Hardware – läuft die Anwendung auf der GPU, dann entsprechend die GPU-Implementierung der Funktion. Native-Funktionen sind in der Regel schneller, dafür besteht die Gefahr, dass die Implementierung einer anderen Hardware leichte Abweichungen in den Ergebnissen liefert.

Listing 8.12: Beispiel für Branching im Code.

```
1 float4 ray;
2
3 // Bad
4 if( isDiff ) { ray = diffuseRay(); }
5 else { ray = getSpecularRay(); }
6
7 // Good
8 ray = isDiff * diffuseRay() + ( 1.0f - isDiff ) * specularRay();
```

Fast-Funktionen. Für einige geometrischen Funktionen existieren Alternativen mit `fast_` als Präfix im Namen. Geometrische Funktionen sind solche, die Vektoren als Parameter haben, z.B. `fast_normalize(float3 myVector)`. Die Fast-Funktionen sind schneller, da sie aufgrund geringerer `float`-Präzision ungenauer arbeiten.

pown(x,n) als Multiplikation ausschreiben. Die Funktion `pown(x,n)` errechnet x^n unter der Bedingung, dass $n \in \mathbb{N}$. Ist der Exponent n aber bereits bekannt, ist es sinnvoller, dies als Multiplikation auszuschreiben. So geschehen für den Fresnel-Faktor in Listing 8.13.

Listing 8.13: Fresnel-Faktor

```
1 inline float fresnel( const float u, const float c ) {
2     const float v = 1.0f - u;
3     return c + ( 1.0f - c ) * v * v * v * v * v;
4 }
```

9 Evaluation

Die folgenden Messungen erfolgten auf einer NVIDIA GTX 560 Ti. Die Größen der verschiedenen Speicher-Hierarchien sind wie folgt: 2047 MB für Global, 64 KB für Constant und 48 KB für Local. Des Weiteren hat der globale Cache eine Größe von 128 KB mit einer Cache Line von 128 Bytes. Die maximale Workgroup-Größe liegt bei 1024. Als CPU wurde ein Intel Core i5-2400 (3.10 GHz) verwendet.

BRDFs

Sowohl in der BRDF von Schlick als auch von Shirley und Ashikhmin ist es möglich, Flächen eine Abstufung zwischen perfekt spekular und perfekt diffus zuzuweisen. Abbildung 9.1 zeigt für die Schlick-BRDF eine unterschiedlich stark spiegelnde Kugel; genauso Abbildung 9.2 für die BRDF von Shirley-Ashikhmin.

Der Parameter r für die Rauheit bei Schlick liegt im Wertebereich $[0, 1]$. Jedoch scheint der Übergang von spekular zu diffus nicht linear zu geschehen, wie man es vielleicht intuitiv annehmen würde. So ist die Fläche für $r = 0,5$ nach wie vor sehr diffus und was man als einen ausgeglichenen, matten Zwischenschritt zwischen spekular und diffus sehen würde, liegt ungefähr bei $r = 0,1$.

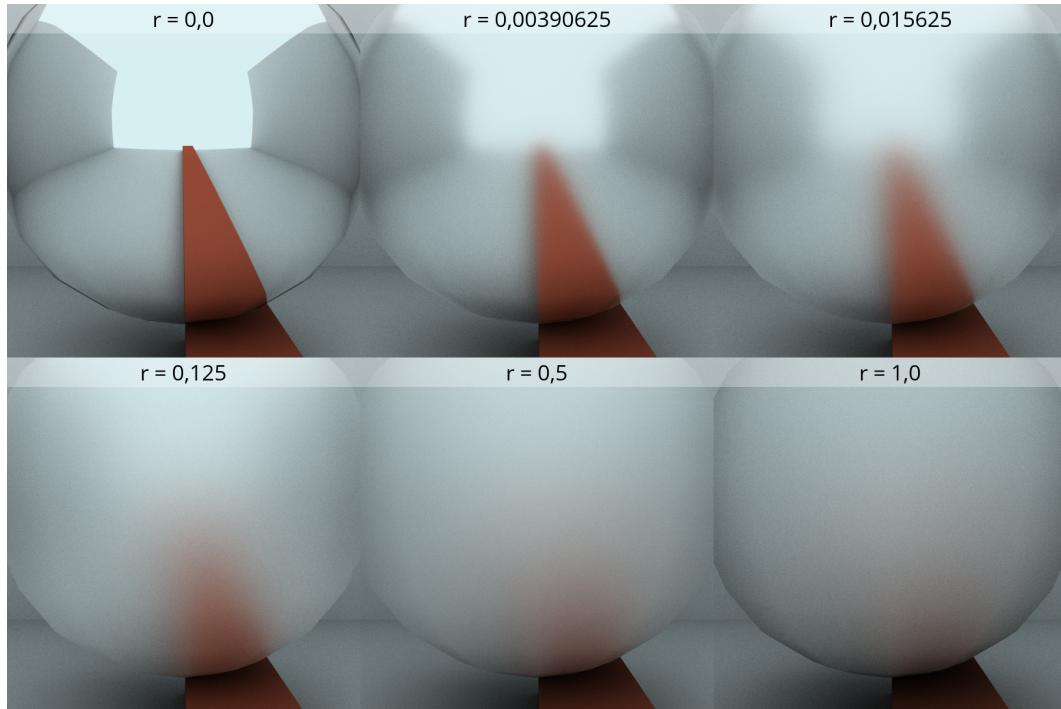


Abbildung 9.1: Schlick-BRDF. Übergang von spekular zu diffus für Rauheit r .

Bei der BRDF von Shirley-Ashikhmin wird für die Parameter n_u, n_v lediglich vorgegeben, dass sie positiv sein müssen. Solange keine Anisotropie betrachtet wird, gilt im folgenden dass $n_u = n_v$. Für eine nahezu perfekt spekulare Fläche sind Werte von mindestens 100.000 notwendig. Ein relativ fließender Übergang von spekular zu diffus wird erreicht, wenn man die Werte logarithmisch abläuft – also z.B. 1000, 100, 10, 1, wie hier geschehen.

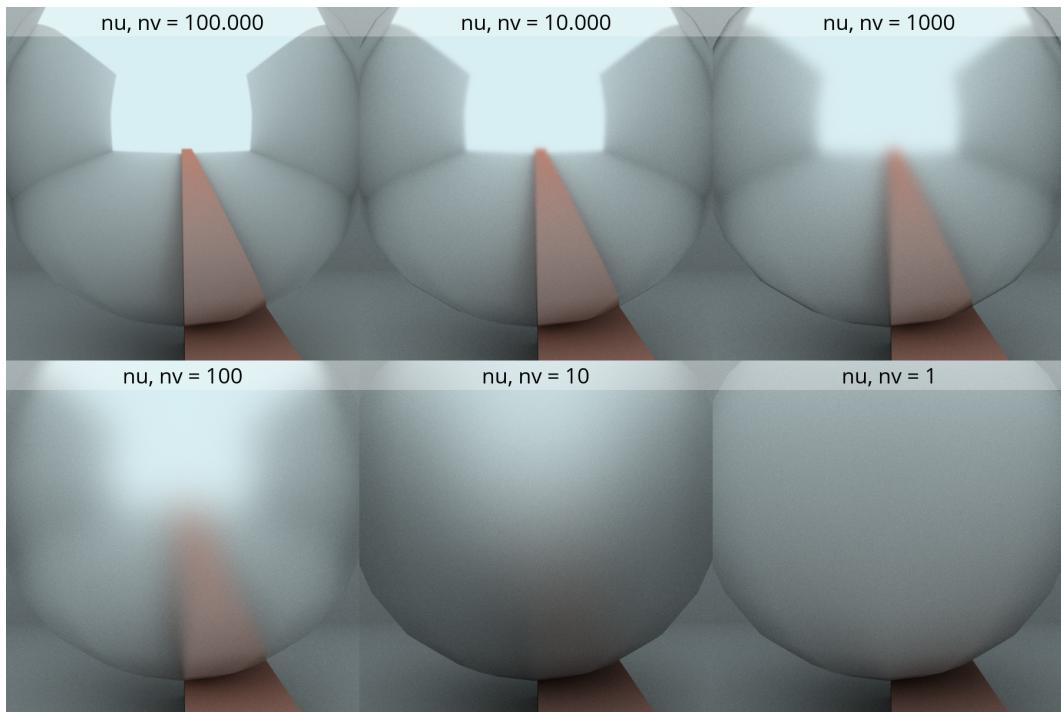


Abbildung 9.2: Shirley-Ashikhmin-BRDF. Übergang von spekular zu diffus für die Parameter n_u, n_v .

Eine weitere Materialeigenschaft, die von beiden BRDFs umgesetzt wird, ist Anisotropie. Im Schlick-BRDF wird diese über den Parameter p reguliert, wobei $p = 1$ eine perfekt isotrope Fläche und $p = 0$ eine anisotrope Fläche ist. Abbildung 9.3 zeigt eine Szene mit einer Kugel für verschiedene Werte für p , bei konstanter Rauheit $r = 0,1$.

In der Shirley-Ashikhmin-BRDF hingegen werden die zwei Parameter n_u, n_v verwendet, wie auch schon für den Übergang von spekular zu diffus. Anisotrope Effekte werden erzeugt, indem n_u und n_v von einander abweichen. Abbildung 9.4 zeigt in der ersten Reihe den Übergang für $n_u = 100.000$ bis $n_u = 100$ bei konstantem $n_v = 10$, während in der zweiten Reihe die Werte vertauscht sind. Wie zu sehen ist, kehrt sich dadurch auch die Richtung der Reflektionen auf der Oberfläche um bzw. erscheinen um 90° gedreht.

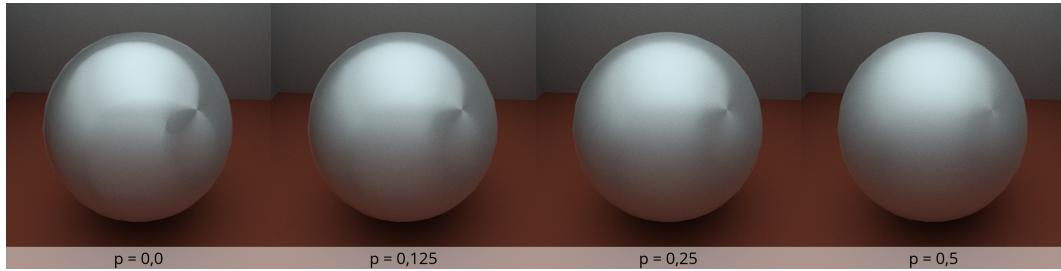


Abbildung 9.3: Anisotropie in der Schlick-BRDF bei Rauheit $r = 0,1$.

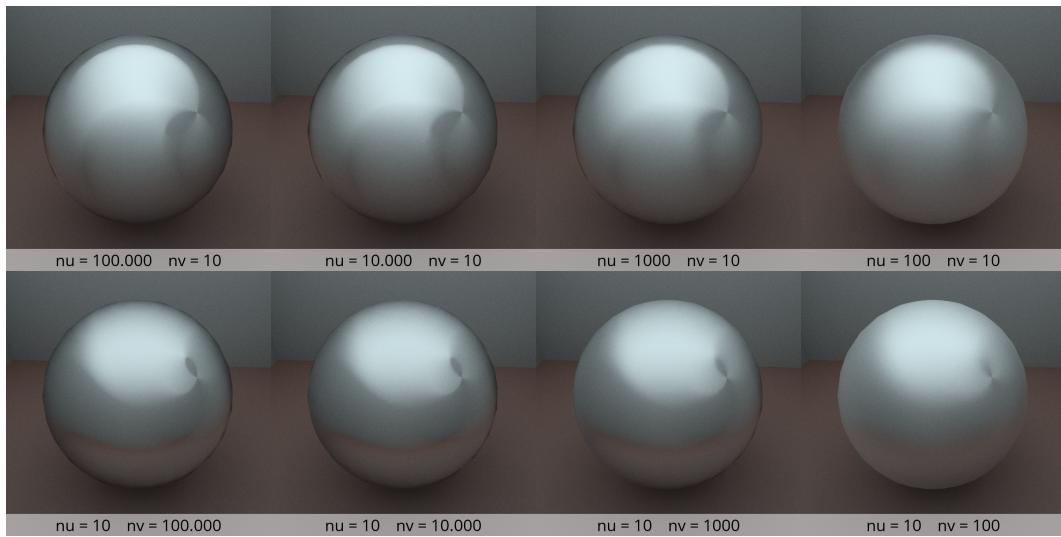


Abbildung 9.4: Anisotropie in der Shirley-Ashikhmin-BRDF. *Oben:* Nur Parameter n_u wird verändert. *Unten:* Nur Parameter n_v wird verändert.

Abgesehen von der Richtungsinvertierung bei der Shirley-Ashikhmin-BRDF durch die Parameter n_u, n_v ist es derzeit nicht möglich, der Anisotropie eine Richtung vorzugeben. Während die BRDFs dem nicht im Wege stehen, ist die Implementierung in diesem Projekt jedoch nicht vollständig. Es muss noch eine weitere Materialeigenschaft definiert werden können, die die Orientierung der Rillen auf der Oberfläche angibt und eine Möglichkeit, zwischen kreisförmigen Rillen (wie z.B. auf der Unterseite eines Topfes) und geraden Linien zu unterscheiden.

Helligkeit der Szene

Eine große Schwäche des Path Tracings ist, dass es keine guten Resultate für dunkle Szenen liefert. Sind Lichtquellen nur sehr klein oder schwer zu erreichen, gelingt dies nur den wenigsten Pfaden, was in einem sehr dunklen Bild resultiert. Die Bilder in Abbildung 9.5 wurden mit einer maximalen Pfadlänge von 5 Strahlen generiert – durch Russisch Roulette wurden manche jedoch frühzeitig abgebrochen.

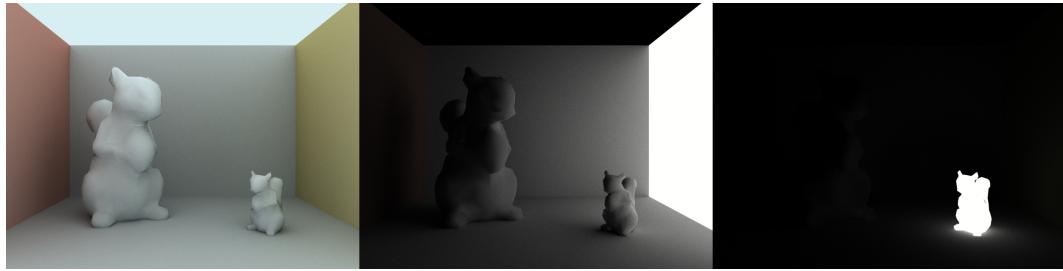


Abbildung 9.5: *Links:* Decke und Frontwand sind offen; die gesamte Umgebung ist eine Lichtquelle. *Mitte:* Nur die rechte Wand ist eine Lichtquelle. *Rechts:* Nur das kleine Eichhörnchen ist eine Lichtquelle.

Im ersten Bild ist die Szene noch hell beleuchtet, da die Strahlen mit hoher Wahrscheinlichkeit eine Lichtquelle finden. Die Szene besteht aus einer Box mit zwei Eichhörnchen darin, wobei sowohl die Decke als auch die Frontwand fehlen. Die gesamte Umgebung um die Szene herum ist eine Lichtquelle. Im zweiten Bild ist nur noch die rechte Wand eine Lichtquelle. Wie zu sehen, ist die Szene bereits beträchtlich dunkler, was auch daran liegt, dass Strahlen die Box verlassen und daher keine Chance mehr haben, die Lichtquelle zu finden. Im dritten Bild ist nur das kleinere Eichhörnchen eine Lichtquelle. Außer in der nächsten Umgebung um das Eichhörnchen herum ist kaum noch etwas von der Szene zu erkennen.

Lichtbrechung

Abbildung 9.6 zeigt ein transparentes Eichhörnchenmodell, dass für verschiedene Brechungsindizes N_i gerendert wurde. Wie zu erwarten werden Lichtstrahlen umso stärker gebrochen, je höher der Brechungsindex ist. Für $N_i = 1,2$ ist die Brechung noch recht schwach, während für $N_i = 1,5$ – was in etwa Glas entspricht – die Brechung deutlich zu erkennen ist. Für $N_i = 1,7$ ist der Effekt noch einmal stärker. Dabei fällt auf, dass mit höherem Brechungsindex auch vermehrt dunkle Regionen im Modell auftauchen – z.B. an der Schnauze, den Ohr-Ansätzen und Hinterpfoten-Ansätzen. Dies ist auf die limitierte Pfadlänge zurückzuführen. In solchen dunklen Regionen wurde der Pfad abgebrochen, bevor er eine Lichtquelle erreichen konnte. Als maximale Pfadlänge wurde eine Länge von 5 eingestellt, mit maximal 5 Verlängerungen, wenn ein Strahl auf ein transparentes Material trifft. Potentiell konnte also eine Pfadlänge von 10 erreicht werden, jedoch wurden energiearme Pfade durch Russisch Roulette auch früher abgebrochen.

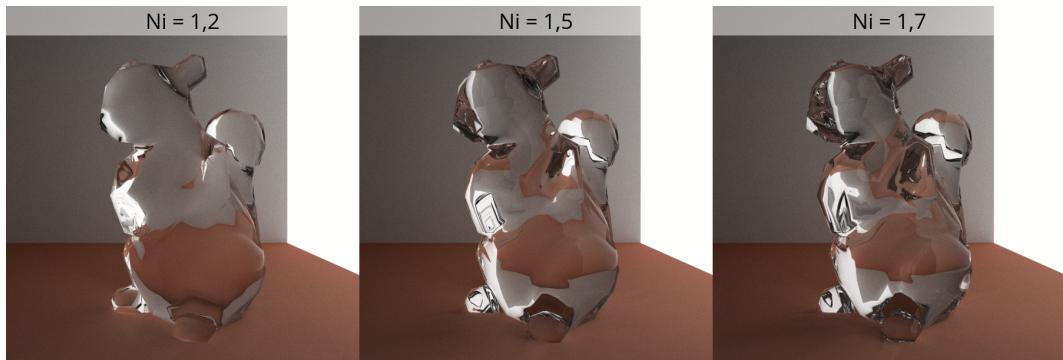


Abbildung 9.6: Transparentes Objekt mit verschiedenen Brechungsindizes Ni .

Kaustiken

Kaustiken – also Bereiche, in denen Licht gebündelt wird – treten wie gewünscht als natürliche Folge des Path Tracings auf. Durch Lichtbrechung in transparenten Objekten wie Glas oder Reflektion an gekrümmten, spiegelnden Flächen kommen entsprechende Pfade zustande. In Abbildung 9.7 sind Kaustiken deutlich im Schattenbereich des Eichhörnchens und schwach im Bereich vor den Füßen zu sehen, die durch Lichtbrechung im Glas zustandekommen. Das Bild wurde nachträglich stark aufgehellt, damit der Effekt besser zu erkennen ist.

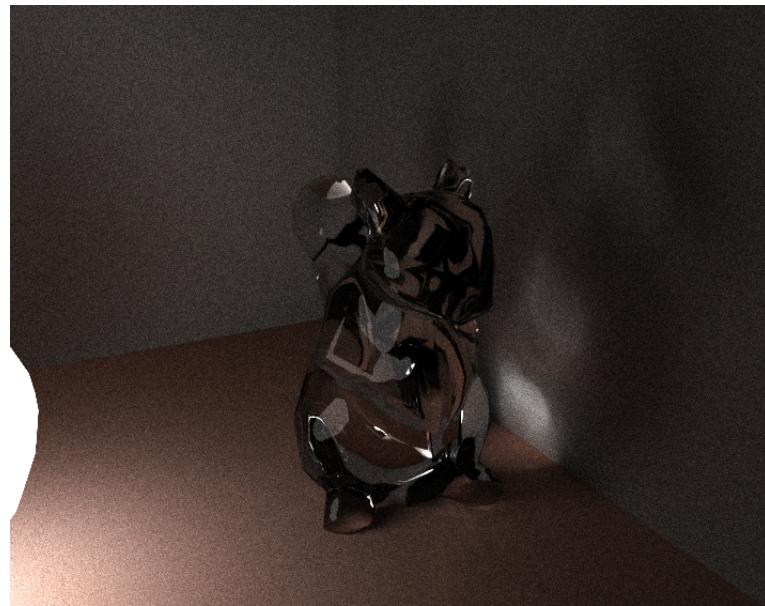


Abbildung 9.7: Strahlen von der Lichtquelle werden durch das gläserne Eichhörnchen gebündelt. (*Helligkeit wurde nachbearbeitet.*)

Während diese Effekte also an sich zu beobachten sind, gelingt dies am besten nur in dunklen Szenen, da die Lichtquelle klein sein muss. Ist die Lichtquelle zu groß,

werden die Kaustiken durch die Gesamthelligkeit in der Szene überdeckt. Dies ist auf das gleiche Problem zurückzuführen, das im Abschnitt „Helligkeit der Szene“ diskutiert wurde: Es ist weniger wahrscheinlich, dass Pfade zu kleinen Lichtquellen zustandekommen. Hier muss in Zukunft noch nachgebessert werden.

Performance

Im folgenden sollen diverse Einflüsse von Materialien und Einstellungen der Anwendung auf die Performance untersucht werden. Als Kriterium für die Performance wird die Ausführungszeit des OpenCL-Kernels herangezogen. Als Testmodell wird die Szene in Abbildung 9.8 in einer Auflösung von 800×640 Pixeln gerendert. Der Boden hat eine grüne SPD, während die Wände und Eichhörnchen ein perfektes Weiß (keine Absorption) darstellen.

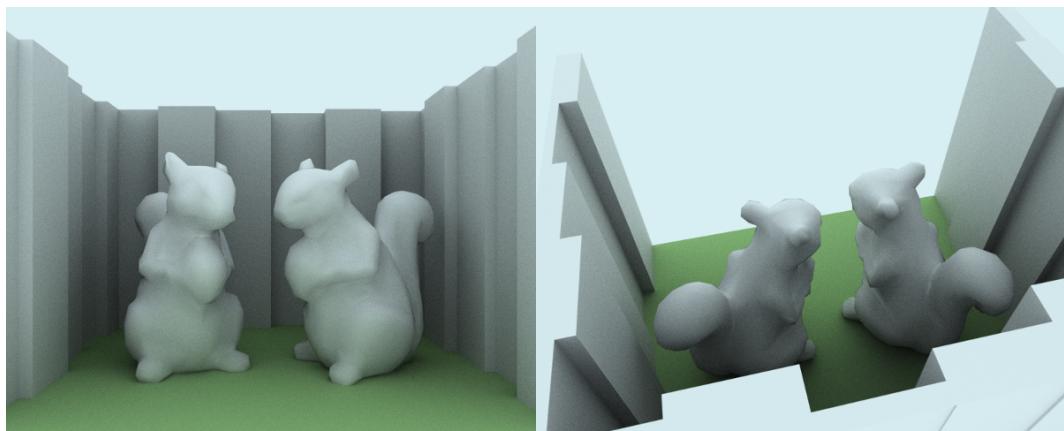


Abbildung 9.8: Die Szene für die diversen Performance-Tests. Verwendet wird dabei die Ansicht des linken Bildes.

Pfadlängen und Materialien

In Abbildung 9.9 wurde die Szene mit unterschiedlichen maximalen Pfadlängen und Materialeigenschaften für die Eichhörnchenmodelle gerendert. Sie beinhaltet insgesamt 1148 Vertices und 2198 Faces. Wie in der Abbildung zu sehen ist ein wichtiger Faktor für die Helligkeit einer Szene die maximale Pfadlänge. Eine Pfadlänge von nur 2 Strahlen bedeutet, dass ein Pfad nur aus einem Strahl zu einem Punkt in der Szene bestehen kann und einem Folgestrahl zu einer Lichtquelle. Ansonsten trägt der Pfad keine Farbinformationen zum Bild bei. Daraus folgt ebenfalls, dass kein Color Bleeding möglich ist, da hierfür mindestens eine weitere Fläche getroffen werden müsste. Wie die Messungen in Tabelle 9.1 zeigen, nimmt die Ausführungsgeschwindigkeit mit zunehmender Pfadlänge deutlich ab.

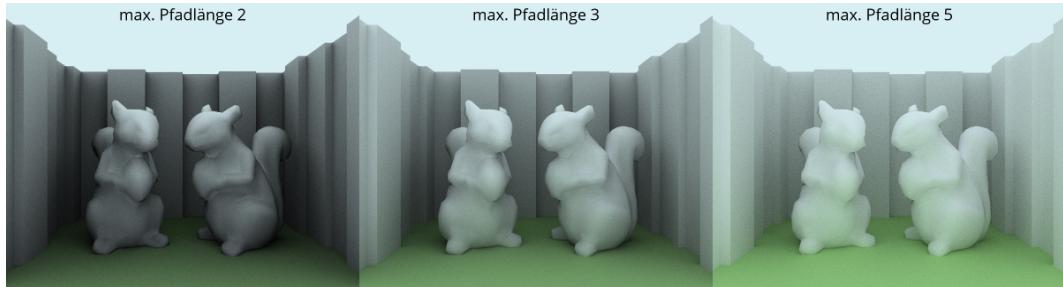


Abbildung 9.9: Die Eichhörnchen haben eine diffuse Oberfläche, maximale Pfadlänge variiert.

Rauheit	Pfadlänge	Zeit
$n_u = n_v = 1$	2	49 ms
$n_u = n_v = 1$	3	88 ms
$n_u = n_v = 1$	5	141 ms

Tabelle 9.1: Vergleich für verschiedene Pfadlängen bei Shirley-Ashikhmin-BRDF.

Tabelle 9.2 zeigt die Auswertung für verschiedene matte bzw. spekulare Materialien und den Einfluss der maximalen Pfadverlängerung (angegeben in Klammern hinter der maximalen Pfadlänge). Eine Pfadverlängerung findet statt, wenn ein Strahl auf eine spekulare oder transparente Fläche trifft. Deaktiviert man Pfadverlängerungen, zeigen Spiegelungen wie im Bild rechts in Abbildung 9.10 vermehrt dunkle Stellen, wenn es sich bei der Reflektion nicht um eine Lichtquelle handelt. Wie die Messungen zeigen, ist es nicht weiter von Bedeutung, wie spekular eine Fläche ist. Der entscheidende Punkt ist die maximale Pfadverlängerung. Deaktiviert man sie, sind die Zeiten nahezu gleich. Der Unterschied von 8 ms ist hier hauptsächlich darauf zurückzuführen, dass in dieser Szene mit der nach vorne offenen Box es wahrscheinlicher ist, dass die umgebende Lichtquelle gespiegelt wird. Sobald der Pfad verlängert wird, erhöhen sich auch die gemessenen Zeiten. Eine zusätzliche Verlängerung von mehr als 3 Strahlen wirkt sich jedoch kaum noch aus. Am Russisch Roulette kann es jedoch nicht liegen, da dies vorwiegend in schlecht beleuchteten Szenen Vorteile bringt.

Ein weiterer Fall, in dem Pfadverlängerungen von Bedeutung sind, sind transparente Materialien, die Licht brechen. Hier werden Pfade i.d.R. öfter verlängert, als bei spiegelnden Flächen. Denn der Strahl wird einmal verlängert, wenn er in das durchsichtige Objekt eintritt, und erneut, wenn er aus ihm austritt. Eine weitere Möglichkeit ist, dass der Pfad länger als eine Strahlenlänge im Objekt bleibt, wenn er aufgrund von totaler interner Reflektion zurück ins Innere gespiegelt wird. Abbildung 9.11 zeigt die Auswirkungen von Pfadverlängerungen auf gläserne Eichhörnchenmodelle. Mit nur einer Verlängerung erscheinen sie sehr unnatürlich dun-

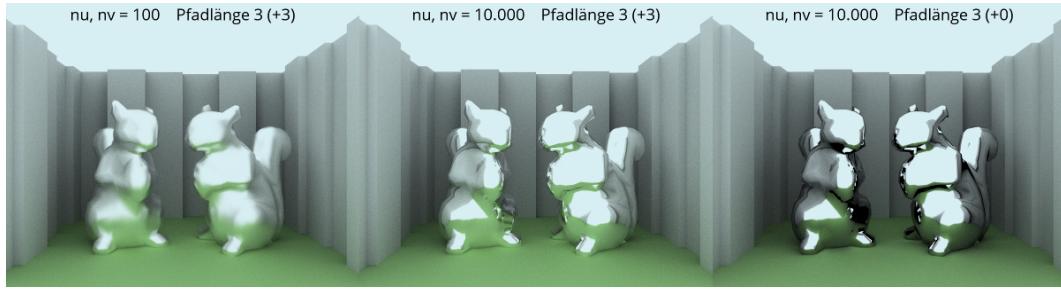


Abbildung 9.10: Die Eichhörnchen haben matte bzw. spekulare Oberflächen, maximale Pfadverlängerung variiert.

Rauheit	Pfadlänge	R. Roulette	Zeit
$n_u = n_v = 100$	3 (+0)	ja	81 ms
$n_u = n_v = 100$	3 (+3)	ja	118 ms
$n_u = n_v = 100$	3 (+5)	ja	119 ms
$n_u = n_v = 1000$	3 (+0)	ja	78 ms
$n_u = n_v = 1000$	3 (+3)	ja	114 ms
$n_u = n_v = 1000$	3 (+5)	ja	115 ms
$n_u = n_v = 10.000$	3 (+0)	ja	76 ms
$n_u = n_v = 10.000$	3 (+3)	ja	113 ms
$n_u = n_v = 10.000$	3 (+5)	ja	114 ms
$n_u = n_v = 10.000$	3 (+3)	nein	110 ms
$n_u = n_v = 10.000$	3 (+7)	nein	112 ms

Tabelle 9.2: Vergleich für matte und spekulare Oberflächen bei variierender maximalen Pfadverlängerung und Shirley-Ashikhmin-BRDF. Der Pluswert in Klammern gibt die max. Pfadverlängerung an.

kel. Ein Pfad kann so maximal eine Länge von 4 Strahlen erreichen. Die ersten drei werden jedoch bereits verbraucht, wenn sie erst eintreten, wieder austreten und dann die Wand dahinter treffen. Somit bleibt nur noch ein Strahl übrig, eine Lichtquelle zu erreichen. Mit mehr Verlängerungen steigt auch die Chance noch eine Lichtquelle zu finden, weshalb die gläsernen Eichhörnchen in den anderen beiden Bildern heller sind. Die notwendigen Verlängerungen der Pfade wirken sich auch negativ auf die Ausführungs geschwindigkeit aus, wie Tabelle 9.3 belegt.

Brechungsindex	Pfadlänge	Zeit
$Ni = 1,5$	3 (+1)	107 ms
$Ni = 1,5$	3 (+3)	164 ms
$Ni = 1,5$	3 (+5)	201 ms

Tabelle 9.3: Transparente Materialien bei variierender maximalen Pfadverlängerung und Shirley-Ashikhmin-BRDF. Der Pluswert in Klammern gibt die max. Pfadverlängerung an.

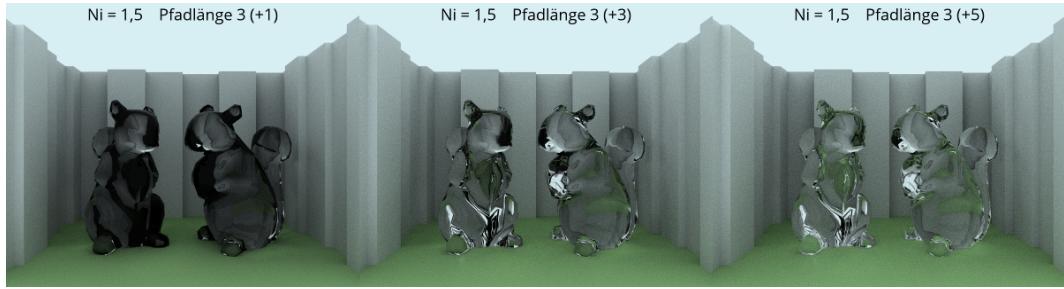


Abbildung 9.11: Eichhörnchen aus Glas, maximale Pfadverlängerung variiert.

Schließlich sollen auch noch die beiden implementierten BRDFs verglichen werden. Für einen fairen Vergleich müssen die Parameter der Materialien jeweils so gewählt sein, dass die zu vergleichenden Bilder möglichst identisch sind. Die Messwerte aus Tabelle 9.4 zeigen, dass die Wahl der BRDF keine nennenswerte Auswirkung auf die Geschwindigkeit hat.

Pfadlänge	Schlick		Shirley-Ashikhmin	
	Rauheit	Zeit	Rauheit	Zeit
3 (+3)	$r = 1$	85 ms	$n_u = n_v = 1$	88 ms
5 (+3)	$r = 1$	135 ms	$n_u = n_v = 1$	141 ms
3 (+3)	$r = 0$	116 ms	$n_u = n_v = 10.000$	113 ms
5 (+3)	$r = 0$	158 ms	$n_u = n_v = 10.000$	160 ms

Tabelle 9.4: Vergleich für verschiedene BRDFs. Der Pluswert in Klammern gibt die max. Pfadverlängerung an.

Workgroup-Größen

In Kapitel 8 wurden Workgroups erklärt und bereits erwähnt, dass sich eine Größe von 8×8 bewährt hat. Hier folgen mit Tabelle 9.5 die Messwerte, die dies belegen. Der nächst-kleinere Wert mit 4 war wieder deutlich schlechter. Der Unterschied zum nächst-größeren Wert 16 ist hingegen nur minimal. Größere Workgroups ließen sich nicht testen, da in diesem Fall OpenCL einen Fehler CL_OUT_OF_RESOURCES warf. Schließlich existiert auch noch die Möglichkeit, OpenCL selbst die Workgroup-Größe bestimmen zu lassen. Dies erweist sich jedoch als suboptimal, da die Ausführungszeit deutlich ansteigt.

Erstellen der BVH

Die Wahl einer geeigneten Beschleunigungsstruktur ist von großer Bedeutung für die Ausführungsgeschwindigkeit eines Renderers. Während an dieser Stelle leider

Rauheit	Pfadlänge	Workgroup-Größe	Zeit
$n_u = n_v = 1$	3	4	172 ms
$n_u = n_v = 1$	3	8	88 ms
$n_u = n_v = 1$	3	16	89 ms
$n_u = n_v = 1$	3	auto	121 ms

Tabelle 9.5: Auswirkung der Workgroup-Größe mit Shirley-Ashikhmin-BRDF.

keine Strukturen verglichen werden können, sollen zumindest die Auswirkungen einiger Parameter der BVH betrachtet werden: Die maximale Anzahl an Faces per Blattknoten und ob eine Surface Area Heuristic verwendet wurde. Tabelle 9.6 listet die Resultate. Zum einen zeigt sich, dass von 1 zu 4 Faces eine kontinuierliche – wenn auch geringfügige – Verbesserung auftritt. Die SAH trägt jedoch deutlich zur Verringerung der Ausführungszeit bei. Ein Nachteil ist jedoch, dass die Zeit zur Erstellung des Baumes stark ansteigt, was gerade für größere Modelle eine Wartezeit von einigen Minuten bedeuten kann. An dieser Stelle sollte noch nachgebessert werden, auch wenn es sich nicht auf die Renderzeit selbst auswirkt. Eine Möglichkeit wäre, zu versuchen, den Vorgang auf mehrere CPU-Threads zu verteilen, oder sogar auf die Grafikkarte zu verschieben.

Max. Faces	mit SAH			ohne SAH		
	Erstell-Zeit	B.-Tiefe	Zeit	Erstell-Zeit	B.-Tiefe	Zeit
1	3707 ms	17	104 ms	505 ms	19	131 ms
2	3667 ms	16	90 ms	260 ms	18	123 ms
3	3652 ms	15	89 ms	158 ms	17	116 ms
4	3638 ms	15	88 ms	129 ms	16	115 ms

Tabelle 9.6: Auswirkungen des BVH-Aufbaus auf Erstell-Zeit des Baumes, die erreichte Baum-Tiefe und Kernel-Ausführungszeit für die Test-Szene. Verwendet wurde die Shirley-Ashikhmin-BRDF und eine maximale Pfadlänge von 3. Die Szene ist durchgängig diffus.

10 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit entstand ein auf der GPU ausgeführter Online-Path Tracer, der Licht und Farben auf eine Physik-basierte Art repräsentiert. Anstatt den sonst üblichen RGB-Werten werden zur Repräsentation Spectral Power Distributions (SPD) verwendet, die Licht bzw. die Farbreflektion von Materialien als Energie je Wellenlänge beschreibt. Dabei werden 40 Wellenlängen in 10 nm-Schritten gesampelt, beginnend bei 380 nm. Zum schnellen Testen der Geometrie einer Szene mit einem Strahl wird als Beschleunigungsstruktur eine Bounding Volume Hierarchy eingesetzt. Dabei wird die Effizienz des zu erzeugenden Baumes durch den Einsatz einer Surface Area Heuristic gesteigert.

Zwei verschiedene Bidirectional Rendering Distribution Functions (BRDF) wurden implementiert: Die von Schlick und jene von Shirley und Ashikhmin. Beide unterstützen wichtige, grundlegende Effekte wie diffuse und spekulare Oberflächen, matte Abstufungen zwischen diffus und spekular, anisotropische Flächen und Lichtbrechung in durchsichtigen Materialien wie Glas. Natürliche Effekte wie Lichtbündelungen durch Lichtbrechung sind dabei ebenfalls zu beobachten. Da die Energie einer Lichtquelle auf dem Pfad zum Auge durch die SPDs getroffener Flächen verändert wird, tritt auch Color Bleeding auf – eine beleuchtete rote Fläche strahlt z.B. auf eine in der Nähe befindliche weiße Wand ab, wodurch diese ebenfalls leicht rötlich erscheint. Zur Glättung von Treppenbildungen im Bild wurde zudem ein Anti-Aliasing-Verfahren implementiert, das darauf basiert, die initialen Strahlen in die Szene leicht zu verwackeln.

Die Evaluation wurde mit einer NVIDIA GTX 560 Ti durchgeführt. Dabei hat sich gezeigt, dass interaktives, Physik-basiertes Rendering auch heute schon möglich ist und zwar nicht nur auf Highend-Hardware. Jedoch ist der mögliche Umfang der Auflösung und Szene noch recht begrenzt. Bei Szenen mit mehr als 2000 Faces und gerendert in einer Auflösung von 800×640 Pixeln stößt die Anwendung mit Kernel-Ausführungszeiten von 100 ms aufwärts (entsprechend 10 FPS und weniger) an ihre Grenzen.

Doch sowohl für die Render-Geschwindigkeit als auch die Bildqualität ist das Potential noch lange nicht ausgeschöpft. Zukünftige Arbeiten werden Alternativen oder Verbesserungen für den BVH-Baum untersuchen, wie die Verwendung von Spatial Splits [SFD09] während der Erstellung. Auch sollte geprüft werden, ob eine BVH aus Bounding Spheres schneller durchlaufen werden kann als die aktuelle, die Bounding Boxes verwendet. Zielsetzung für weitere Arbeiten ist, dass aus dem Gebiet des Rendering bekannte Szenen wie „Sponza“ (ca. 66.000 Faces) oder „Sibenik“ (ca. 75.000 Faces) (für beide siehe Abbildung 10.1) gerendert werden können.



Abbildung 10.1: Links: Sponza. Rechts: Sibenik. Beide Szenen wurden mit dem Gradient-Domain Metropolis Light Transport-Verfahren gerendert. Aus [LKL⁺13].

Zur Verbesserung der Bildqualität bieten sich modernere, auf Path Tracing aufbauende Techniken an. Namentlich sind das Metropolis Light Transport und Energy Redistribution Path Tracing. Daneben existieren noch Verfahren zur Rauschunterdrückung in generierten Bildern, wie beispielsweise in [SD12] ausgeführt wird.

Für gesteigerten Realismus in den gerenderten Bildern ist es wichtig, möglichst viele natürliche Effekte zu berücksichtigen. Neben den bereits umgesetzten wären mögliche Erweiterungen noch Tiefunschärfe, polarisiertes Licht, Lichtbrechung in Participating Media (z.B. Nebel oder Rauch) und die Auswirkung von Temperatur (z.B. Hitzeblitzen). Auch kann die Auswahl an BRDFs noch erweitert werden, z.B. um Cook-Torrance [CT82], Oren-Nayar [ON94] oder GGX [WMLT07].

Literaturverzeichnis

- [App68] APPEL, Arthur: Some Techniques for Shading Machine Renderings of Solids. In: *Proceedings of the April 30–May 2, 1968, spring joint computer conference ACM*, 1968, S. 37–45
- [AS00] ASHIKHMİN, Michael ; SHIRLEY, Peter: An Anisotropic Phong BRDF Model. In: *Journal of graphics tools* 5 (2000), Nr. 2, S. 25–32
- [CE05] CLINE, David ; EGBERT, Parris: A Practical Introduction to Metropolis Light Transport / Tech. rep., Brigham Young University. 2005. – Forschungsbericht
- [CNS⁺11] CRASSIN, Cyril ; NEYRET, Fabrice ; SAINZ, Miguel ; GREEN, Simon ; EISEMANN, Elmar: Interactive indirect illumination using voxel cone tracing. In: *Computer Graphics Forum* Bd. 30 Wiley Online Library, 2011, S. 1921–1930
- [CT82] COOK, Robert L. ; TORRANCE, Kenneth E.: A Reflectance Model for Computer Graphics. In: *ACM Transactions on Graphics (TOG)* 1 (1982), Nr. 1, S. 7–24
- [CTE05] CLINE, David ; TALBOT, Justin ; EGBERT, Parris: Energy redistribution path tracing. In: *ACM Transactions on Graphics (TOG)* Bd. 24 ACM, 2005, S. 1186–1195
- [DCWP02] DEVLIN, Kate ; CHALMERS, Alan ; WILKIE, Alexander ; PURGATHOFER, Werner: Tone Reproduction and Physically Based Spectral Rendering. In: *Eurographics 2002: State of the Art Reports* (2002), S. 101–123
- [GS04] GUY, Stephane ; SOLER, Cyril: Graphics Gems Revisited: Fast and Physically-based Rendering of Gemstones. In: *ACM Transactions on Graphics (TOG)* Bd. 23 ACM, 2004, S. 231–238
- [HDW⁺11] HAPALA, Michal ; DAVIDOVIČ, Tomáš ; WALD, Ingo ; HAVRAN, Vlastimil ; SLUSALLEK, Philipp: Efficient stack-less bvh traversal for ray tracing. In: *Proceedings of the 27th Spring Conference on Computer Graphics ACM*, 2011, S. 7–12
- [Hec90] HECKBERT, Paul S.: Adaptive Radiosity Textures for Bidirectional Ray Tracing. In: *ACM SIGGRAPH Computer Graphics* Bd. 24 ACM, 1990, S. 145–154
- [HGNR⁺12] HEISENBERG (GOEBBELS), Gernot ; NATARAJAN, Ramesh K. ; REZAEI, Yashar A. ; SIMON, Nicolas ; HEIDEN, Wolfgang: Robust EEG time series transient detection with a momentary frequency estimator

- for the indication of an emotional change. In: *6th Workshop of Emotion and Computing at the 35th German Conference on Artificial Intelligence (KI 2012)*, 2012
- [Jen01] JENSEN, Henrik W.: *Realistic Image Synthesis Using Photon Mapping*. Natick, MA, USA : A. K. Peters, Ltd., 2001. – ISBN 1-56881-147-0
 - [Kaj86] KAJIYA, James T.: The Rendering Equation. In: *ACM Siggraph Computer Graphics* Bd. 20 ACM, 1986, S. 143–150
 - [Laf96] LAFORTUNE, Eric: *Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering*, Citeseer, Diss., 1996
 - [LKL⁺13] LEHTINEN, Jaakko ; KARRAS, Tero ; LAINE, Samuli ; AITTALA, Miika ; DURAND, Frédo ; AILA, Timo: Gradient-domain Metropolis Light Transport. In: *ACM Trans. Graph.* 32 (2013), Juli, Nr. 4, 95:1–95:12. <http://dx.doi.org/10.1145/2461912.2461943>. – DOI 10.1145/2461912.2461943. – ISSN 0730-0301
 - [LW93] LAFORTUNE, Eric P. ; WILLEMS, Yves D.: Bi-directional path tracing. In: *Proceedings of CompuGraphics* Bd. 93, 1993, S. 145–153
 - [Mit12] MITTRING, Martin: The Technology Behind the “Unreal Engine 4 Elemental demo”. In: *Part of “Advances in Real-Time Rendering in 3D Graphics and Games,” SIGGRAPH* (2012)
 - [MT05] MÖLLER, Tomas ; TRUMBORE, Ben: Fast, minimum storage ray/triangle intersection. In: *ACM SIGGRAPH 2005 Courses* ACM, 2005, S. 7
 - [ON94] OREN, Michael ; NAYAR, Shree K.: Generalization of Lambert’s Reflectance Model. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* ACM, 1994, S. 239–246
 - [PF90] POULIN, Pierre ; FOURNIER, Alain: A Model for Anisotropic Reflection. In: *ACM SIGGRAPH Computer Graphics* Bd. 24 ACM, 1990, S. 273–282
 - [RBA09] RADZISZEWSKI, Michal ; BORYCZKO, Krzysztof ; ALDA, Witold: An Improved Technique for Full Spectral Rendering. In: *Journal of WSCG* 17 (2009), Nr. 1–3, S. 9–16
 - [RPJV93] RUSHMEIER, Holly E. ; PATTERSON JR, Charles W. ; VEERASAMY, Aravindan: Geometric Simplification for Indirect Illumination Calculation. In: *Proceedings of Graphics Interface ’93* (1993), Mai, S. 227–236
 - [SC97] SHIRLEY, Peter ; CHIU, Kenneth: A Low Distortion Map Between Disk and Square. In: *Journal of graphics tools* 2 (1997), Nr. 3, S. 45–52

- [Sch94] SCHLICK, Christophe: An Inexpensive BRDF Model for Physically-based Rendering. In: *Computer graphics forum* Bd. 13 Wiley Online Library, 1994, S. 233–246
- [SD12] SEN, Pradeep ; DARABI, Soheil: On Filtering the Noise from the Random Parameters in Monte Carlo Rendering. In: *ACM Trans. Graph.* 31 (2012), Nr. 3, S. 18
- [SFD09] STICH, Martin ; FRIEDRICH, Heiko ; DIETRICH, Andreas: Spatial Splits in Bounding Volume Hierarchies. In: *Proceedings of the Conference on High Performance Graphics 2009* ACM, 2009, S. 7–13
- [VG97] VEAUCH, Eric ; GUIBAS, Leonidas J.: Metropolis Light Transport. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* ACM Press/Addison-Wesley Publishing Co., 1997, S. 65–76
- [WBMS05] WILLIAMS, Amy ; BARRUS, Steve ; MORLEY, R K. ; SHIRLEY, Peter: An efficient and robust ray-box intersection algorithm. In: *ACM SIGGRAPH 2005 Courses* ACM, 2005, S. 9
- [Whi05] WHITTED, Turner: An improved illumination model for shaded display. In: *ACM SIGGRAPH 2005 Courses* ACM, 2005, S. 4
- [WMLT07] WALTER, Bruce ; MARSCHNER, Stephen R. ; LI, Hongsong ; TORRANCE, Kenneth E.: Microfacet Models for Refraction through Rough Surfaces. In: *Proceedings of the 18th Eurographics conference on Rendering Techniques* Eurographics Association, 2007, S. 195–206
- [WSBW01] WALD, Ingo ; SLUSALLEK, Philipp ; BENTHIN, Carsten ; WAGNER, Markus: Interactive Rendering with Coherent Ray Tracing. In: *Computer graphics forum* Bd. 20 Wiley Online Library, 2001, S. 153–165

Online-Quellen

- [Bal12] BALDWIN, Andrew: *ThNdl: Noise from numbers*. <http://thndl.com/?14>, November 2012
- [Bat05] BATTY, Christopher: *Implementing Energy Redistribution Path Tracing*. <https://www.cs.ubc.ca/~batty/projects/ERPT-report.pdf>, 2005. – [letzter Zugriff: 2014-03-02]
- [Bor03] BORMAN, Sean: *Raytracing and the camera matrix—a connection*. <http://www.seanborman.com/publications/raycam.pdf>, June 2003. – [letzter Zugriff: 2013-12-30]

-
- [DG06] DE GREVE, Bram: *Reflections and Refractions in Ray Tracing*.
http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf, November 2006
 - [Int07] INTERNATIONAL COMISSION ON ILLUMINATION: *Selected Colorimetric Tables*. http://www.cie.co.at/index.php/LEFTMENU/index.php?i_ca_id=298, 2007. – [letzter Zugriff: 2014-03-02]
 - [Lin11] LINDBLOOM, Bruce J.: *RGB/XYZ Matrices*. http://www.brucelindbloom.com/index.html?Eqn_XYZ_to_RGB.html, Januar 2011. – [letzter Zugriff: 2014-02-21]
 - [NVI09] NVIDIA CORPORATION: *NVIDIA OpenCL Best Practices Guide*. http://www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf, August 2009. – [letzter Zugriff: 2014-03-03]
 - [onl10] *Ray-Box Intersection*. <http://scratchapixel.com/lessons/3d-basic-lessons/lesson-7-intersecting-simple-shapes/ray-box-intersection/>, 2010. – [letzter Zugriff: 2014-01-04]
 - [Wal96] WALKER, John: *Colour Rendering of Spectra*. <http://www.fourmilab.ch/documents/specrend/>, April 1996. – [letzter Zugriff: 2013-12-30]
 - [Wat12] WATERS, Zack: *Realistic Raytracing*. http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/realistic_raytracing.html, 2012. – [letzter Zugriff: 2014-04-15]