

BACHELORARBEIT

Analyse und Implementierung von Shader Techniken zur Lichtberechnung in Echtzeit-Render-Engines

vorgelegt am 31. August 2023
Florian Bökenberg
Matrikelnummer: XXXXXXXXXX

1. Prüfer: Prof. Dr. Roland Greule
2. Prüfer: Mike Bartscher, M. Sc.

**HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG**
Fachbereich Medientechnik
Finkenau 35
22081 Hamburg

Zusammenfassung

In Echtzeit-Render-Engines werden komplexe Lichteffekte in Echtzeit auf Bildschirmen erzeugt. Die Arbeit untersucht unterschiedliche Techniken und Algorithmen, die in diesen Engines verwendet werden, um realistische Beleuchtung zu simulieren.

Ein Schwerpunkt liegt auf der Berechnung von Schatten, Wechselwirkung von Licht und Materie sowie indirekter und direkter Beleuchtung. Dazu werden verschiedene Ansätze und Techniken analysiert und verglichen.

Die Arbeit diskutiert auch die Herausforderungen bei der Implementierung dieser Techniken in Render-Engines, wie zum Beispiel die Begrenzung durch die Rechenleistung und Speicherressourcen.

Abschließend werden mögliche zukünftige Entwicklungen und Forschungsrichtungen in diesem Bereich vorgestellt, um die Qualität und Leistung der Lichtberechnung in Echtzeit-Render-Engines weiter zu verbessern.

Abstract

In real-time rendering engines, complex lighting effects are generated in real-time on screens. The thesis examines various techniques and algorithms used in these engines to simulate realistic lighting.

One focus is on the calculation of shadows, the interaction of light and matter, as well as indirect and direct illumination. Different approaches and techniques are analyzed and compared.

The thesis also discusses the challenges of implementing these techniques in real-time rendering engines, such as limitations imposed by computational power and memory resources.

Finally, potential future developments and research directions in this field are presented to further enhance the quality and performance of real-time lighting computation in real-time rendering engines.

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Abkürzungsverzeichnis	VI
1 Einleitung	1
1.1 Problemstellung	1
1.2 Zentrale Forschungsfragen	2
1.3 Aufbau der Arbeit	2
2 Grundlagen	3
2.1 Shader	3
2.2 Hardware	4
2.3 Definition Echtzeit	4
2.4 Abgrenzung Offline-Render-Engine	5
2.5 Physikalische Grundlagen	6
2.6 Rendergleichung	8
3 Rendertechniken	9
3.1 Physically based shading	9
3.2 Global Illumination	12
3.2.1 Direkte Beleuchtung	13
3.2.2 Indirekte Beleuchtung	14
3.3 Rasterisierung	14
3.4 Raytracing	15
3.5 Path Tracing	16
4 Analyse von Rendertechniken in Echtzeit-Render-Engine	17
4.1 Vorstellung von Render-Engines	17
4.1.1 Blender	17
4.1.2 Unreal Engine 5	18
4.2 Rendertechniken in EEVEE	18
4.2.1 Rasterisierung	18
4.2.2 Global Illumination	18
4.2.3 Schatten	20
4.3 Unreal Engine 5 Rendertechniken	21
4.3.1 "Lumen"	21
4.3.1.1 Raytracing in "Lumen"	21
4.3.1.2 Surface Cache	22
4.3.1.3 Global Illumination in "Lumen"	23
4.3.2 Schatten	23
5 Methodik	24
5.1 Messung der Leistung von Echtzeit-Render-Engines	24
5.1.1 Messung FPS	25
5.1.2 Messung Latenz	25
5.1.3 Messung VRAM	26
5.2 Visuelle Qualität	27
5.2.1 Vergleich zu Offline-Render-Engine	27

5.2.2	Evaluierung im Anwendungsfenster	27
5.3	Testszenarien	28
5.3.1	Testszenario 1	28
5.3.2	Testszenario 2	29
5.4	Vergleichbarkeit von Messgrößen	29
5.4.1	Export	30
5.4.2	Lumen oder Watt	30
5.4.2.1	Lösungsansatz 1 Einsatz von IES Lights	31
5.4.2.2	Lösungsansatz 2 Mathematische Umrechnung realer Werte	33
5.4.2.3	Lösungsansatz 3 Vergleichsmethode	34
6	Evaluierung visuelle Qualität	35
6.1	Reflexionen	37
6.1.1	Diffuse Reflexionen	37
6.1.2	Spiegelnde Reflexionen	37
6.1.3	Gestreute Reflexionen	38
6.1.4	Brechung und Reflexion	39
6.1.5	Brechung, Absorption, Reflexion	40
6.2	Schatten	41
6.3	Global Illumination	42
7	Evaluierung Performance	43
8	Fazit	46
9	Ausblick	47
9.1	Zukunft von Echtzeit-Render-Engines	47
9.1.1	Spektrales Rendern	47
9.2	Künstliche Intelligenz	47
10	Literaturverzeichnis	49
11	Anhang	52
12	Eigenständigkeitserklärung	53

Abbildungsverzeichnis

Abbildung 2.1: Beispiel einer Rendering Pipeline	3
Abbildung 2.2: Verhältnis von radiometrischen und photometrischen Größen	7
Abbildung 3.1: Veranschaulichung diffuser (links), gestreuter (mitte) und spiegelnder Reflexion (rechts)	10
Abbildung 3.2: Alle Materialeigenschaften von Disneys BRDF	11
Abbildung 3.3: Das Licht gelangt an P in das Material von der Lichtquelle, die durch ω_i angezeigt wird, es bewegt sich entlang des roten Pfades, prallt innerhalb des Materials ab und tritt an Q in Richtung ω_o aus.	12
Abbildung 3.4: Darstellung der Betrachtung eines Punktes im Bezug auf die indirekten und direkten Beleuchtung	13
Abbildung 3.5: Beispiel eines HDRI (rechts), das als Umgebungsbeleuchtung in einer Kugel verwendet wird (links)	14
Abbildung 3.6: Die Beleuchtung an Position p wird von dem Punkt r indirekt durch die Beleuchtungsrichtung l mithilfe der einfallenden Strahldichte $L_i(p, l)$ bzw. ausgehenden Strahldichte $L_o(r(p, l), -l)$ beschrieben	14
Abbildung 3.7: Darstellung eines Dreiecks mit den Punkten p_0, p_1, p_2 , das in einem 16×8 Pixelraster dargestellt wird. Die grünen Pixel stellen das Rechteck dar und die gelben die Anti Aliasing Pixel	15
Abbildung 3.8: Darstellung von verschiedenen Strahlen durch ein Raster zur Beschreibung von Raytracing	16
Abbildung 3.9: Vergleichsbild zwischen einem realen Foto (links) und eines mit Path Tracing gerenderten Bildes (rechts)	17
Abbildung 4.1: Beispielhafte Berechnung von Ambient Occlusion an p_0, p_1, p_2	19
Abbildung 4.2: Darstellung von Lichtproben in EEVEE	19
Abbildung 4.3: Darstellung der Erstellung einer Shadow Map aus Sicht der Lichtquelle (links) und der Berechnung des Bildes beim Rendern (rechts)	20
Abbildung 4.4: Darstellung von Ray Marching in einer zweidimensionalen Umgebung	22
Abbildung 4.5: "Lumen Scene": Vor Auslesen des Surface Cache (links) und nach Auslesen des Surface Cache (rechts)	23
Abbildung 4.6: Szene zur Darstellung von Schatten in Unreal Engine 5	24
Abbildung 5.1: Beschreibung des Aufbaus von TestszENARIO 1 (Cornell Box)	28

Abbildung 5.2: Bilder des Aufbaus von TestszENARIO 2	29
Abbildung 5.3: Darstellung der TestszENARIEN: FBX (links), USD (mitte), Datasmith (rechts)	30
Abbildung 5.4: photometrische Strahlungsäquivalenzkurve	31
Abbildung 5.5: IES Licht in Blender (links) und in Unreal Engine 5 (rechts)	32
Abbildung 5.6: Lichtkompensation in Unreal Engine an (links), aus (rechts)	35
Abbildung 6.1: TestszENARIO 1 in Cycles	35
Abbildung 6.2: TestszENARIO 1 in EEVEE	36
Abbildung 6.3: TestszENARIO 1 in Unreal Engine 5	36
Abbildung 6.4: Darstellung diffuser Würfel in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)	37
Abbildung 6.5: Vergrößerte Darstellung spiegelnder Reflexionen in Cycles (links oben), Eevee (rechts oben), Unreal Engine 5 (unten)	38
Abbildung 6.6: Vergrößerte Darstellung gestreuter Reflexionen in Cycles (links), Eevee (rechts), Unreal Engine 5 (unten)	38
Abbildung 6.7: Vergrößerte Darstellung der Glaskugel in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)	39
Abbildung 6.8: Vergrößerte Darstellung des Glaszylinders in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)	39
Abbildung 6.9: Vergrößerte Darstellung vom Subsurface Scattering Affenkopf in Cycles (links), Eevee (rechts), Unreal Engine 5 (unten)	40
Abbildung 6.10: Vergrößerte Darstellung von Schatteneffekten in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)	41
Abbildung 6.11: Vergrößerte Darstellung von Bildrauschen in Unreal Engine 5 (links), Cycles (rechts)	41
Abbildung 6.12: Vergrößerte Darstellung von Kaustiks in Cycles (links), Unreal Engine 5 (rechts)	41
Abbildung 6.13: Vergrößerte Darstellung von Global Illumination in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)	42
Abbildung 7.1: Cornell Box als Referenzbild bezüglich der Materialeigenschaften	43
Abbildung 7.2: Alle reflektierenden Objekte mit ihren Reflexionseigenschaften in TestszENARIO 2	44
Abbildung 7.3: GPU Visualizer zeigt die zeitliche Zusammensetzung eines Bildes in Unreal Engine 5 an	45

Abkürzungsverzeichnis

BRDF	Bidirectional Reflectance Distribution Function.
BSDF	Bidirectional Scattering Distribution Function.
BSSRDF	Bidirectional Scattering Surface Reflectance Distribution Function.
BTDF	Bidirectional Transmittance Distribution Function.
EEVEE	Extra Easy Virtual Environment Engine.
FPS	Bilder pro Sekunde
PBS	Physically Based Shading
SSR	Screen-Space-Reflections
SSS	Subsurface Scattering
VRAM	Video Random Access Memory.

1 Einleitung

Die Berechnung von Licht in Echtzeit-Render-Engines spielt eine entscheidende Rolle für die visuelle Qualität und den Realismus von 3D-Szenen. Seit den 80er Jahren haben sich Konzepte entwickelt, die das Ziel hatten, physikalische akkurate Berechnungen mithilfe von Shadern durchzuführen [1, S. 127]. Eines der wesentlichsten Konzepte zum Erzielen fotorealistischer Bilder war der Einsatz von Raytracing (vollständige Erklärung des Begriffs in Kapitel 3.4). Doch die Berechnung einfacher Bilder konnte oft mehrere Stunden in Anspruch nehmen und erforderte eine erhebliche Rechenleistung. Jedoch haben sich in den letzten Jahren sowohl die Algorithmen als auch die Hardware erheblich weiterentwickelt, was zu einer Verbesserung der Leistung und Qualität von Raytracing geführt hat. Diese Fortschritte haben es ermöglicht, Raytracing in Echtzeit-Render-Engines einzusetzen, wodurch komplexe Lichteffekte in Echtzeit auf Bildschirmen oder Displays erzeugt werden können. Das eröffnet neue Möglichkeiten für die Erzeugung hochwertiger visueller Inhalte in Echtzeit.

1.1 Problemstellung

Vor diesem Hintergrund widmet sich die Bachelorarbeit der Untersuchung und Analyse der Lichtberechnung in Echtzeit-Render-Engines und untersucht verschiedene Techniken, Algorithmen und Herausforderungen, um sie auf Qualität und Leistung zu überprüfen.

Es werden jedoch nicht alle möglichen Variationen von Techniken betrachtet, sondern ausschließlich diejenigen, die in den 3D-Programmen Blender (Version 3.6.1) und Unreal Engine 5.2, zum Einsatz kommen.

Hierbei liegt der Fokus auf den Techniken, die auf die Lichtberechnung direkten Einfluss haben. Auf Techniken wie Denoising¹ oder Anti-Aliasing² zur Verbesserung der Performance und visuellen Qualität wird nur am Rande eingegangen. Die Arbeit beschränkt sich auf relevante Inhalte im Kontext der genannten Themen und behandelt keine weiteren Aspekte zum Thema Fotorealismus, wie Animationen, Simulationen oder Postprocessing.

¹ Denoising ist ein Prozess, bei dem Rauschen oder Unregelmäßigkeiten aus einem Bild entfernt werden, um ein glattes und sauberes Endergebnis zu erzielen.

² Anti-Aliasing ist eine Technik, die verwendet wird, um das Flimmern oder Treppeneffekte (Aliasing) in gerenderten Bildern zu reduzieren.

1.2 Zentrale Forschungsfragen

Zentrale Forschungsfragen sind hierbei:

- Welche Techniken zur Lichtberechnung werden in Echtzeit-Render-Engines von EEVEE³ und Unreal Engine 5 eingesetzt?
- Wie wirken sich diese Techniken auf die visuelle Qualität und Performance der Echtzeit-Render-Engines aus?

1.3 Aufbau der Arbeit

In Kapitel 2 werden grundlegende Themen behandelt, die den Rahmen für die Lichtberechnung in Echtzeit bilden und einen Einfluss auf die Erzeugung von realistischen Bildern haben. Es werden relevante Konzepte und Technologien erläutert, die für das Verständnis und die Umsetzung in Echtzeit-Render-Engines unerlässlich sind.

Kapitel 3 führt verschiedene allgemeine Lösungsansätze ein, die zur Erzeugung realistischer Lichtberechnungen in Echtzeit eingesetzt werden können. Es werden Techniken und Algorithmen vorgestellt, die es ermöglichen, Beleuchtung, Schatten und Materialeigenschaften zu simulieren.

In Kapitel 4 erfolgt eine detaillierte Analyse der Lösungsansätze, die in den Echtzeit-Render-Engines zur Anwendung kommen. Dabei werden diese Lösungsansätze eingehend untersucht, um ein tieferes Verständnis für ihre Funktionsweise, ihre Vor- und Nachteile sowie ihre Anwendbarkeit in verschiedenen Kontexten zu gewinnen.

In Kapitel 5 dieser Arbeit werden die Methodik und Testsznarien zur Evaluation von EEVEE und Unreal Engine 5 vorgestellt. Ziel ist es, geeignete Testmethoden zu finden, zum Vergleich der Performance und visuellen Qualität in den Echtzeit-Render-Engines.

Kapitel 6 liegt der Fokus auf der Evaluierung der ausgeführten Versuche hinsichtlich ihrer visuellen Qualität. Hierbei werden die erzielten Ergebnisse in Bezug auf visuelle Aspekte analysiert und bewertet.

Kapitel 7 widmet sich der Bewertung der Performance. Es werden die Ergebnisse der Testsznarien analysiert und mögliche Optimierungsmöglichkeiten zur Verbesserung der Echtzeit-Lichtberechnung erörtert.

³ EEVEE ist die Echtzeit-Render-Engine in Blender.

Schließlich gibt Kapitel 9 einen Ausblick auf mögliche zukünftige Technologien und Entwicklungen. Es werden aktuelle Forschungstrends und innovative Ansätze diskutiert, die das Potenzial haben, noch realistischere und leistungsfähigere Darstellungen zu ermöglichen.

2 Grundlagen

Dieses Kapitel widmet sich den Grundlagen der Lichtberechnung in Echtzeit-Render-Engines. Um ein fundiertes Verständnis für dieses Thema zu entwickeln, werden verschiedene Schlüsselkonzepte behandelt, darunter Shader, Hardware, die Definition von Echtzeit, die physikalischen Grundlagen und die Rendergleichung.

2.1 Shader

Shader sind Computerprogramme, die für die Berechnung und Darstellung von visuellen Effekten, Licht, Schatten und Materialien in 3D-Szenen verantwortlich sind. Die Hauptaufgabe von Shadern besteht darin, das Aussehen von Objekten auf dem Bildschirm zu definieren und zu manipulieren. Das Berechnen dieser Bilder wird von Render-Engines übernommen, die diese Berechnung vornehmen. Shader nehmen eine zentrale Rolle in der sogenannten Rendering Pipeline ein. Diese beschreibt den Ablauf, wie dreidimensionale Objekte in zweidimensionale Bilder umgewandelt werden. Dieser Prozess beinhaltet verschiedene Schritte, die von der Vorbereitung der Geometrie bis zur Anwendung von Texturen, Beleuchtung und Effekten reichen (Abbildung 2.1) und für jedes Pixel auf einem Computerbildschirm durchlaufen werden muss [2, S. 11].

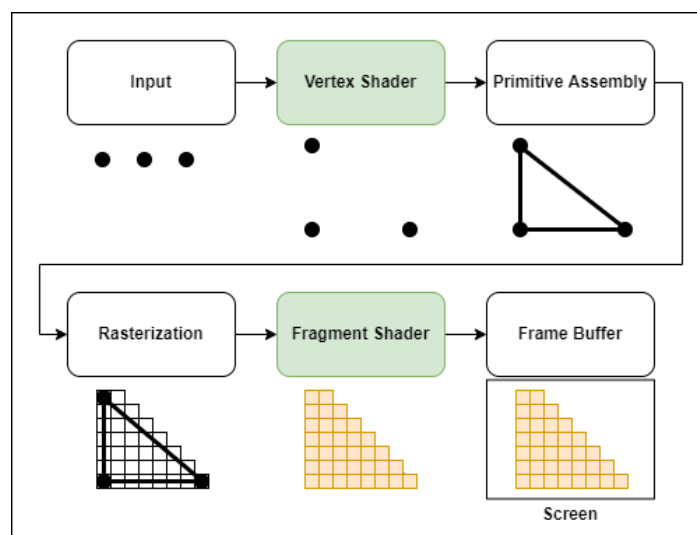


Abbildung 2.1: Beispiel einer Rendering Pipeline
Quelle: Darstellung von Carmen Cincotti [3]

Für Echtzeit-Render-Engines ist eine effiziente Implementierung von Shadern und ihren Techniken von essenzieller Bedeutung. Am Ende entscheiden die Komplexität der Shader und die Optimierung der Arbeitsprozesse der Pipeline über die Geschwindigkeit bei der Generierung von Bildern.

2.2 Hardware

Für die Ausführung der Schritte in der Rendering Pipeline ist die Graphic Processor Unit (GPU) zuständig. Die GPU ist eine spezialisierte Hardware-Komponente, die für die Beschleunigung von Grafikberechnungen verantwortlich ist und in der Lage ist, die Aufgaben in der Rendering Pipeline effizient auszuführen. Sie zeichnen sich besonders durch hohe Geschwindigkeit für parallele Aufgaben aus. Die Aufgaben in der GPU übernehmen sogenannte Shader-Kerne [2, S. 30], die für die Berechnung der in den Programmcode von Shader verantwortlich sind [2, S. 13].

In modernen GPUs befinden sich neben den Shader-Kernen aber noch weitere Kerne, sogenannte Raytracing-Kerne. Diese sind für die Beschleunigung von Raytracing Prozessen optimiert. Die Einführung von dedizierten Raytracing Kernen in GPUs begann mit der NVIDIA⁴ Turing-Architektur, die im August 2018 eingeführt wurde [4]. Die Turing-Architektur wurde erstmals in Grafikkarten der GeForce RTX 20-Serie implementiert [5]. Diese Grafikkarten waren die ersten, die spezielle Hardware-Einheiten für Raytracing-Berechnungen enthielten. Die Funktionsweise von Raytracing wird in Kapitel 3.4 beschrieben.

Neben den Kernen ist aber noch eine weitere Komponente für die Effizienz der Hardware in Echtzeit-Render-Engines von entscheidender Bedeutung.

Video Random Access Memory (VRAM) ist eine spezialisierte Art von Arbeitsspeicher, der in GPUs verwendet wird, um Grafikdaten zwischenspeichern [2, S. 1006]. GPUs benötigen schnellen Zugriff auf große Mengen an Daten, um Bilder flüssig darzustellen. Insgesamt ist VRAM für Echtzeit-Render-Engines entscheidend, da es die Leistung, Detailgenauigkeit und Interaktivität der gerenderten Grafiken beeinflusst. Je mehr VRAM vorhanden ist, desto besser kann die Engine komplexe Szenen in Echtzeit darstellen.

2.3 Definition Echtzeit

Das Hauptziel von Echtzeit-Render-Engines ist es, eine interaktive Anwendung in Echtzeit zu ermöglichen, die eine hohe visuelle Qualität bietet und gleichzeitig eine reibungslose Bildrate und Responsivität gewährleistet. Das Ziel besteht darin, eine möglichst genaue Annäherung an die physikalischen Phänomene der realen Welt zu erreichen, ohne dabei die Leistungsfähigkeit der Engine zu beeinträchtigen.

⁴ NVIDIA Corporation ist ein Hersteller von GPUs.

Laut der ISO/IEC 2382:2015 [6] versteht man unter Echtzeit *den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind* [7, S. 39]. Diese Definition weist jedoch darauf hin, dass die Interpretation der "vorgegebenen Zeitspanne" variieren kann, was zu unterschiedlichen Ansichten darüber führen kann, was tatsächlich als Echtzeit betrachtet wird.

In dieser Bachelorarbeit wird deshalb auf die Quantifizierung und Messung von Echtzeit zurückgegriffen, die im allgemeinen Anwendungsgebiet als Aspekte der Interaktivität und minimalen Verzögerung zwischen der Zeit der Eingabe und der Zeit der Ausgabe definiert wird [7, S. 39]. Dazu werden zwei messbare Größen herangezogen: Frames per Seconds (FPS) und Latenz.

FPS gibt die Bildrate an, wie viele Bilder pro Sekunde auf dem Bildschirm gerendert und angezeigt werden [2, S. 1]. Dies ist eine der häufigsten Metriken zur Bewertung der Leistungsfähigkeit, die eine einfache Vergleichbarkeit von verschiedenen Anwendungen erlaubt. Grundsätzlich kann in Echtzeit im Zusammenhang mit FPS gesprochen werden, sobald die Bilder nicht mehr als einzelne statische Bilder wahrgenommen werden, sondern als kontinuierliche und flüssige Bewegung [1, S. 1]. Ab einer Bildrate von 24 FPS gilt eine Darstellung als akzeptabel [1, S. 1]. Je nach Anwendungsbereich kann diese Bildrate aber variieren und höher liegen.

Der Begriff Latenz kann je nach Kontext für unterschiedliche Verzögerungen innerhalb eines Computersystems stehen. Im Kontext der Bachelorarbeit bezieht sich der Begriff Latenz auf die Zeitspanne, die eine Render-Engine benötigt, um die entsprechenden visuellen Ergebnisse auf dem Bildschirm darzustellen. Die Latenz hängt somit mit den FPS zusammen und wird üblicherweise in Millisekunden (ms) gemessen. Eine Latenz von 41,6 ms entspricht beispielsweise 24 FPS, da $1000 \text{ ms} / 24 \text{ FPS} \approx 41,6 \text{ ms}$. Der Vorteil der Latenz ist die genauere Betrachtung der Kosten bestimmter Funktionen, wie z.B. Lichtberechnungen, auf die Renderzeit eines Bildes.

2.4 Abgrenzung Offline-Render-Engine

Es gibt zwei Haupttypen von Render-Engines: Offline-Render-Engines und Echtzeit-Render-Engines, die sich in ihrer Funktionsweise unterscheiden.

Echtzeit-Render-Engines sind darauf ausgelegt, Bilder und Animationen in Echtzeit zu berechnen und anzuzeigen. Sie werden häufig in Videospielen, virtuellen Umgebungen, Simulationen und interaktiven Anwendungen eingesetzt. Echtzeit-Render-Engines zeichnen sich durch ihre Fähigkeit

aus, eine hohe Bildrate zu liefern, um eine flüssige visuelle Darstellung und eine reaktionsschnelle Benutzererfahrung zu ermöglichen.

Offline-Render-Engines hingegen konzentrieren sich auf die Berechnung von hochwertigen, fotorealistischen Bildern und Animationen. Sie werden häufig in der Filmproduktion, Architekturvisualisierung und anderen Bereichen eingesetzt, in denen hohe Qualität und Genauigkeit wichtiger sind als Echtzeit-Feedback.

Thus, interactive game applications (for which animating at a high number of frames per second is essential for success) must rely on the fast algorithms that compromise on realism. On the other hand, movie production applications have the luxury of being able to devote hours to computing a single frame of animation [8, S. 118].

Eine der schwierigsten Herausforderungen bei der Darstellung von Bildern in Echtzeit ist die Simulation von Licht. In den folgenden Abschnitten werden wir uns eingehender mit dieser Herausforderung befassen und untersuchen, wie Echtzeit-Render-Engines mit den komplexen physikalischen Phänomenen umgehen.

2.5 Physikalische Grundlagen

Render-Engines können Licht nicht einfach darstellen, weil Licht in der realen Welt äußerst komplex ist und eine Vielzahl von physikalischen Phänomenen umfasst, die schwer zu simulieren sind. Sie enthalten zu viele Informationen für eine Render-Engine, um sie in ihrer Gänze darzustellen. Deshalb werden Prinzipien der Physik und Geometrie mit mathematischen Modellen und Algorithmen annäherungsweise in diesen Programmen implementiert. Zur Behandlung von Licht in solchen Systemen werden die Grundlagen der geometrischen Optik herangezogen [2, S. 313].

Die zugrunde liegende Physik, die Licht quantifizierbar macht, nennt sich Radiometrie, die sich mit der Messung und Berechnung von elektromagnetischer Strahlung befasst [2, S. 267]. Somit können physikalische Phänomene wie Reflexion, Absorption, Brechung mit radiometrischen Größen erklärt werden.

Die ausgesendete Energiemenge Q von einer Lichtquelle ist der Strahlungsfluss in Watt ($\Phi_e = \frac{dQ}{dt}$). Die Bestrahlungsstärke ist das auf eine Fläche auftreffende Licht ($E_e = \frac{d\Phi_e}{dA}$) oder von einer Fläche abgegebene Licht ($M_e = \frac{d\Phi_e}{dA}$). Das von einer Lichtquelle in einen bestimmte Raumwinkel ($\Omega = \frac{A}{r^2}$) abgegebene Licht ist die Strahlstärke ($I_e = \frac{\Phi_e}{d\Omega}$). Die Strahldichte beschreibt ($L_e = \frac{d^2\Phi_e}{dA_{proj} \cdot d\Omega}$) den Zusammenhang zwischen der Bestrahlungsstärke und Strahlstärke. [9, S. 14]

Das Pendant zur Radiometrie ist die Photometrie und beschreibt die Gewichtung der radiometrischen Größen auf den für das menschliche Auge sichtbaren Teil des Lichts [10, S. 441]. Das Verhältnis von photometrischen zu radiometrischen Größen kann in Abbildung 2.2 abgelesen werden.

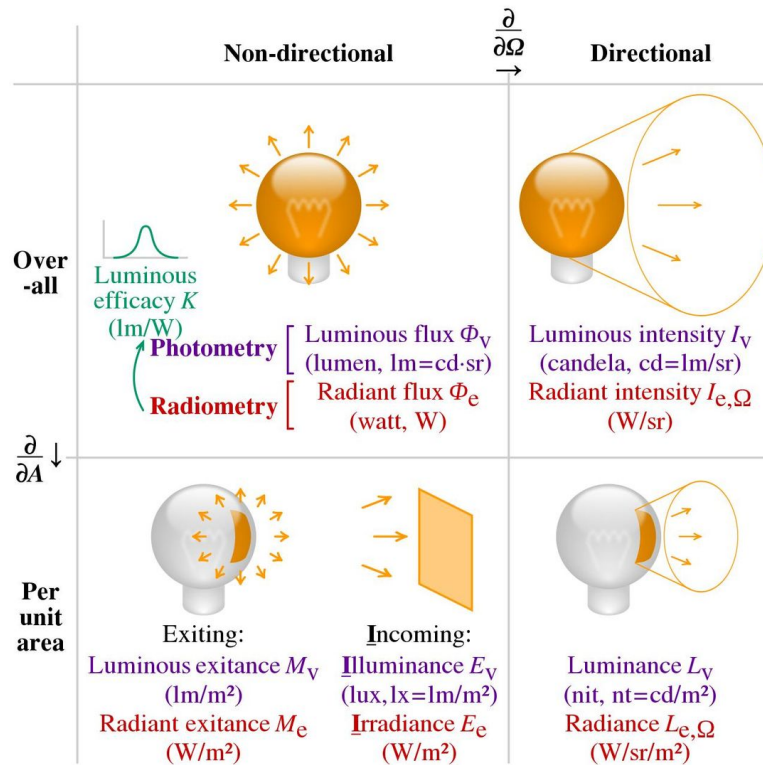


Abbildung 2.2: Verhältnis von radiometrischen und photometrischen Größen
 Quelle: Abbildung von Cmglee [11]

Diese radiometrischen Größen werden durch ein Modell ergänzt, das Lichtstrahlen in Render-Engines entlang gerader Linien durch ein Vakuum beschreibt. Diese Strahlen beeinflussen sich nicht gegenseitig und ihre Energiebeiträge werden ohne gegenseitige Interferenzen einfach summiert. Das ermöglicht für Render-Engines die Berechnung für Licht Interaktionen mithilfe eines geometrischen Strahls und eines Vektors. [8, S. 335]

Die Radiometrie liefert uns die notwendigen Größen und Konzepte, um die Wechselwirkung von Licht mit Oberflächen und Materialien zu verstehen. Dieses Verständnis bildet die Grundlage für die Rendergleichung, die eine zentrale Rolle bei der Generierung von realistischen Bildern spielt.

2.6 Rendergleichung

Die Rendergleichung ist eine mathematische Gleichung, die den physikalischen Prozess der Lichtausbreitung in einer 3D-Szene beschreibt. Sie wurde von James Kajiya im Jahr 1986 [12, S. 143] entwickelt und hat einen großen Einfluss auf die Entwicklung von Render-Algorithmen. Sie lautet:

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{\Omega} f_r(\mathbf{p}, \omega_i, \omega_o) \cdot L_i(\mathbf{p}, \omega_i) \cdot (\mathbf{n} \cdot \omega_i) d\omega_i \quad (1)$$

Die Rendergleichung lässt sich in verschiedene Komponenten unterteilen:

- $L_o(\mathbf{p}, \omega_o)$ ist die ausgehende Strahldichte an einem Punkt \mathbf{p}
- $L_e(\mathbf{p}, \omega_o)$ ist die Strahldichte die das Objekt selbst abgibt
- \int_{Ω} beschreibt die Gesamtheit aller Raumwinkel unter der Hemisphäre
- $f_r(\mathbf{p}, \omega_i, \omega_o)$ ist die Reflexionsfunktion des Objektes
- $L_i(\mathbf{p}, \omega_i)$ ist die ankommenden Strahldichte an einem Punkt \mathbf{p}
- $(\mathbf{n} \cdot \omega_i)$ ist das Skalarprodukt zwischen dem Einfallswinkel und der Oberflächennormale

Die Oberflächennormale ist ein Vektor, der senkrecht zur Oberfläche eines Objekts steht und die Ausrichtung der Oberfläche an einem bestimmten Punkt angibt.

Eine vereinfachte Schreibweise der Rendergleichung lässt sie wie folgt formulieren:

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + L_r(\mathbf{p}, \omega_o) \quad (2)$$

- $L_r(\mathbf{p}, \omega_o)$ beschreibt hierbei die reflektierende Strahldichte

Die ursprüngliche Rendergleichung berücksichtigt nicht die Betrachtung von Transmission bzw. Brechung. Dennoch lassen sich diese Terme durch die Betrachtung einer gespiegelten Hemisphäre ergänzen:

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + L_r(\mathbf{p}, \omega_o) + L_t(\mathbf{p}, \omega_o) \quad (3)$$

Dieser Term kann nun in einer allgemeingültigen Rendergleichung zusammengefasst werden, der alle Richtungen auf einer Kugel S beinhaltet.

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_S f_s(\mathbf{p}, \omega_i, \omega_o) \cdot L_i(\mathbf{p}, \omega_i) \cdot (\mathbf{n} \cdot \omega_i) d\omega_i \quad (4)$$

- $f_s(\mathbf{p}, \omega_i, \omega_o)$ beinhaltet nun die Brechungs- und Reflexionseigenschaft des Objekts

The rendering equation is significant in that it sums up all possible paths in a simple-looking equation [2, S. 438]. Sie berücksichtigt die Eigenschaften der Oberfläche eines Objekts, die Einflüsse der

Beleuchtung und die Richtungsabhängigkeit der Lichtausbreitung. Die Lösung der Rendergleichung ist eine Herausforderung, da sie ein Integrieren über alle möglichen Einfallswinkel ω_i erfordert. Es gibt verschiedene numerische Verfahren und Approximationen, um die Rendergleichung zu lösen und realistische Beleuchtungseffekte in Render-Algorithmen zu erzeugen. Deshalb wurden verschiedene Lösungsansätze formuliert, die in Offline- wie auch Echtzeit-Render-Engines verwendet werden, um Teile oder die gesamte Rendergleichung möglichst genau zu approximieren.

3 Rendertechniken

In diesem Kapitel werden wir uns mit den unterschiedlichen Beleuchtungsmodellen befassen, die zur Lösung der Rendergleichung in Bezug auf direkte und indirekte Lichtberechnungen verwendet werden. Wir werden die grundlegenden Konzepte und Ansätze untersuchen, die von den Render-Engines angewendet werden, um realistische Lichteffekte zu erzeugen.

3.1 Physically based shading

Physically based shading⁵ (PBS), auch als Physically based rendering (PBR) bekannt, ist eine Methode zur Darstellung von Oberflächenmaterialien in Render-Engines. Diese Methodik strebt danach, die physikalischen Eigenschaften von Materialien auf realistische Weise zu simulieren [1, S. 44]. Innerhalb der Rendergleichung ist das zentrale Ziel von PBS die Lösung der Funktion $f_s(\mathbf{p}, \omega_i, \omega_o)$.

PBS setzt auf die Modellierung von realen Materialien, indem es Eigenschaften wie das Brechungsgesetz, Reflexionsgesetz und den Energieerhaltungssatz beachtet. Eine herausfordernde Aufgabe besteht darin, möglichst viele Materialeigenschaften eines Objekts erfassen und mittels einer einfachen mathematischen Funktion darzustellen, die auch für komplexe Szenarien mit zahlreichen variierenden Komponenten berechenbar bleibt.

Prinzipiell werden Lichtberechnungen für Oberflächen berechnet, *indem die passende Formel zur Berechnung der Oberfläche "geladen" und dann für bestimmte Punkte auf der Oberfläche berechnet wird* (freie Übersetzung) [8, S. 127].

BRDF steht für "Bidirectional Reflectance Distribution Function" (bidirektionale Reflexionsverteilungsfunktion), die in der Render-Engine verwendet wird, um das

⁵ Shading bezieht sich auf den Prozess, bei dem die Oberflächen eines Objekts mit Farben oder Texturen versehen werden, um deren Aussehen und Lichtverhalten zu simulieren.

Reflexionsverhalten von Oberflächen zu modellieren. Die BRDF ist die Funktion $f_r(\mathbf{P}, \omega_i, \omega_o)$ innerhalb der Rendergleichung.

Hierfür müssen 3 verschiedene Arten von Reflexionen berücksichtigt werden (siehe Abbildung 3.1), diffuse Reflexion (Streuung des Lichts in alle Richtungen), gestreuter Reflexion (reflektiertes Licht in einer bestimmten Richtung) und spiegelnder Reflexion (reflektiertes Licht in eine Richtung).

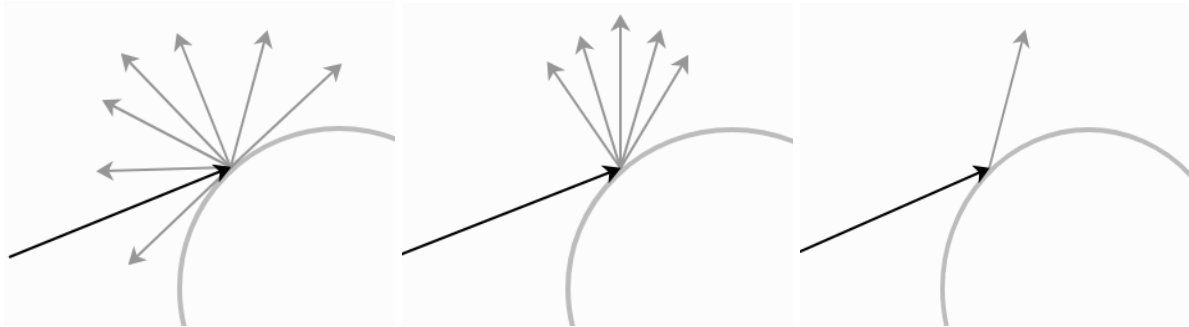


Abbildung 3.1: Veranschaulichung diffuser (links), gestreuter (mitte) und spiegelnder Reflexion (rechts)
Quelle: Blender Dokumentation [13]

Die Basis in BRDF ist die Mikrofacetten Theorie Die Theorie postuliert, dass die Oberfläche eines Materials aus vielen winzigen mikroskopischen Spiegeln besteht. Diese Facetten können sich in ihrer Orientierung und Neigung unterscheiden [1, S. 1113]. Wenn Licht auf eine Oberfläche trifft, wird es nicht gleichmäßig reflektiert. Stattdessen wird das Licht von den Mikrofacetten der Oberfläche reflektiert. Jede Mikrofacette wirft Licht in eine bestimmte Richtung zurück. Die Ausrichtung der Mikrofacetten liegt eine komplexe Berechnung mit mehreren Komponenten zugrunde.

Für die Berechnung von diffus gestreuten Reflexionen wird anstelle der Mikrofacetten Theorie das Lambertsche Gesetz verwendet. Das Lambertsche Gesetz besagt, dass die Beleuchtungsstärke abnimmt, je größer der Winkel zwischen der Oberflächennormalen und der Richtung des einfallenden Lichts ist. Dadurch müssen zur Berechnung nur der Winkel des einfallenden Lichts und die Normale der Oberfläche bekannt sein. Dadurch wird die Berechnung für diese Oberflächen deutlich vereinfacht.

Die bedeutenden Bestandteile einer bidirektionalen Reflexionsverteilungsfunktion (BRDF) sind im Folgenden aufgeführt:

1. Reflektivität: Dies ist der Anteil des einfallenden Lichts, der von einer Oberfläche reflektiert wird.
2. Rauheit: Die Rauheit beschreibt, wie glatt oder grob die Oberfläche eines Materials ist. Sie beeinflusst die Verteilung der reflektierten Lichtstrahlen und beeinflusst direkt die Intensität von Reflexionen.
3. Metallisierung: Dieser Wert gibt an, wie metallisch oder nichtmetallisch ein Material ist.

Die Liste stammt als Auszug aus Physically-Based Shading at Disney [14, S. 12–13].

In Ergänzung zu diesen Hauptkomponenten existieren weitere Eigenschaften, die jedoch in der vorliegenden Bachelorarbeit nicht weiter untersucht werden. Dennoch werden sie in einer beigelegten Abbildung 3.2 aufgeführt und stellen einen integralen Bestandteil eines BRDF dar.

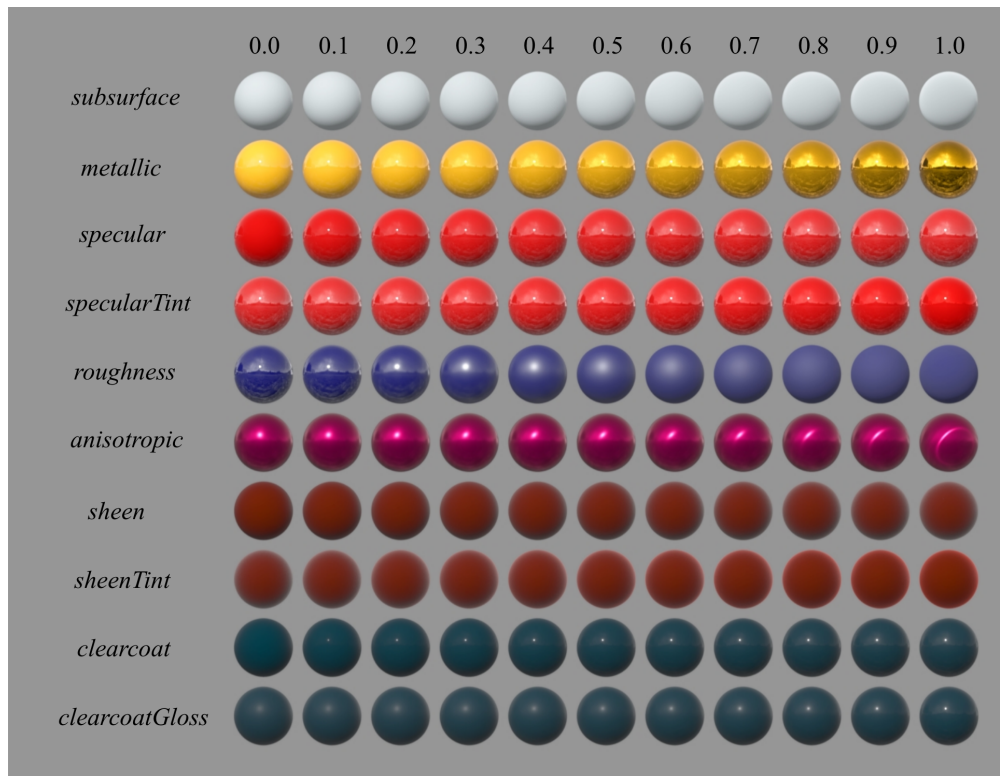


Abbildung 3.2: Alle Materialeigenschaften von Disneys BRDF

Quelle: Abbildung aus Physically-Based Shading at Disney [15, S. 13]

Zusätzlich zur bidirektionalen Reflexionsverteilungsfunktion (BRDF) gibt es die bidirektionale Transmissionverteilungsfunktion (BTDF) für Transmissionen. Diese beiden Funktionen werden häufig als Gesamtheit in einer bidirektionalen Streuverteilungsfunktion (BSDF) zusammengefasst [1, S. 1045].

Beim Übergang von einem Medium in ein anderes kommt es nicht nur ausschließlich zu Reflexions- oder Transmissionsphänomenen, sondern auch zu Lichtbrechungen innerhalb des Mediums. Die bidirektionale Streuverteilungsfunktion für Oberflächenstreuung und -reflexion (BSSRDF, von "bidirectional scattering surface reflectance distribution function") beschreibt, wie Licht an einer Stelle aus einem Objekt austritt, die nicht mit dem Ort des Eintritts übereinstimmt [1, S. 63]. Diese Funktion ist eine erweiterte Form der BRDF (Bidirectional Reflectance Distribution Function). Das Konzept des Subsurface Scattering (SSS) kann durch die BSSRDF dargestellt werden (siehe Abbildung 3.3). Dieses Phänomen tritt auf, wenn Licht in ein transparentes oder transluzentes Material eindringt und dann innerhalb des Materials gestreut wird, bevor es wieder austritt. Die

BSSRDF erfasst diese Streuung und Reflexion auf subtilere und realistischere Weise, als es mit einer herkömmlichen BRDF oder BTDF möglich wäre.

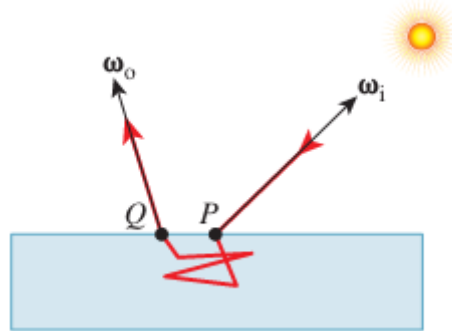


Abbildung 3.3: Das Licht gelangt an P in das Material von der Lichtquelle, die durch ω_i angezeigt wird, es bewegt sich entlang des roten Pfades, prallt innerhalb des Materials ab und tritt an Q in Richtung ω_o aus.
Quelle: Darstellung von James D. Foley [8, S. 738]

Insgesamt bildet das PBS-Konzept eine wichtige Grundlage für die modernen Render-Engines, um visuell ansprechende und physikalisch korrekte Bilder zu erzeugen. Dabei darf aber die BRDF Funktion nicht zu kompliziert sein, sodass die Lösung der Gleichung in Echtzeit erfolgen kann.

3.2 Global Illumination

Ergänzend zur Lösung der Rendergleichung müssen nicht nur die Materialeigenschaften, sondern das aus sämtlichen Richtungen einfallende Licht berücksichtigt werden, also die direkte und indirekte Beleuchtung. Beide zusammen ergeben Global Illumination.

Die direkte Beleuchtung ist das von einer Lichtquelle auf das Objekt oder die Szene fallende Licht. Unter indirekter Beleuchtung versteht man das Licht, das nicht direkt von Lichtquellen stammt, sondern durch Reflexion, Brechung und Streuung von den Oberflächen der Objekte in der Umgebung abgegeben wird (Abbildung 3.4).

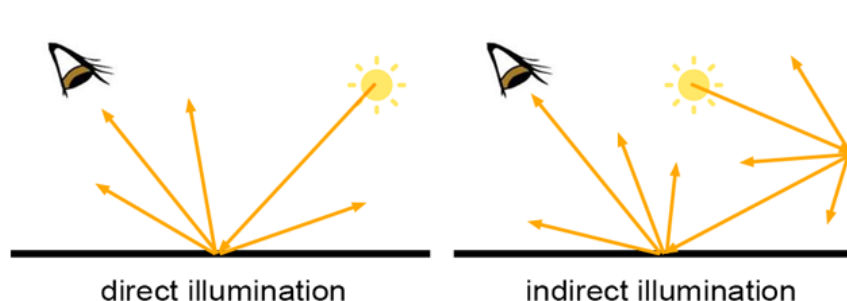


Abbildung 3.4: Darstellung der Betrachtung eines Punktes im Bezug auf die indirekten und direkten Beleuchtung
Quelle: Darstellung von Scratchapixel [16]

3.2.1 Direkte Beleuchtung

Um die direkte Beleuchtung zu beschreiben, ist es notwendig, die verschiedenen Arten von Lichtquellen zu betrachten. Hierbei handelt es sich um vereinfachte Modelle von Lichtquellen, die die physikalischen Grundlagen der Ausbreitung von Licht berücksichtigen. Diese Lichtquellen können in Bezug auf ihre Strahlrichtung unterschieden werden:

1. Punktlichtquelle: Eine Punktlichtquelle emittiert Licht gleichmäßig von einem einzigen Punkt aus in alle Richtungen. Sie erzeugt Kernschatten und strahlt gleichmäßig in alle Richtungen aus [2, S. 110].

2. Richtungslichtquelle: Eine Richtungslichtquelle sendet parallele Lichtstrahlen in eine bestimmte Richtung. Sie wird oft verwendet, um das Sonnenlicht zu simulieren [8, S. 381]. Richtungslichtquellen erzeugen Kernschatten.

3. Flächenlichtquelle: Eine Flächenlichtquelle besitzt eine bestimmte Größe und Form und emittiert Licht über ihre Fläche [2, S. 116]. Sie kann rechteckig, quadratisch oder beliebig geformt sein. Flächenlichtquellen erzeugen Übergangsschatten und sorgen für eine gleichmäßige Beleuchtung.

4. Spotlichtquelle: Eine Spotlichtquelle emittiert Licht in eine bestimmte Richtung, begrenzt durch einen kegelförmigen Lichtkegel. Sie kann als eine Art Punktlicht mit Richtung betrachtet werden [8, S. 133]. Eine Spotlichtquelle kann einen anpassbaren Spot-Winkel aufweisen. Diese Lichtquellen erzeugen harte Schatten und können für gezielte Beleuchtungseffekte eingesetzt werden [2, S. 114].

5. Umgebungslicht: Ein Umgebungslicht ist eine indirekte Lichtquelle, die eine gleichmäßige Beleuchtung in der gesamten Szene erzeugt. Sie simuliert das Licht, das von den Oberflächen der Umgebung reflektiert wird. Umgebungslicht hat weder eine spezifische Position noch eine bestimmte Richtung und erzeugt keine Schatten.

Um das Umgebungslicht in der Render-Engine zu simulieren, können spezielle Bilder namens HDRIs (High Dynamic Range Images) verwendet werden [2, S. 392]. Diese Bilder erfassen eine breite Palette von Helligkeitswerten und dienen als Grundlage für die Beleuchtungsberechnungen, um die Lichteffekte der Umgebung auf Objekten in einer Szene darzustellen (siehe Abbildung 3.5).



Abbildung 3.5: Beispiel eines HDRI (rechts), das als Umgebungsbeleuchtung in einer Kugel verwendet wird (links)

Quelle: Abbildung aus dem Buch Real-time Rendering [2, S. 438]

3.2.2 Indirekte Beleuchtung

Das Hauptziel der indirekten Beleuchtung besteht darin, die Einbeziehung anderer Objekte in einer Szene in die Lichtberechnung zu ermöglichen (Abbildung 3.6). Dabei werden die Wechselwirkungen des Lichts berücksichtigt, wenn es von einer Oberfläche reflektiert, gebrochen und gestreut wird.

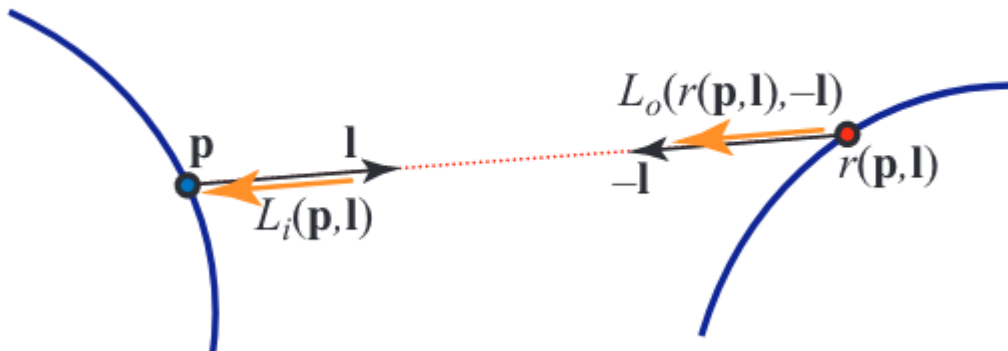


Abbildung 3.6: Die Beleuchtung an Position p wird von dem Punkt r indirekt durch die Beleuchtungsrichtung l mithilfe der einfallenden Strahllichte $L_i(p, l)$ bzw. ausgehenden Strahllichte $L_o(r(p, l), -l)$ beschrieben

Quelle: Abbildung aus dem Buch Real-time Rendering [2, S. 438]

3.3 Rasterisierung

Die praktische Umsetzung von Beleuchtung in Render-Engines erfolgt mithilfe der Rasterisierung. Das Ziel der Rasterisierung ist, die 3D-Objekte in einer Szene auf 2D-Bildschirmen darzustellen. Das erfolgt mithilfe von Abtastung der Oberflächen von Objekten [8, S. 17]. Diese werden in einer 3D-Szene hinsichtlich ihrer Eigenschaften bezüglich direkter, indirekter Beleuchtung und Materialeigenschaft überprüft und auf einem Raster aus Pixeln dargestellt (siehe Abbildung 3.7).

Bei der Abbildung der unterschiedlichen Oberflächen kommt ein sogenannter Z-Buffer oder Tiefenpuffer zum Einsatz [8, S. 392], der die Tiefeninformationen, also den Abstand der Oberfläche

eines Objekts, zur Kamera speichert. Die Oberfläche mit dem kleineren Wert für die Tiefe wird dargestellt und verdeckt die Oberfläche mit dem größeren Wert. Erst im Anschluss daran wird die Beleuchtung und Materialeigenschaften berücksichtigt.

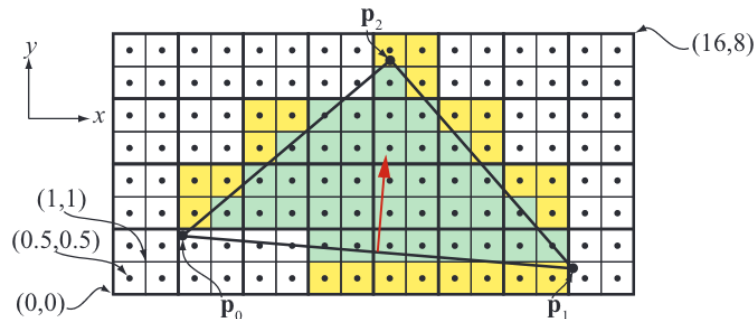


Abbildung 3.7: Darstellung eines Dreiecks mit den Punkten p_0 , p_1 , p_2 , das in einem 16 x 8 Pixelraster dargestellt wird. Die grünen Pixel stellen das Rechteck dar und die gelben die Anti Aliasing Pixel
Quelle: Darstellung aus dem Buch Real-time Rendering [2, S. 994]

In der Rasterisierung werden einzelne Pixel mehrmals abgetastet, sodass ein vollständiger erster Durchlauf erfolgt, um die Sichtbarkeit zu bestimmen, und dann erst ein zweiter oder mehrere weitere Durchläufe erfolgen, um alle Eigenschaften der Oberflächen zu erfassen. Das größte Problem bei der Rasterisierung ist der Treppenstufeneffekt oder auch Aliasing genannt, der sich dadurch ergibt, dass Oberflächen und Linien in Pixeln, die rechteckig sind, dargestellt werden und runde oder gebogene Linien nicht als solche abgebildet werden können. Anti-Aliasing-Algorithmen (siehe Abbildung 3.7) dienen dazu, das Flimmern oder die Treppeneffekte (Aliasing) zu reduzieren. Diese Algorithmen glätten die Kanten und sorgen für eine weichere Darstellung.

3.4 Raytracing

Raytracing stellt eine alternative Methodik zur Rasterisierung dar. Hierbei erfolgt die Abtastung durch einen Strahl, der vom Standpunkt des Betrachters durch jedes Pixel auf dem Bildschirm in die Szene "geschossen" wird [2, S. 443]. Beim Auftreffen der Strahlen auf Oberflächen werden sie gemäß den Materialeigenschaften reflektiert, gebrochen oder absorbiert. Hierfür können weitere Strahlen verwendet werden, die vom Schnittpunkt der Oberfläche einen Strahl aussenden (Sekundärstrahlen), beispielsweise zur Feststellung von Reflexion oder Schattierung. Die Strahlen werden anschließend zurückverfolgt und enthalten Informationen über die Farbe, Helligkeit und andere Eigenschaften der Oberfläche (Abbildung 3.8). Diese Informationen werden dann genutzt, um die Pixel auf einem Bildschirm zu berechnen.

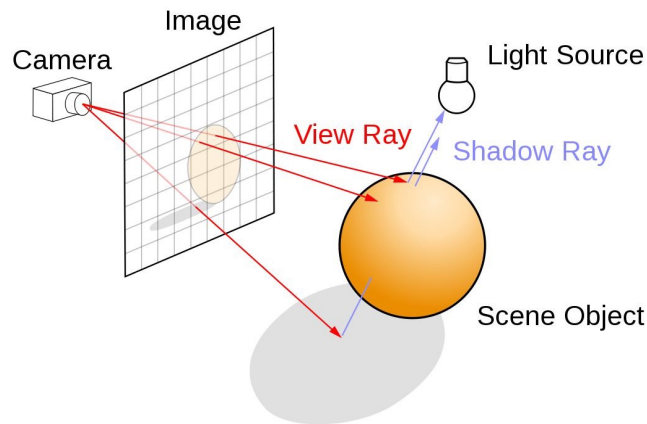


Abbildung 3.8: Darstellung von verschiedenen Strahlen durch ein Raster zur Beschreibung von Raytracing
 Quelle: Darstellung von Henrik [17]

Raytracing ist im Vergleich zu Rasterisierung rechenintensiver. In der Theorie muss jeder Strahl auf das Schneiden von Oberflächen von jedem Objekt in der Szene anhand der Oberflächennormale überprüft werden. Außerdem führt die Verfolgung eines Sekundärstrahls dazu, dass die gleiche Berechnung erforderlich ist. Eine unzureichende Bildabtastung kann sich als Unterschied in der Helligkeit zwischen benachbarten Pixeln zeigen und als sichtbares Rauschen wahrgenommen werden.

Raytracing steht vor der Problematik, nur Strahlen zu verfolgen, die direkt von der Oberfläche gebrochen, reflektiert oder absorbiert werden. Es erfolgt aber keine weitere Verfolgung über die Sekundärstrahlen hinaus. Eine Erfassung von indirekter Beleuchtung wird somit nicht berücksichtigt.

3.5 Path Tracing

Path Tracing beginnt ähnlich wie Raytracing, indem es Strahlen von der Kamera aus in die Szene sendet. Der Hauptunterschied zu Raytracing besteht darin, dass Path Tracing weitere Strahlen verfolgt (Lichtpfade), die an Oberflächen reflektiert oder gebrochen werden. Anstatt nur einen Strahl pro Pixel zu verfolgen, werden beim Path Tracing mehrere Strahlen verfolgt, um eine akkurate Darstellung des Pixels zu ermöglichen. Diese Strahlen werden zufällig reflektiert oder gebrochen und von der Kollision mit weiteren Oberflächen zufällige neue Strahlen verfolgt. Dieses Vorgehen wird so lange wiederholt, bis der Lichtstrahl auf eine Lichtquelle trifft. Eine Erfassung von Global Illumination wird somit möglich, da die Oberflächen von Objekten in Relation zueinander berücksichtigt werden. Damit lassen sich, wie in Abbildung 3.9 ersichtlich, sehr realistische Ergebnisse erzielen.



Abbildung 3.9: Vergleichsbild zwischen einem realen Foto (links) und eines mit Path Tracing gerenderten Bildes (rechts)
Quelle: Darstellung von Lionel David [18]

4 Analyse von Rendertechniken in Echtzeit-Render-Engine

Dieses Kapitel konzentriert sich auf die grundlegenden technischen Konzepte und Prinzipien von Unreal Engine 5 und Eevee. Hier wird die Funktionsweise dieser Engines im Hinblick auf Rasterisierung und Raytracing erörtert.

4.1 Vorstellung von Render-Engines

Wir beginnen mit einer kurzen Vorstellung der ausgewählten Render-Engines, die in dieser Bachelorarbeit untersucht werden. Dazu gehören Eevee, Cycles und Unreal Engine 5. Jede Engine wird kurz beschrieben, um einen Überblick über ihre Funktionen, Stärken und Anwendungsbereiche zu geben.

4.1.1 Blender

Blender besitzt zwei Render-Engines. Extra Easy Virtual Environment Engine (Eevee) ist eine Echtzeit-Render-Engine, die auf Geschwindigkeit und Interaktivität ausgelegt ist und gleichzeitig das Ziel verfolgt, PBR-Materialien zu rendern. Eevee verwendet eine Rendering Pipeline basierend auf Rasterisierung zur Berechnung von Lichteffekten. [19]

Bei Cycles handelt es sich um eine Offline-Render-Engine, die Path Tracing verwendet, um physikalisch akkurate Ergebnisse zu liefern [20]. Weshalb Cycles für die Vergleichbarkeit von Echtzeit-Render-Engines wichtig sind, wird in Kapitel 5.2.1 erläutert.

4.1.2 Unreal Engine 5

Unreal Engine 5 ist eine Echtzeit-Render-Engine und verwendet eine hybride Rendering Pipeline. Das bedeutet, dass Techniken zur Rasterisierung und Raytracing verwendet werden. Unreal Engine 5 verwendet ein globales Beleuchtungssystem namens "Lumen". "Lumen" ist nicht zu verwechseln mit der Einheit für den Lichtstrom.

4.2 Rendertechniken in EEVEE

4.2.1 Rasterisierung

EEVEE verwendet Disneys BSDF⁶ zur Berechnung von Materialeigenschaften [21]. Innerhalb von EEVEE erfolgt eine Darstellung von BRDF und BTDF mittels Screen-Space-Reflexionen (SSR) [22]. SSR, operiert im sogenannten Bildraum, wodurch ausschließlich Informationen genutzt werden, die auf dem Bildschirm angezeigt werden. Dabei wird das vorherige Bild zwischengespeichert. Dieses wird dann als Reflexion auf Oberflächen projiziert. EEVEE spart sich weitere komplexe Berechnungen und nutzt das Bildmaterial der Screen-Space Reflexionen auch für die Darstellung von Brechungen. Bei der Aufnahme des Reflexionsbildes werden Objekte mit Transmissionseigenschaften ausgelassen und nur für die Projektion auf die Oberflächen berücksichtigt.

4.2.2 Global Illumination

EEVEE nutzt Ambient Occlusion und Umgebungslicht zur Darstellung von indirekter Beleuchtung [23], [24].

Die originale Rendergleichung besagt, dass ein Punkt Licht aus sämtlichen Richtungen einer Halbkugel empfängt. Es kann jedoch vorkommen, dass ein Punkt durch benachbarte Punkte desselben Objekts oder anderer Objekte verdeckt wird, was zur Bildung von Schatten führt. Ambient Occlusion approximiert diesen Effekt, indem es die Positionen der verschiedenen Oberflächen eines Objekts zueinander in Beziehung setzt und die Abstände zwischen ihnen misst (siehe Abbildung 4.1).

⁶ Dabei handelt es sich um eine Erweiterung des in Kapitel 3.1 genannten BRDFs mit Berücksichtigung von Transmissionen.

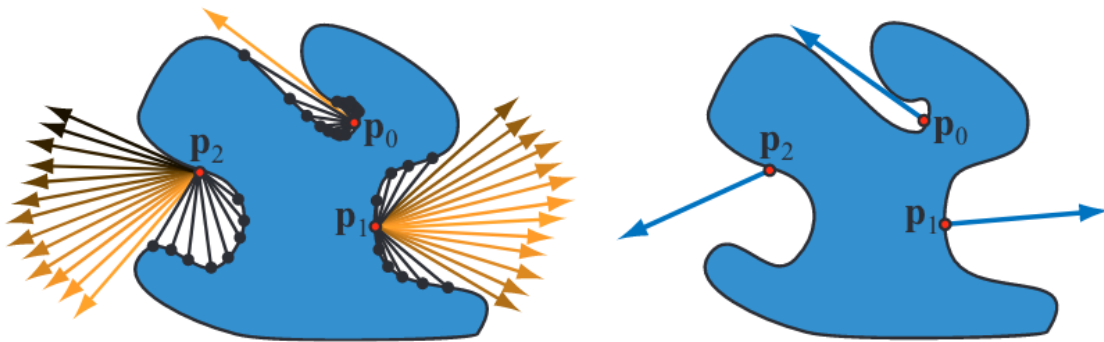


Abbildung 4.1: Beispielhafte Berechnung von Ambient Occlusion an p_0 , p_1 , p_2
 Quelle: Auszug aus dem Buch Real-time Rendering [2, S. 448]

Es handelt sich um eine lokale Berechnung, die nicht die Reflexion und Brechung von Lichtstrahlen berücksichtigt, die zwischen Oberflächen auftreten. Ähnlich verhält es sich mit der Verwendung von Umgebungslicht. Diese werden auf jedes Objekt in der Szene einzeln angewendet, um den Effekt von indirekter Beleuchtung zu simulieren. Es wird aber keine Beleuchtung von anderen Objekten in einer Szene in Betracht gezogen.

EEVEE bietet aber noch eine weitere Möglichkeit an, indirekte Beleuchtung zu simulieren: Irradiance Volumes [25]. Hierbei werden Lichtproben in einem dreidimensionalen Gitternetz angeordnet, um die Beleuchtung von der Umgebung einzufangen (siehe Abbildung 4.2). Diese Informationen werden dann, in Abhängigkeit von der Distanz zu den einzelnen Lichtproben, auf die Objekte in der Szene projiziert.

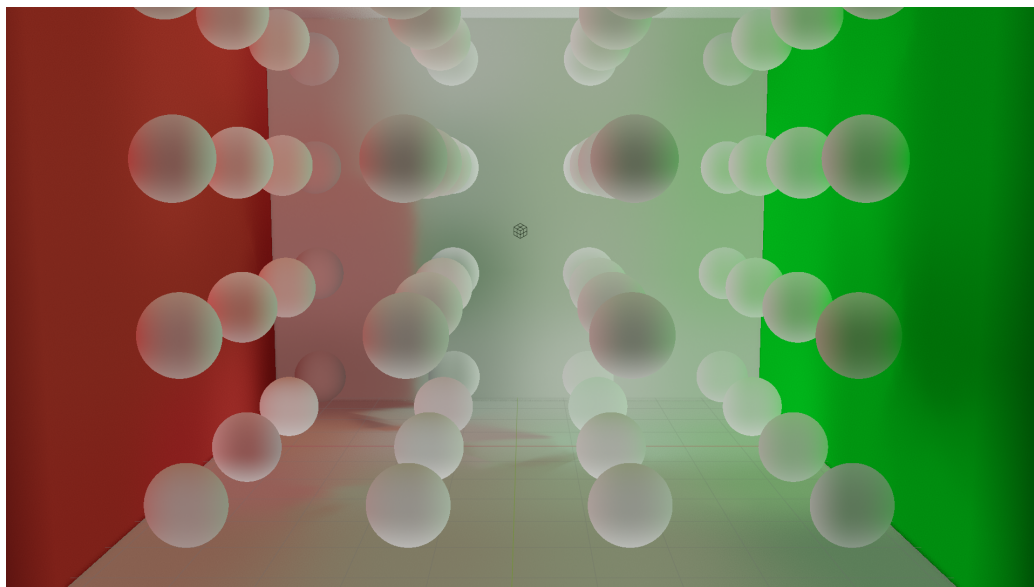


Abbildung 4.2: Darstellung von Lichtproben in EEVEE
 Quelle: Eigene Darstellung

Der Rechenaufwand für die Erstellung der Lichtproben in Echtzeit ist aber zu groß, weshalb diese Option nur mit Vorberechnung möglich ist. Außerdem ist eine Aktualisierung der Lichtproben nur durch ein erneutes Berechnen möglich. Aufgrund dessen wird diese Technik in der späteren Evaluation nicht berücksichtigt.

4.2.3 Schatten

EEVEE verwendet Shadow Maps zur Generierung von Schatten. Shadow Maps sind Texturen⁷, die Informationen über die Sichtbarkeit von Objekten aus der Perspektive der Lichtquelle speichern [26, S. 43]. Eine Shadow Map wird aus der Sicht der Lichtquelle berechnet und enthält Tiefeninformationen über die Objekte in der Szene [2, S. 234].

In EEVEE werden mehrere Shadow Maps, über mehrere Bilder verteilt, aufgenommen, die in ihrer Position variiert werden. Die Ergebnisse der verschiedenen Shadow Maps werden überlagert, um den Effekt von Übergangsschatten zu simulieren. Allerdings wird dieser Prozess jedes Mal von vorne gestartet, sobald ein Objekt, Lichtquelle oder Betrachter bewegt wird.

Während des Renderns wird für jedes Pixel überprüft, ob es von der Lichtquelle beleuchtet wird oder im Schatten liegt. Dazu wird die Tiefeninformation der Shadow Map verwendet. Wenn die Tiefeninformation des Pixels größer ist, als die entsprechende Tiefeninformation in der Shadow Map, befindet sich das Pixel im Schatten und wird entsprechend abgedunkelt (siehe Abbildung 4.3).

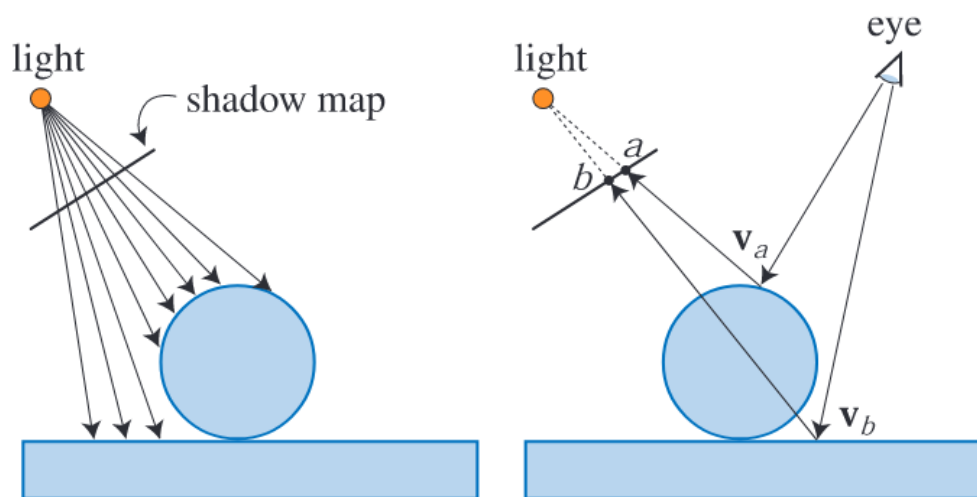


Abbildung 4.3: Darstellung der Erstellung einer Shadow Map aus Sicht der Lichtquelle (links) und der Berechnung des Bildes beim Rendern (rechts)

Quelle: Darstellung aus dem Buch Real-time Rendering [2, S. 235]

⁷ Texturen sind Bilder, die auf Oberflächen aufgebracht werden, um visuelle Informationen in Form von Farben, Muster, Reflexionen und Beleuchtungseffekte simulieren.

Diese Texturen werden im VRAM (Video Random Access Memory) von GPUs gespeichert und müssen bei Bewegung der Lichtquelle oder Betrachter erneut berechnet werden. Für jede Lichtquelle muss separat eine eigene Shadow Map erstellt werden.

4.3 Unreal Engine 5 Rendertechniken

4.3.1 "Lumen"

Bei "Lumen" handelt es sich um ein System für Global Illumination und Reflexionen [27]. "Lumen" verwendet verschiedene Beschleunigungs-Techniken, um Informationen über das Aussehen der Pixel zu sammeln. Diese sind optimiert, um eine minimale Anzahl an Strahlen zu verfolgen und Echtzeit gewährleisten zu können.

4.3.1.1 Raytracing in "Lumen"

Dazu werden Algorithmen verwendet, die die Anzahl der zu verfolgenden Strahlen und die Anzahl der Tests bezüglich der Schnittpunkte mit Oberflächen reduzieren. Das passiert mit der Unterstützung von Rasterisierung. Strahlen werden vorerst auf die Schnittpunkte im Bildraum überprüft [28]. Das erfolgt durch die Testung der Strahlen mit den Tiefenwerten aus dem Z-Buffer. Das bedeutet, dass Strahlen vorerst nur verfolgt werden, wenn sie im Sichtfeld sind. Der Z-Buffer hilft bei der Testung der Schnittpunkte, weil der Schnittpunkt eines Strahls für das Pixel bekannt ist und nicht gegen die Kollision mit Oberflächen getestet werden muss. Für Strahlen, die außerhalb des Bildes oder hinter einem Objekt fliegen, wo kein Tiefenwert existiert, wird eine zweite Methode verwendet.

Die Testung eines Strahls auf Schnittpunkte gegenüber allen Oberflächen außerhalb des sichtbaren Bereichs ist für Echtzeit nicht zu gewährleisten. Hier kommen Mesh Distance Fields zum Einsatz [29]. Mesh Distance Fields repräsentieren den Raum um ein Objekt, indem sie für jeden Punkt im Raum den kürzesten Abstand zur nächstgelegenen Oberfläche des Objekts berechnen. Die Beschreibung eines Objekts im Mesh Distance Field wird beim Importieren von Objekten angelegt und im VRAM abgelegt. Daraufhin wird für die Strahlen Ray Marching verwendet [30]. Das bedeutet, dass ein Strahl den leeren Raum innerhalb eines Distance Fields ignoriert und nur an der Oberfläche der Kugel überprüft wird, ob eine Kollision mit einer Oberfläche stattgefunden hat (siehe Abbildung 4.4). Das wird so lange durchgeführt, bis der Radius kleiner als ein festgelegter Wert ist und das Objekt als getroffen gilt. Dadurch können Strahlen auf eine effiziente Art auch außerhalb des sichtbaren Bereichs berechnet werden.

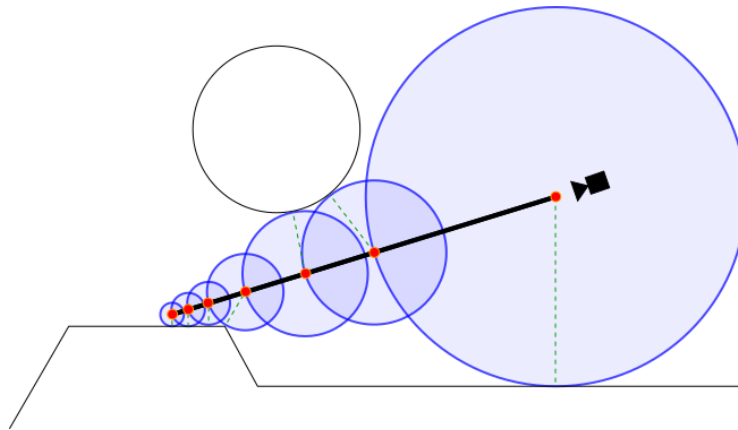


Abbildung 4.4: Darstellung von Ray Marching in einer zweidimensionalen Umgebung
 Quelle: Darstellung von Adrian Biagioli [31]

Das Problem von Mesh Distance Fields ist, dass nur eine Näherung der tatsächlichen Geometrie erreicht wird. Sie arbeiten mit einer diskreten Rasterisierung der 3D-Welt und können daher die feinen Details von komplexen Objekten nicht immer genau erfassen. Die Darstellung der Objekte mithilfe von Mesh Distance Fields enthält aber keine Informationen über die Materialeigenschaften des Objekts.

4.3.1.2 Surface Cache

Hierfür kommt der Surface Cache ins Spiel [28]. Der Surface Cache ist ein Speicher, der alle Oberflächeneigenschaften eines Objektes enthält. Dabei erfasst und speichert er Informationen über die Beleuchtung von Oberflächen. Dazu gehören Reflexionen, Schatten und Beleuchtung. "Lumen" erfasst die Materialeigenschaften aus allen Blickwinkeln für jedes Objekt vor der Verwendung in Echtzeit. Die erfassten Positionen, auch "Cards" genannt, werden für jedes Objekt generiert und dienen dazu, die Komplexität der Berechnung von Strahlen mit Oberflächen auf ein Minimum zu reduzieren. Trifft ein Strahl auf die Oberfläche an einem Punkt, dann werden für diesen Punkt auf dem Objekt die vorliegenden Material- und Umgebungseigenschaften im Surface Cache "nachgeschlagen". Ein Surface Cache ermöglicht es, bereits berechnete Informationen über die Oberflächen zu speichern, anstatt sie für jedes Bild oder jede Interaktion erneut zu berechnen. Dazu wird der Surface Cache im VRAM abgelegt.

Die Kombination aus Surface Cache und Mesh Distance Fields wird in der "Lumen Scene" [28] (siehe Abbildung 4.5) dargestellt.



Abbildung 4.5: "Lumen Scene": Vor Auslesen des Surface Cache (links) und nach Auslesen des Surface Cache (rechts)

Quelle: Eigene Darstellung

4.3.1.3 Global Illumination in "Lumen"

"Lumen" zeichnet sich dadurch aus, indirekte Beleuchtung darzustellen, ohne dass auf die Notwendigkeit von Path Tracing zurückgegriffen werden muss. Ähnlich wie in Eevee werden hierbei Lichtproben in einem festen Gittermuster angeordnet (siehe Abbildung 4.2). Von den Strahlen, die Oberflächen im Bildbereich treffen, werden Sekundärstrahlen zu benachbarten Lichtproben ausgesendet. Die Daten der verwendeten Lichtproben und die Informationen über die Oberfläche des getroffenen Objekts werden dann für jede getroffene Lichtprobe im Surface Cache gespeichert. Jeder neue Strahl aktualisiert und ergänzt die Werte der jeweiligen Lichtprobe im Surface Cache. Dieser Prozess erfolgt Bild für Bild.

Dadurch kann im Gegensatz zu Blender auch indirekte Beleuchtung in Echtzeit zur Darstellung der Oberfläche berücksichtigt werden.

4.3.2 Schatten

Unreal Engine 5 nutzt in "Lumen" Shadow Maps zur Erstellung von Schatten, jedoch kann dank des Einsatzes von RT-Kernen die Möglichkeit von Raytracing-Schatten verwendet werden, diese werden ergänzend zu "Lumen" verwendet, um eine physikalisch akkuratere Darstellung von Schatten zu ermöglichen. Nur durch die RT-Kerne kann die Performance geliefert werden, die eine Verfolgung

von Schattenstrahlen ermöglicht [32]. Deshalb wird diese Technik in dieser Bachelorarbeit benutzt. In Unreal Engine 5 werden Schatten erzeugt, indem Schattenstrahlen von einer Oberfläche direkt zur Lichtquelle fliegen. Auf dem Weg wird überprüft, ob sie von anderen Oberflächen blockiert werden (siehe Abbildung 3.8 im Kapitel 3.4). Basierend auf der Größe der Lichtquelle wird der Schatten eines Objekts in der Nähe der Kontaktfläche schärfer dargestellt, als weiter entfernt (Abbildung 4.6) [32].



Abbildung 4.6: Szene zur Darstellung von Schatten in Unreal Engine 5
Quelle: Darstellung von Epic Games, Inc. [32]

5 Methodik

Dieses Kapitel konzentriert sich auf die Entwicklung einer Methodik zur Evaluierung der visuellen Qualität und Performance von Lichtberechnungen. Ziel sollte es sein, die beste Qualität mit der besten Performance zu erreichen. *For real-time interactive rendering, efficiency is paramount. A low-quality animation that is interactive almost always leads to a better experience in a virtual world than a high-quality one with limited or high-latency interaction* [8, S. 323].

5.1 Messung der Leistung von Echtzeit-Render-Engines

Im Rahmen dieser Bewertung wird auch die Hardware berücksichtigt, mit der die Tests durchgeführt werden, da sie einen bedeutenden Einfluss auf die Leistung hat.

Das Testsystem besteht aus:

- Prozessor: AMD Ryzen 3950X
- Grafikkarte: Nvidia Geforce RTX 2070 Super 8 Gigabyte VRAM (Treiberversion: 528.24)
- Arbeitsspeicher: 64 GB DDR4 3200 MHz
- Betriebssystem: Windows 11 (Version 22H2, Build 22621.1992)

Die Test wurden bei folgenden Versionen der Render-Engine durchgeführt:

- Blender Version: 3.6.1
- Unreal Engine 5 Version: 5.2.1

5.1.1 Messung FPS

In der Unreal Engine 5 ist die Messung der FPS bereits integriert und kann direkt innerhalb der Anwendung abgelesen werden. Im Fall von Blender hingegen wird die FPS mithilfe eines Python⁸-Scripts ermittelt [33].

5.1.2 Messung Latenz

In Blender ist es nicht direkt möglich, die Latenz auszulesen. Daher wird mithilfe eines eigenen Python-Scripts (Codeblock 1) die Zeit gemessen, die benötigt wird, um das Ansichtsfenster zu aktualisieren. Diese Methode bietet zwar keine exakte Messung der Latenz und spiegelt nicht vollständig die Erkenntnisse aus dem Kapitel 2.3 zur Definition von Latenz wider, dennoch stellt sie eine ausreichende Annäherung dar, um Rückschlüsse auf die Performance in EEVEE zu ziehen. In der Unreal Engine 5 ist die Latenzmessung integriert.

⁸ Python ist eine objektorientierte Programmiersprache und die Sprache, in der Blender programmiert wurde

Codeblock 1: Reaktionszeitskript.py
Quelle: Eigene Darstellung

```
import bpy
import time

# Function to measure viewport redraw time in milliseconds
def measure_viewport_redraw_time(engine_name):
    # Set the render engine
    bpy.context.scene.render.engine = engine_name

    # Measure the viewport redraw time in milliseconds
    start_time = time.time()
    bpy.ops.wm.redraw_timer(type='DRAW_WIN_SWAP', iterations=1)
    end_time = time.time()

    # Calculate the average viewport redraw time in milliseconds
    avg_redraw_time_ms = (end_time - start_time)

    return avg_redraw_time_ms

if __name__ == "__main__":
    redraw_time_cycles = measure_viewport_redraw_time('CYCLES')
    redraw_time_eevee = measure_viewport_redraw_time('BLENDER_EEVEE')

print("viewport redraw time in milliseconds in Cycles:", redraw_time_cycles)
print("viewport redraw time in milliseconds in Eevee:", redraw_time_eevee)
```

5.1.3 Messung VRAM

Die Methode zur Messung des VRAM (Video Random Access Memory) unterscheidet sich zwischen den beiden Render-Engines. In Blender wird der gesamte VRAM-Verbrauch angezeigt, der zum Zeitpunkt der Render-Engine-Nutzung im gesamten System auftritt. Dies schließt jedoch auch alle Hintergrundprozesse ein, die von der GPU verarbeitet werden und nichts mit Blender zu tun haben. Dies kann die Messergebnisse verfälschen. Um das zu vermeiden, wird ein Basiswert vor dem Öffnen der Render-Engine ermittelt und von den gemessenen Werten während der Testszenarien abgezogen. Auf diese Weise wird der tatsächliche VRAM-Verbrauch durch Blender isoliert. Im Gegensatz dazu bietet die Unreal Engine 5 das "Render Resource Viewer"-Tool [34], ein integriertes Werkzeug, das den VRAM-Verbrauch misst. Dieses Werkzeug ermöglicht eine präzise Erfassung und bietet somit eine zuverlässige Basis für die Evaluierung.

5.2 Visuelle Qualität

Eine Messbarkeit der visuellen Qualität lässt sich nicht quantifizieren [8, S. 322], weshalb es hierfür anderer Bewertungskriterien bedarf.

5.2.1 Vergleich zu Offline-Render-Engine

Eine naheliegende Vorgehensweise besteht darin, ein Bild aus der Echtzeit-Render-Engine mit dem Bild einer Offline-Render-Engine zu vergleichen. Diese Offline-Render-Engine bietet den Vorteil, dass sie mithilfe von Path Tracing die derzeit beste Annäherung zur Lösung der Rendergleichung ermöglicht. Zur Beurteilung der visuellen Qualität wird deshalb Cycles als Benchmark⁹ verwendet. Durch die Verwendung von Cycles als Standard für visuelle Qualität wird die Basis geschaffen, um Echtzeit-Render-Engines hinsichtlich ihrer visuellen Performance und Annäherung an die Rendergleichung zu evaluieren. Cycles wird mit 20000 Abtastungen und 32 Lichtpfaden bei allen Materialarten verwendet, um das bestmögliche Ergebnis zu produzieren, allerdings wird auf denoising verzichtet, damit keine Details verloren gehen.

Für Unreal Engine 5 wird der Befehl "highresshot" mit der entsprechenden Auflösung ausgeführt, um eine Momentaufnahme der Szene zu machen und die visuelle Qualität beurteilen zu können. In Blender kann die Momentaufnahme über den Menüpunkt "View → Viewport Render Image" dokumentiert werden. Hierdurch ist ein Vergleich in Echtzeit möglich.

5.2.2 Evaluierung im Anwendungsfenster

Im Kapitel 2.3 wurde beschrieben, dass Echtzeit-Render-Engines besonders die Interaktivität in den Vordergrund stellen. Infolgedessen werden die durchgeführten Testszenarien innerhalb des Ansichtsfensters der jeweiligen Render-Engine bewertet. Um ein optimales Maß an Vergleichbarkeit zu gewährleisten, wurde eine Auflösung von 1920x1080 Pixel festgelegt.

In Unreal Engine 5 kann die exakte Auflösung des Ansichtsfensters direkt aus dem sogenannten "Viewport Log" entnommen werden. In Blender erfordert die Ermittlung der Auflösung eine externe Render-Engine zur Messung der Pixel, um die ausgewählte Auflösung zu erreichen.

Zusätzlich wird eine Kamera innerhalb einer iterativen Schleife animiert. Auf diese Weise wird jedem einzelnen Bild nur die minimal mögliche Berechnungszeit zugewiesen, um in dieser Zeit die bestmögliche Qualität zu liefern.

⁹ Ein Benchmark ist ein Referenzwert zur Messung von Leistung oder Qualität einer Render-Engine oder Hardware

5.3 Testszzenarien

Die Testszzenarien bilden die Grundlage für die Evaluierung der Engines. Sie werden entwickelt, um verschiedene Aspekte der visuellen Qualität und Performance abzudecken.

5.3.1 TestszENARIO 1

Das erste TestszENARIO (Abbildung 5.1) zielt darauf ab, die visuelle Qualität im Vergleich zur Offline-Render-Engine Cycles zu bewerten. Hier werden wir uns auf Aspekte, wie die Darstellung von Beleuchtung, Schatten und Materialien konzentrieren. Um eine einfache Vergleichbarkeit zu erreichen, wird hierbei ein einfaches TestszENARIO in Form einer Cornell-Box [25] verwendet.

Das TestszENARIO besteht aus folgenden Elementen:

- (1) Kugel mit reflektierender Oberfläche
- (2) Würfel mit diffuser Oberfläche
- (3) Quader mit diffus gestreuter Oberfläche
- (4) Affenkopf mit Subsurface Scattering Oberfläche
- (5) Zylinder mit hohlem Innenraum und Brechungsindex 1,45
- (6) Kugel mit metallener, reflektierender Oberfläche
- (7) Kugel mit Brechungsindex von 1,45

Die Umgebung besteht aus einer roten und grünen Wand zur Beurteilung von indirekter Beleuchtung. Die Beleuchtung erfolgt durch eine Flächenlichtquelle von oben, um die Möglichkeiten zur Erzeugung von Übergangsschatten zu testen.

Mit diesen Materialeigenschaften soll ein Großteil der Eigenschaften von PBS hinsichtlich Transmission, Reflexion und Absorption überprüft werden.

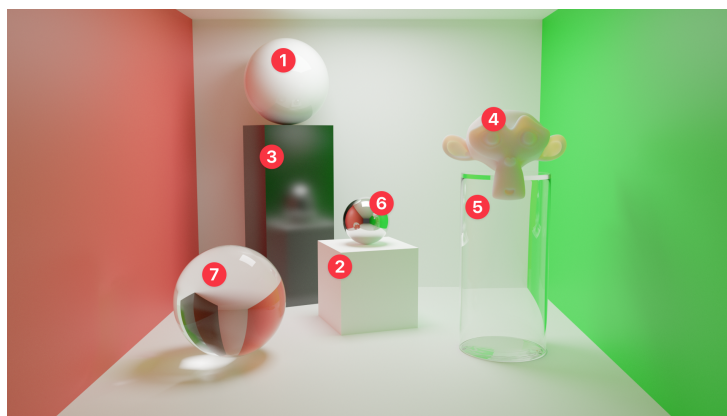


Abbildung 5.1: Beschreibung des Aufbaus von TestszENARIO 1 (Cornell Box)
Quelle: Eigene Darstellung

5.3.2 TestszENARIO 2

Im zweiten TestszENARIO liegt der Fokus auf der Performance der Echtzeit-Render-Engines. Hier werden vorrangig die Bildraten (FPS), Videospeicher und Latenz der Render-Engine analysiert. Um die Performance besser bestimmen zu können, wird eine komplexe Szene in Form einer Architekturvisualisierung von Park3D [35] (Abbildung 5.2) verwendet. Diese besteht aus vielen verschiedenen Oberflächenstrukturen und 67 Lichtquellen. Diese setzen sich zusammen aus einer Mischung von Flächen, Punkt-, Spotlichter, sowie eines HDRI Umgebungslichts und einer Richtungslichtquelle.



Abbildung 5.2: Bilder des Aufbaus von TestszENARIO 2
Quelle: Eigene Darstellung

5.4 Vergleichbarkeit von Messgrößen

Beide TestszENARIEN wurden mit Blender erstellt. Während der Übertragung der TestszENARIEN von Blender zur Unreal Engine 5 haben sich verschiedene Herausforderungen und Problemstellungen ergeben. Diese sollen im folgenden Kapitel kurz beschrieben werden.

5.4.1 Export

Aufgrund des Fehlens eines einheitlichen Standards für die Übertragung, wurden 3 Exportformate, FBX¹⁰, USD¹¹ und Datasmith¹² (Abbildung 5.3) getestet. Entscheidend für das Importieren in Unreal Engine 5 war eine geringe Fehlerquote hinsichtlich folgender Parameter:

- Skalierung
- Kameradaten und Animationen
- Lichtquellen
- PBS-Texturen und Materialeigenschaften

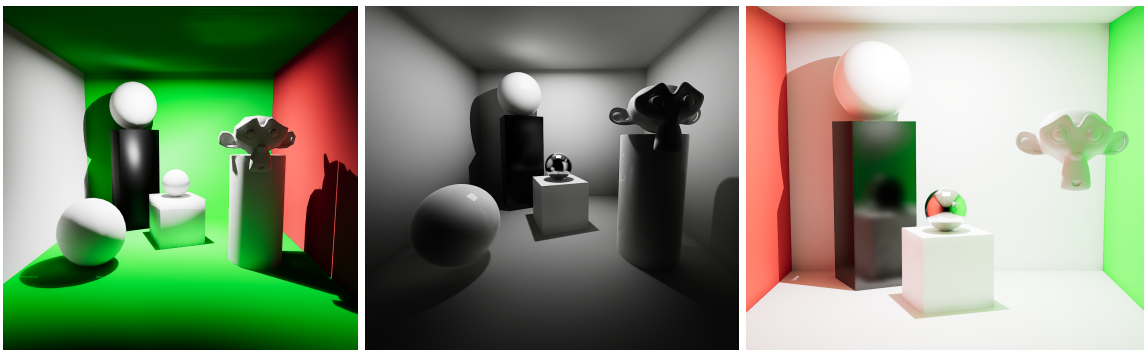


Abbildung 5.3: Darstellung der Testszenarien: FBX (links), USD (mitte), Datasmith (rechts)
Quelle: Eigene Darstellung

Blender bietet keine direkte Exportfunktion von Datasmith, deshalb wurde ein Add-on [36] verwendet. Bei der Übertragung ergab sich jedoch eine weitere Herausforderung, die die Entscheidung erschwerte: Beim Testen der drei Exportformate variierte die Helligkeit des Testszenarios erheblich. Dies galt es zu untersuchen.

5.4.2 Lumen oder Watt

Blender verwendet radiometrische Größen [37] und Unreal Engine 5 verwendet photometrische Größen [38]. Um eine “Übersetzung” von der photometrischen Größe zur radiometrischen Größe zu machen, muss die Lichtquelle in Blender mit dem spektralen photometrischen Strahlungsäquivalent (Abbildung 5.4) gewichtet werden.

¹⁰ FBX ist ein Dateiformat von Autodesk, dass in beiden Programmen als Import und Exportmöglichkeit zur Verfügung steht

¹¹ USD steht für Universal Scene Description und ist ein von Pixar entwickelte Technologie, die speziell zum Austauschen von 3D Inhalten zwischen verschiedenen Render-Engine Lösungen entwickelt wurde

¹² Datasmith ist eine von Unreal Engine entwickelte Technologie, die das Importieren von verschiedenen Render-Engine ermöglicht

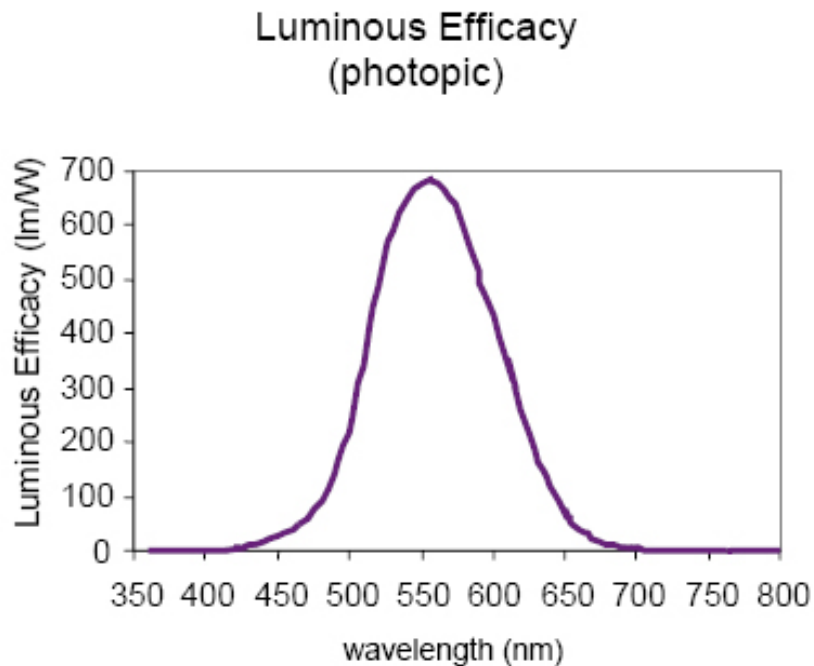


Abbildung 5.4: photometrische Strahlungsäquivalenzkurve
Quelle: Darstellung von Angelo V. et. al. [39]

5.4.2.1 Lösungsansatz 1 Einsatz von IES Lights

IES ("Illuminating Engineering Society") ist ein Dateiformat, das die Lichtverteilung und Intensität von Leuchten beschreibt. In der Theorie sind IES-Lichter für die Generierung von fotorealistischen Bildern sinnvoll, da sie die Daten einer realen Lichtverteilung beinhalten.

Codeblock 2: Metadaten IES Licht
Quelle: IES Library [40]

```
IESNA:LM-63-1995
[TEST]
[MANUFAC] BEGA
[MORE] Copyright LUMCat V
[LUMCAT]
[LUMINAIRE] 50849.2K3
[LAMPCAT] LED 8,5W
[LAMP]      884 lm, 10 W
```

In der Praxis ergeben sich aber mehrere Probleme. Zwar erzeugen IES-Lichter mit ihrem unterschiedlichen Lichtverteilung realistische Ergebnisse, dennoch existiert auch hier kein einheitlicher Standard für die Interpretation der Metadaten (siehe Codeblock 2). In den Metadaten ist der Stromverbrauch der Lichtquelle in Watt angegeben. Blender verwendet den Begriff "Watt" nicht, um die Energiemenge anzugeben, die ein System verbraucht, sondern vielmehr die Menge der von einer Lichtquelle emittierten Strahlung [37]. Dadurch gestaltet es sich schwierig, die im obigen Beispiel genannten 10 Watt in Blender korrekt darzustellen.

Blender beschreibt in der Dokumentation des Programmcodes folgendes:

Codeblock 3: Auszug Blender Programmcode

Quelle: Blender Programmcode [41]

```
/* Intensity values in IES files are specified in candela  
(lumen/sr), a photometric quantity.  
Cycles expects radiometric quantities, though, which requires  
a conversion.  
However, the Luminous efficacy (ratio of lumens per Watt)  
depends on the spectral distribution  
of the light source since lumens take human perception into  
account.  
Since this spectral distribution is not known from the IES  
file, a typical one must be  
assumed. The D65 standard illuminant has a Luminous efficacy  
of 177.83, which is used here to  
convert to Watt/sr. Also, the Watt/sr value must be multiplied  
by 4*pi to get the Watt value that  
Cycles expects for lamp strength. Therefore, the conversion  
here uses  $4\pi/177.83$  as a Candela  
to Watt factor.  
  
*/
```

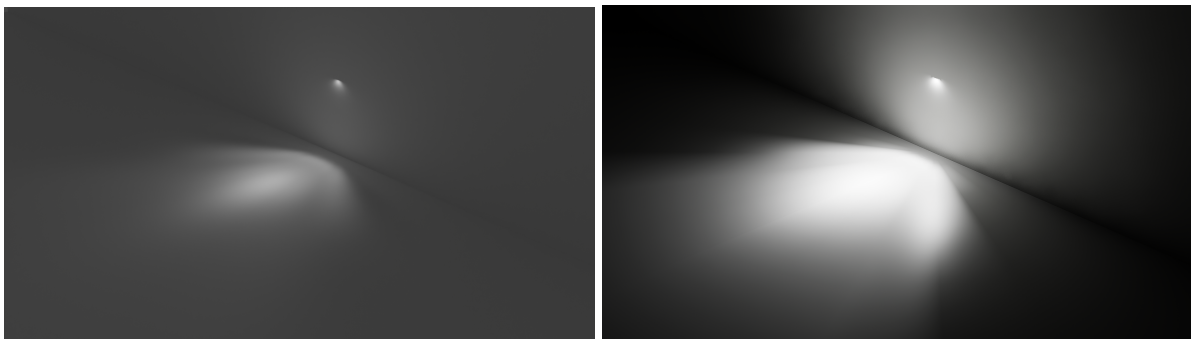


Abbildung 5.5: IES Licht in Blender (links) und in Unreal Engine 5 (rechts)

Quelle: Eigene Darstellung

Blender gewichtet IES Lichter mit einem Faktor von 0.0706 um den Lichtstrom zu bestimmen, wohingegen Unreal Engine 5 den Wert für den Lichtstrom direkt aus den Metadaten übernimmt. Die Helligkeitswerte stimmen somit nicht überein (Abbildung 5.5).

Außerdem werden IES Lights nicht von EEVEE nativ unterstützt, sodass eine Vergleichbarkeit nicht nur zwischen Unreal Engine 5 und EEVEE schwer fällt, sondern auch innerhalb Blenders zwischen Cycles und EEVEE.

5.4.2.2 Lösungsansatz 2 Mathematische Umrechnung realer Werte

Die zweite Möglichkeit wäre der mathematische Ansatz zur Berechnung von Lumen in Watt. Dafür muss der das photometrische Strahlungsäquivalent für Blender oder Unreal Engine 5 für die verschiedenen Lichtquellen (Punkt-, Spot- oder Flächenlichtquelle) bestimmt werden mithilfe folgender Gleichung:

$$K = \frac{\Phi_v}{\Phi_e} = \frac{\int_0^\infty K(\lambda) \Phi_{e,\lambda} d\lambda}{\int_0^\infty \Phi_{e,\lambda} d\lambda} \quad (5)$$

- K ist das Strahlungsäquivalent;
- Φ_v ist der Lichtstrom in Lumen;
- Φ_e ist der Strahlungsfluss in Watt;
- $K(\lambda)$, wird oft auch $\bar{y}(\lambda)$ beschrieben, ist das photometrische Strahlungsäquivalent in Abhängigkeit von der Wellenlänge, dimensionslos;
- $\Phi_{e,\lambda}(\lambda)$ ist der Strahlungsfluss in Abhängigkeit von der Wellenlänge in Watt pro Nanometer;
- λ ist die Wellenlänge in Nanometer;

Folglich erfordert eine präzise Umrechnung der Lichtquellen zwischen Unreal Engine 5 und Blender die Kenntnis des Strahlungsäquivalents über den gesamten Wellenlängenbereich. Diese Informationen sind für keine der beiden Render-Engine definiert. Dieser Umstand ist hauptsächlich darauf zurückzuführen, dass sowohl Radiometrie als auch Photometrie in beiden verwendeten Systemen autonom agieren können. Daher besteht weder in Blender noch der Unreal Engine 5, abgesehen von der IES-Umrechnung, eine zwingende Notwendigkeit für weitere Konvertierungen.

Jedoch gibt es eine alternative Herangehensweise. Anstatt die Werte direkt von Blender in Unreal Engine 5 zu übertragen, können reale Referenzwerte genutzt werden, um eine Umrechnung zu ermöglichen. Blender bedient sich einer eigenen Tabelle zur Umrechnung von realen Werten. Gemäß dieser Tabelle entspricht beispielsweise eine Punktlichtquelle mit einer Leistung von 2,1 Watt einer realen LED-Lichtquelle mit 800 Lumen. Falls wir derartige Werte in die Unreal Engine 5 übertragen würden, ergäbe sich ein ähnlicher Unterschied in der Helligkeit wie bei der Umrechnung von IES-Lichtquellen (siehe Abbildung 5.5). Indessen geht aus dieser Tabelle auch hervor, dass keine allgemeingültige Übersetzung möglich ist, da die spektrale Zusammensetzung der in der Tabelle aufgeführten Lichtquellen nicht angegeben wird.

Tabelle 1: Umrechnungstabelle von realen Werten in Blender
 Quelle: Blender Wiki [37]

Real world light	Power	Suggested Light Type
Candle	0.05 W	Point
800 lm LED bulb	2.1 W	Point
1000 lm light bulb	2.9 W	Point
1500 lm PAR38 floodlight	4 W	Area, Disk
2500 lm fluorescent tube	4.5 W	Area, Rectangle
5000 lm car headlight	22 W	Spot, size 125 degrees

5.4.2.3 Lösungsansatz 3 Vergleichsmethode

Das primäre Ziel dieser Analyse liegt in der Schaffung einer Vergleichsbasis zwischen den beiden Render-Engine. Die Nutzung von realen Berechnungen zur Ermittlung dieser Vergleichsbasis erwies sich als unpraktikabel. Daher wurde eine alternative, einfacher gestaltete Testmethode entwickelt, welche eine möglichst präzise Vergleichbarkeit der Resultate anstrebt.

In beiden Render-Engines wurde eine Punktlichtquelle auf gleicher Höhe auf eine ebene Fläche gerichtet, während eine Kamera in identischer Entfernung positioniert wurde. Anschließend erfolgte der Export eines Bildes aus beiden Render-Engines. In einem Bildbearbeitungsprogramm wurde dann die Differenz zwischen den beiden Bildern gebildet und die Helligkeitswerte so lange angepasst, bis das Resultat weitgehend übereinstimmte und ein nahezu schwarzes Bild ergab. Dies bedeutet, dass alle Pixelwerte in den beiden Bildern exakt übereinstimmen und somit keine Unterschiede zwischen den Bildern bestehen. Diese Vorgehensweise führte zu dem Ergebnis, dass ein Lumen ungefähr einem Watt entspricht. Dieses Lumen-zu-Watt-Verhältnis wurde auch in der automatischen Datasmith-Übertragung berücksichtigt, wodurch weitere Umrechnungen entfallen konnten.

Darüber hinaus war es erforderlich, sämtliche Lichtkompensationen in der Unreal Engine 5 zu deaktivieren, um eine verlässliche Vergleichbarkeit der Ergebnisse zu erzielen (siehe Abbildung 5.6). Die Unreal Engine 5 gleicht standardmäßig die Helligkeit von Lichtquellen anhand der gemessenen Luminanz des aktuellen Bildes aus. Wenngleich aus künstlerischer Perspektive lediglich eine relative Anpassung der Helligkeit notwendig ist, um eine gewünschte Wirkung zu erzielen, ist diese Art der Kompensation für wissenschaftliche Vergleichszwecke ungeeignet und wurde deshalb deaktiviert.

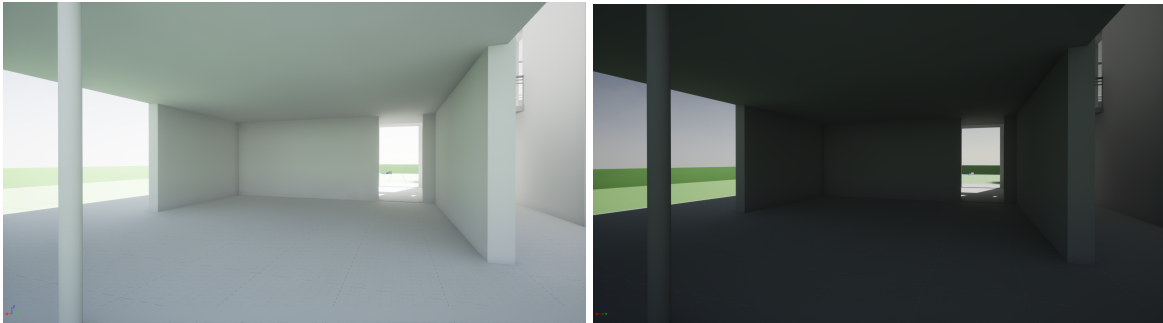


Abbildung 5.6: Lichtkompensation in Unreal Engine an (links), aus (rechts)
Quelle: Eigene Darstellung

Mit der Bewältigung dieser Herausforderungen durch passende Lösungen kann nun die Evaluierung der Ergebnisse erfolgen.

6 Evaluierung visuelle Qualität

Durch das nachfolgende Testszenario sollen Erkenntnisse darüber gewonnen werden, welche Techniken zur besten visuellen Qualität führen. Für den Vergleich von Reflexion, Brechung und Subsurface Scattering werden nachfolgend jeweils Ausschnitte der Ergebnisse von Abbildung 6.1, 6.2, 6.3 des Testszenarios verwendet.

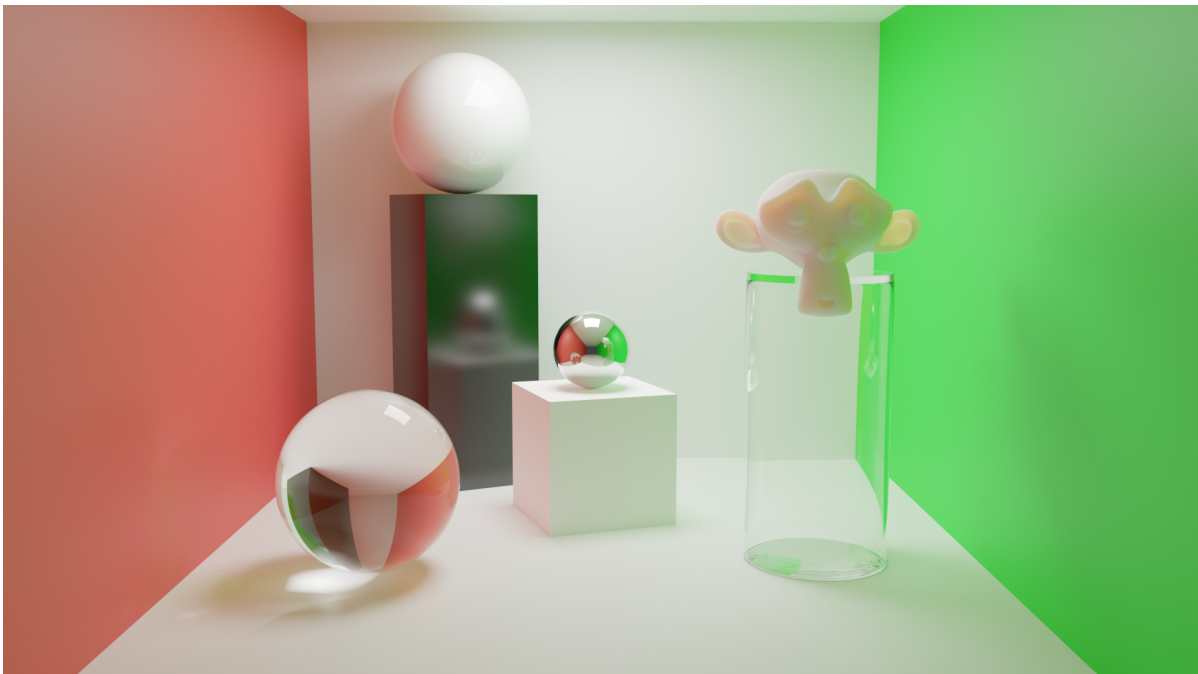


Abbildung 6.1: Testszenario 1 in Cycles
Quelle: Eigene Darstellung

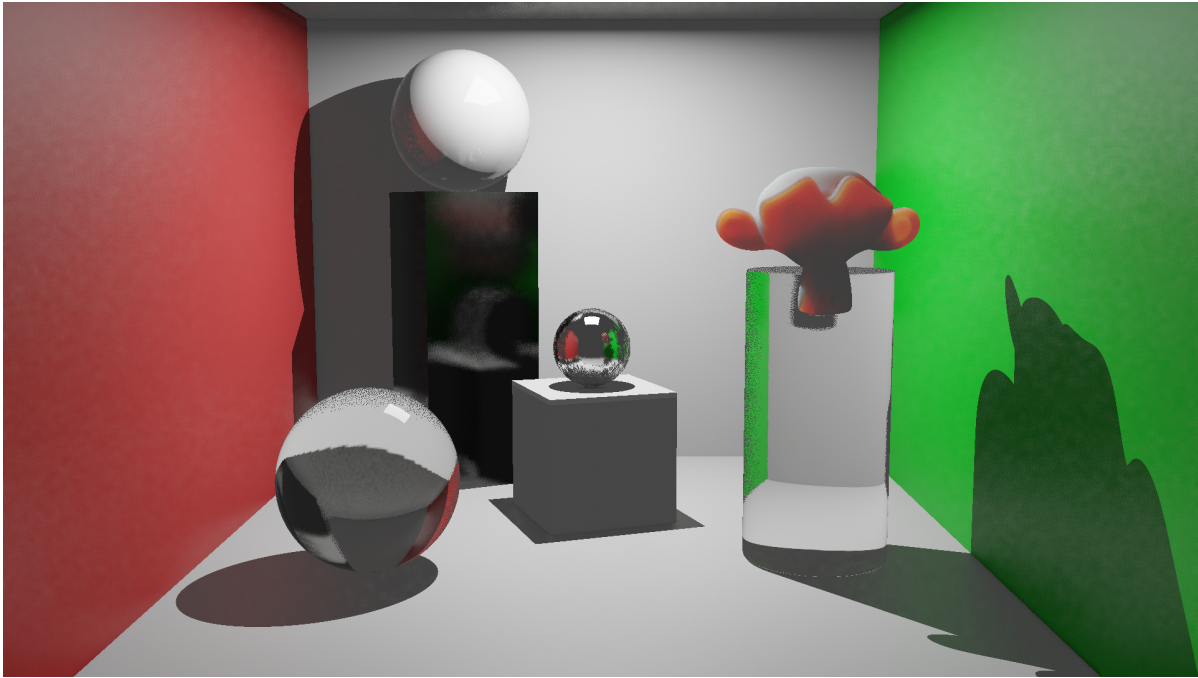


Abbildung 6.2: TestszENARIO 1 in EEVEE
Quelle: Eigene Darstellung



Abbildung 6.3: TestszENARIO 1 in Unreal Engine 5
Quelle: Eigene Darstellung

6.1 Reflexionen

6.1.1 Diffuse Reflexionen

Bei diffusen Reflexionen lassen sich keine visuellen Unterschiede zwischen den 3 Render-Engines erkennen (Abbildung 6.4). Die Betrachtung von indirekter Beleuchtung und Schatten wurde an dieser Stelle außer Acht gelassen.

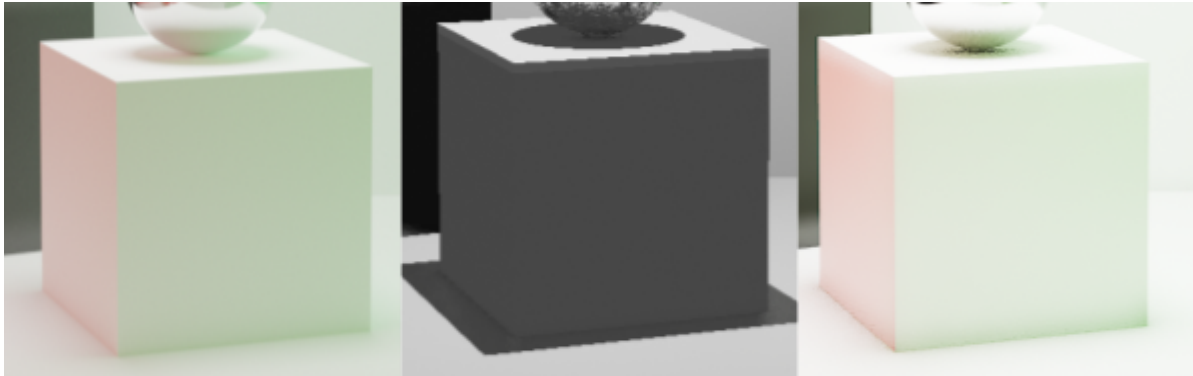


Abbildung 6.4: Darstellung diffuser Würfel in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)
Quelle: Eigene Darstellung

6.1.2 Spiegelnde Reflexionen

Bei den spiegelnden und gerichteten Reflexionen sind starke Unterschiede zu erkennen (Abbildung 6.5). Hierbei sind in Eevee deutliche Aliasing-Artefakte zu erkennen. Die Reflexionen sind nur im Ansatz zu erkennen. Aufgrund der Verwendung von SSR verschwindet die Decke in den Reflexionen. Unreal Engine 5 und Cycles zeigen nahezu identische Ergebnisse. Cycles schafft es jedoch, gläserne Objekte in Reflexionen darzustellen.



Abbildung 6.5: Vergrößerte Darstellung spiegelnder Reflexionen in Cycles (links oben), Eevee (rechts oben), Unreal Engine 5 (unten)
 Quelle: Eigene Darstellung

6.1.3 Gestreute Reflexionen

Bei der Darstellung von gestreuten Reflexionen sind deutliche Unterschiede festzustellen. (Abbildung 6.6). Hier treten bei Eevee Verzerrungen in den Reflexionen auf. Unreal Engine 5 zeigt wieder eine ähnliche Darstellung wie Cycles. Allerdings ist die Reflexion der Metallkugel dunkler als bei Cycles

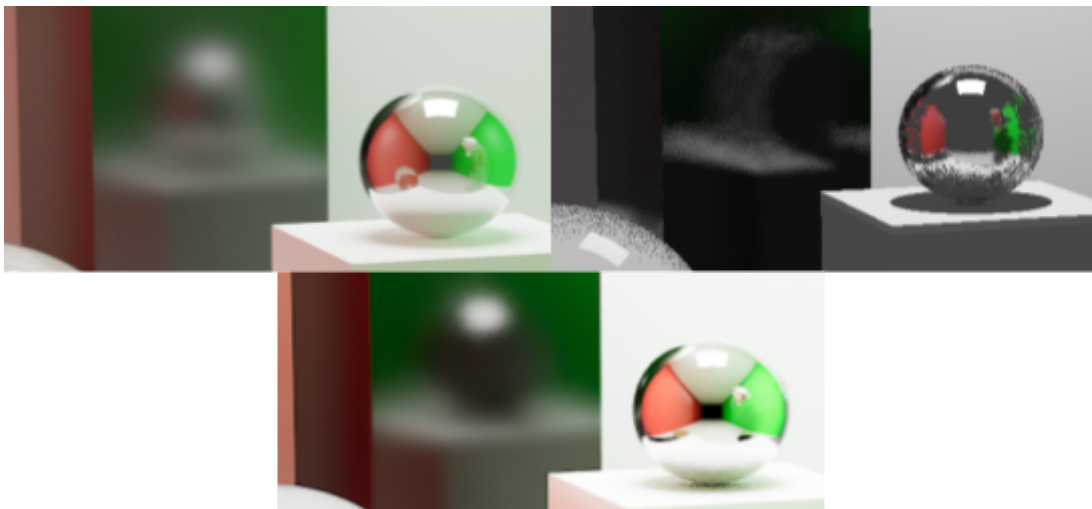


Abbildung 6.6: Vergrößerte Darstellung gestreuter Reflexionen in Cycles (links), Eevee (rechts), Unreal Engine 5 (unten)
 Quelle: Eigene Darstellung

6.1.4 Brechung und Reflexion

Die Glaskugel in Abbildung 6.7 weist in EEVEE Aliasing Effekte in den Schatten auf. Außerdem ist die Darstellung des Brechungsindex fehlerhaft. Unreal Engines 5 und Cycles stellen zumindest den gleichen Brechungsindex dar. Jedoch verliert Unreal Engine 5 Details in den Schatten der Brechung. Das äußert sich in "Abdunkelungseffekten" und Bildrauschen.

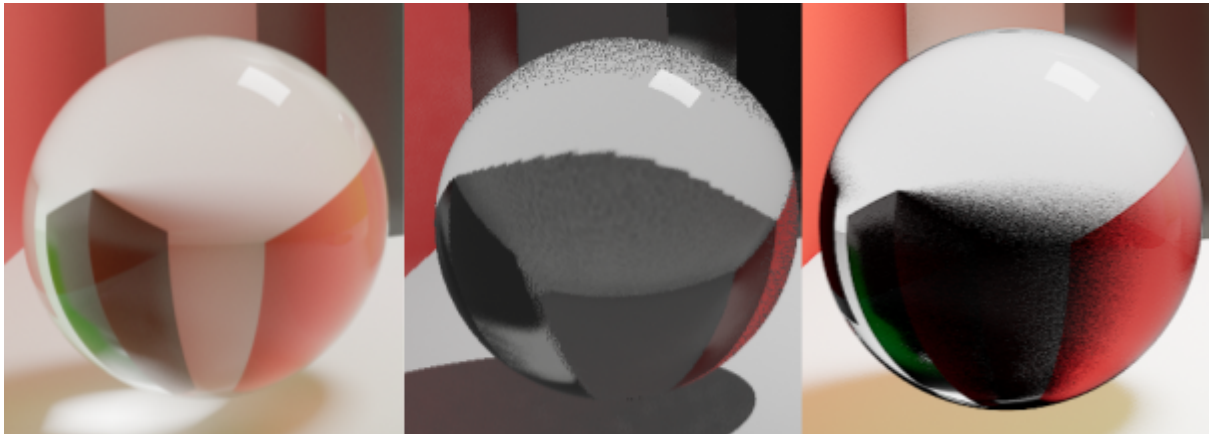


Abbildung 6.7: Vergrößerte Darstellung der Glaskugel in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)
Quelle: Eigene Darstellung

Noch mehr leidet die visuelle Qualität bei Unreal Engine 5 bei der Darstellung des gläsernen Zylinders (Abbildung 6.8). Auch, wenn die Brechung realistisch ist, ist das Glas nicht realistisch dargestellt. Der Boden des Zylinders verschwindet vollständig. EEVEE ignoriert den Fakt, dass der Innenraum des Zylinders keine Brechung aufweisen sollte und stellt das Objekt wie die Glaskugel dar.

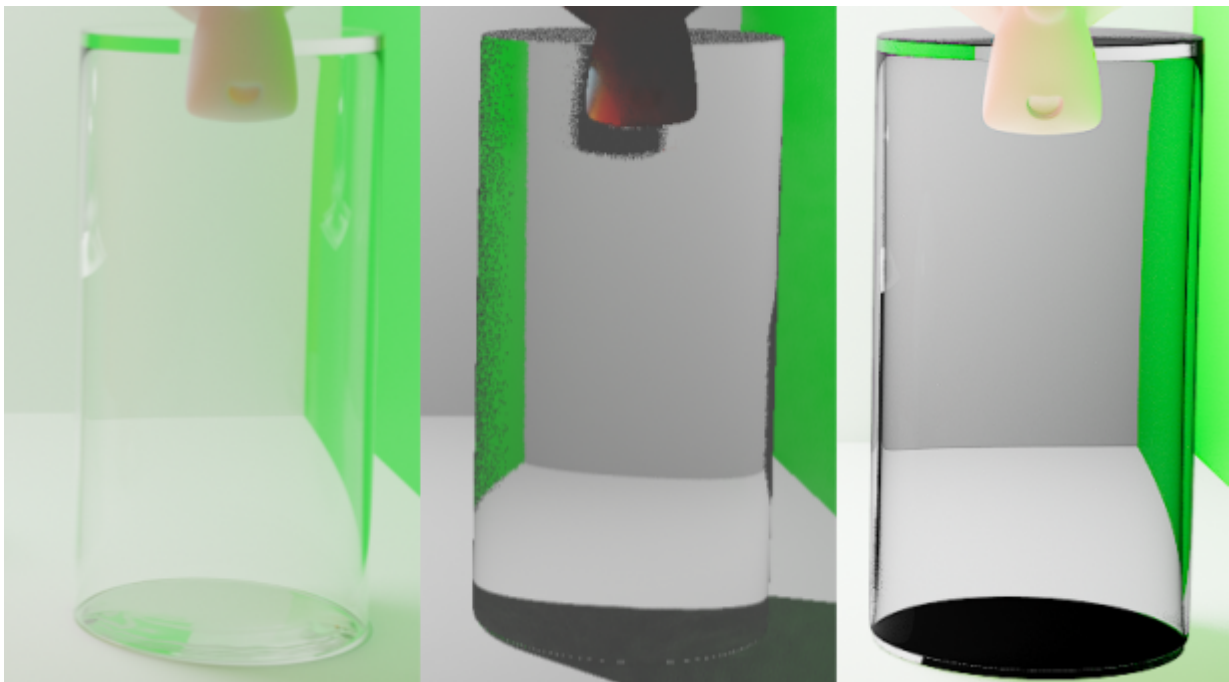


Abbildung 6.8: Vergrößerte Darstellung des Glaszylinders in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)
Quelle: Eigene Darstellung

6.1.5 Brechung, Absorption, Reflexion

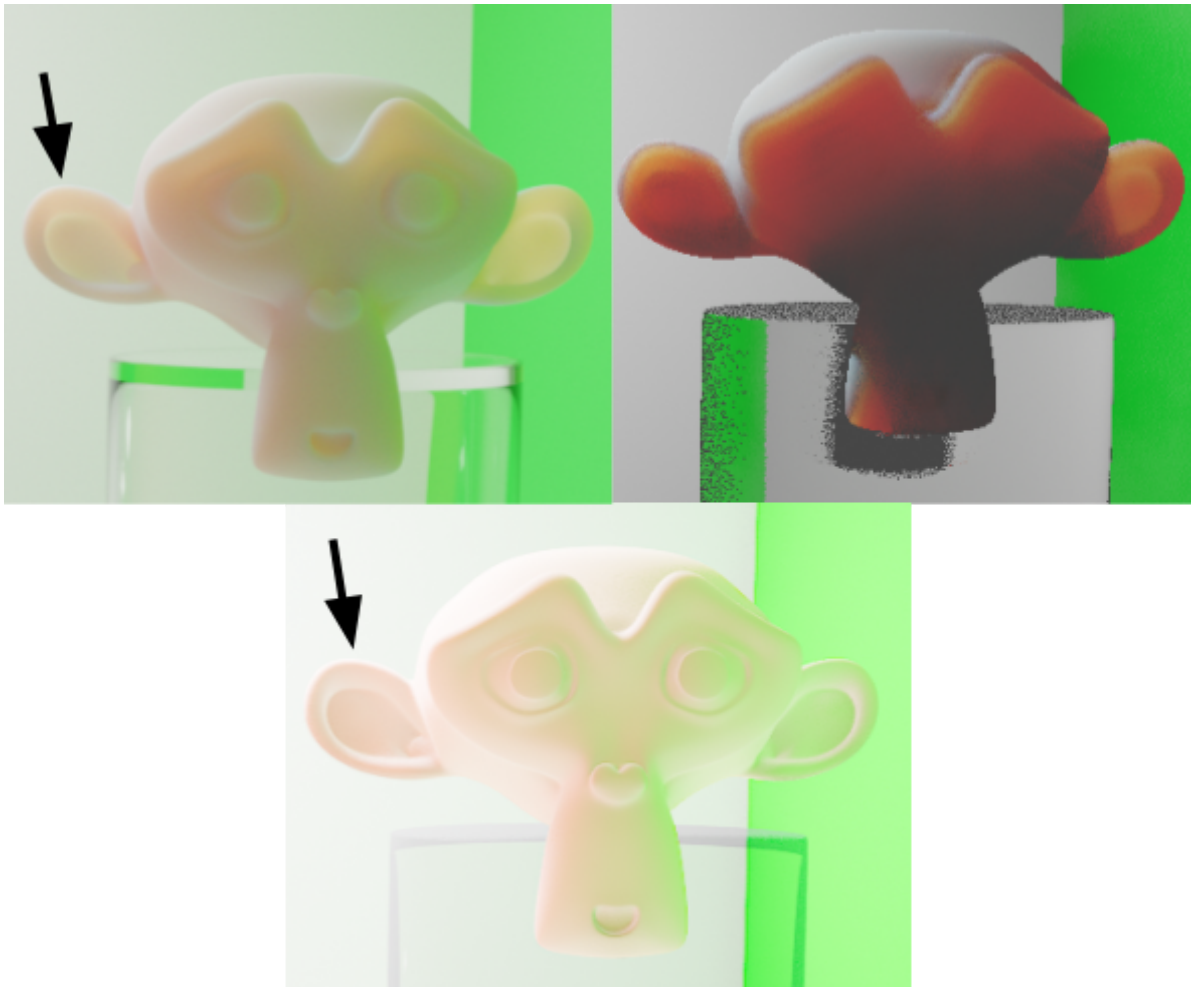


Abbildung 6.9: Vergrößerte Darstellung vom Subsurface Scattering Affenkopf in Cycles (links), Eevee (rechts), Unreal Engine 5 (unten)
Quelle: Eigene Darstellung

Das Durchscheinen des Lichts an dünnen Stellen des Modells kann weder in Unreal Engine 5, noch Eevee vergleichbar mit Cycles dargestellt werden (Abbildung 6.9). Obwohl das Ergebnis in der Unreal Engine 5 auf den ersten Blick ähnlich dem Ergebnis von Cycles erscheinen mag, ist diese Ähnlichkeit auf die indirekte Beleuchtung zurückzuführen und nicht auf die Methode zum Subsurface Scattering. Besonders deutlich wird dies am linken Ohr, wo der obere Teil nicht lichtdurchlässiger dargestellt wird, als der untere.

6.2 Schatten

EEVEE kann keine Übergangsschatten, sondern nur Kernschatten in Echtzeit abbilden (siehe Abbildung 6.10).

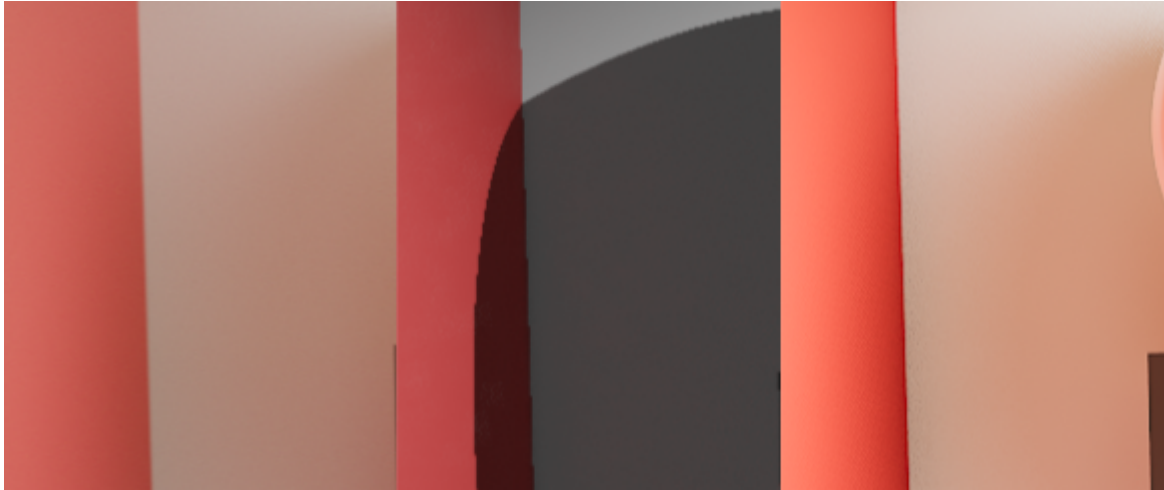


Abbildung 6.10: Vergrößerte Darstellung von Schatteneffekten in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)
Quelle: Eigene Darstellung

Unreal Engines 5 Raytraced Shadows liefern Ergebnisse, die sich nur durch das Bildrauschen von Cycles erheblich unterscheiden (Abbildung 6.11). Zusätzlich fehlen die Abbildungen von Lichtphänomenen in Form von Kaustiks (Abbildung 6.12).

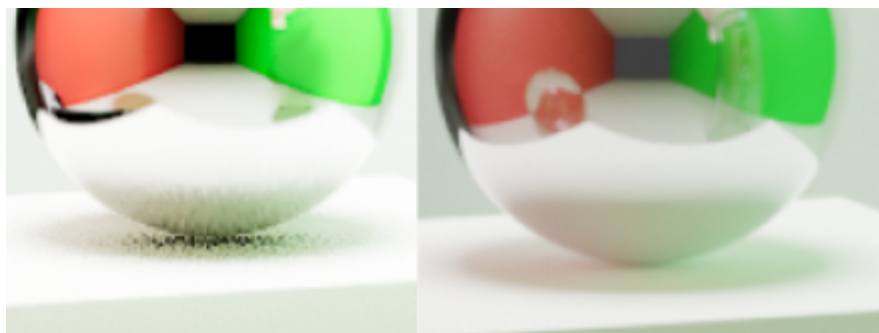


Abbildung 6.11: Vergrößerte Darstellung von Bildrauschen in Unreal Engine 5 (links), Cycles (rechts)
Quelle: Eigene Darstellung

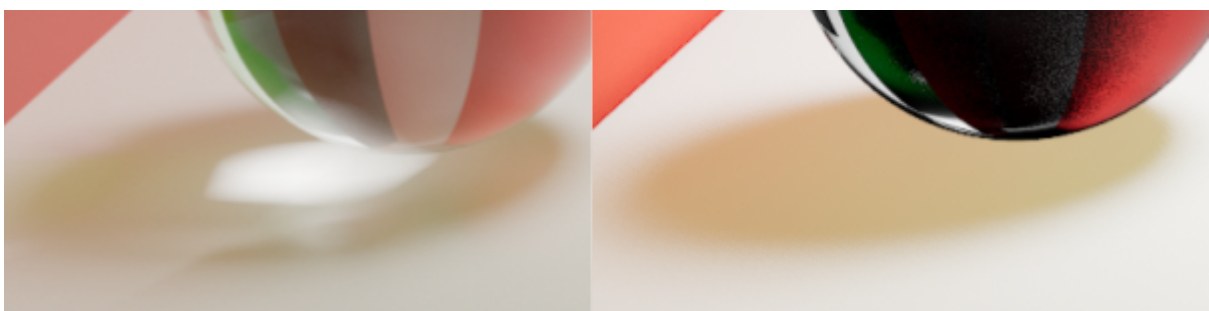


Abbildung 6.12: Vergrößerte Darstellung von Kaustiks in Cycles (links), Unreal Engine 5 (rechts)
Quelle: Eigene Darstellung

6.3 Global Illumination

Bei der Global Illumination zeigt sich der größte Unterschied in der visuellen Qualität zwischen Eevee und Unreal Engine 5. Am deutlichsten ist das bei der Farbübertragung der roten und grünen Wände zu erkennen (Abbildung 6.13), die sich selbst in den Schatten Bereichen zeigen.

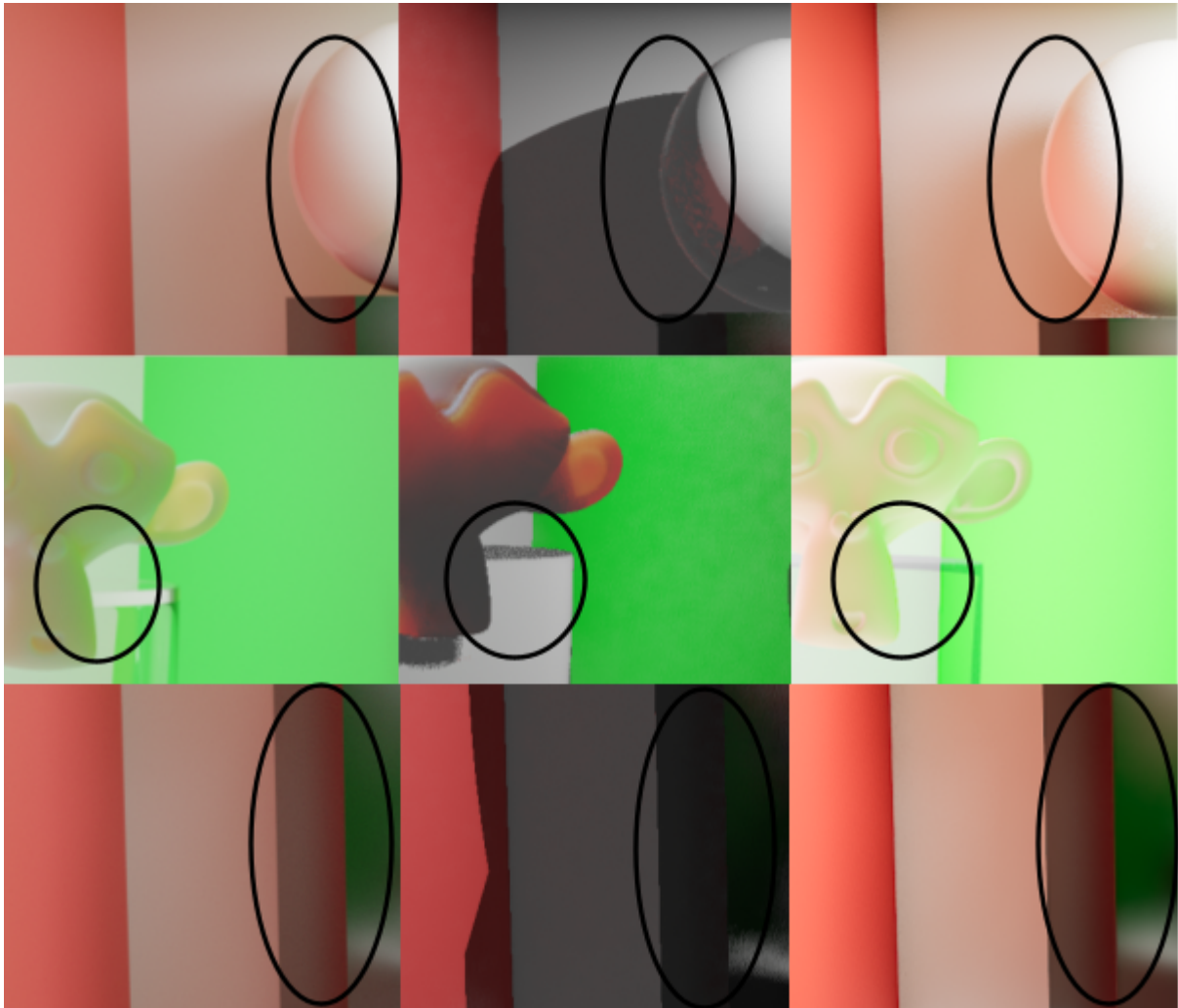


Abbildung 6.13: Vergrößerte Darstellung von Global Illumination in Cycles (links), Eevee (Mitte), Unreal Engine 5 (rechts)

Quelle: Eigene Darstellung

7 Evaluierung Performance

Es wird deutlich, dass die visuelle Qualität in Unreal Engine 5 und EEVEE mit zunehmender Komplexität der Materialeigenschaften abnimmt. Um die Auswirkungen dieser Komplexität auf die Echtzeitberechnung der verschiedenen Materialeigenschaften zu überprüfen, wurden einzelne Objekte vom Rendern ausgeschlossen, um festzustellen, welche Materialeigenschaft in der Berechnung am anspruchsvollsten ist.

Tabelle 2: Tabelle zum Performance Vergleich deaktivierter Materialien, mehr FPS entsprechen einer komplexeren Darstellung

Quelle: Eigene Darstellung

sichtbare Objekte	FPS in Unreal Engine 5	FPS in EEVEE
alle aktiviert	53 FPS	151 FPS
reflektierende Kugel (1)	54 FPS	158 FPS
Diffuser Würfel (2)	53 FPS	152 FPS
reflektierender Quader (3)	54 FPS	155 FPS
SSS Affenkopf (4)	55 FPS	159 FPS
Transparenter Zylinder (5)	58 FPS	170 FPS
Metallkugel (6)	54 FPS	157 FPS
Glaskugel (7)	58 FPS	168 FPS

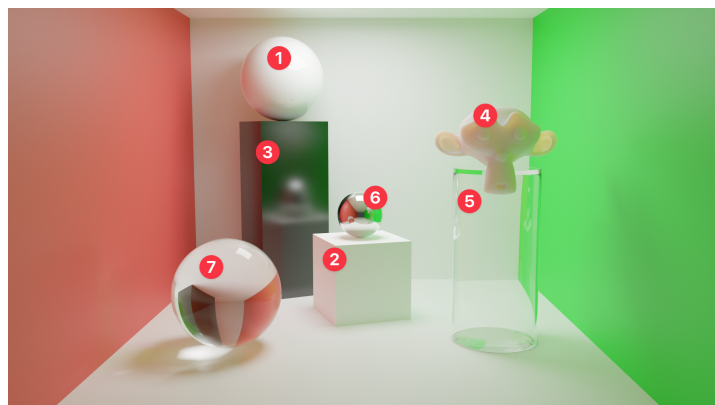


Abbildung 7.1: Cornell Box als Referenzbild bezüglich der Materialeigenschaften

Quelle: Eigene Darstellung

Der größte Performanceverlust in Unreal Engine 5 und EEVEE zeigt sich für Objekte mit Brechung, wodurch 5 FPS in Unreal Engine 5 und ca. 17 FPS pro Objekt mehr erreicht werden können, wenn auf

die Berechnung von Brechungen verzichtet wird. In Situationen, in denen mehrere solcher Objekte vorhanden sind, würden diese Unterschiede stärker in Erscheinung treten. Es ist bemerkenswert, dass trotz der Verwendung unterschiedlicher Techniken beide Programme vor ähnlichen Herausforderungen bei der Darstellung von Materialeigenschaften stehen.

Im Rahmen von Testszenario 1 betrug die durchschnittliche Bildrate in EEVEE 123 FPS, während die Unreal Engine 5 bei 57 FPS lag. Hier zeigt sich ein deutlicher Performance-Vorteil von EEVEE.

In Testszenario 2 zeigte sich ein ähnliches Bild. Hier erreichte die Unreal Engine 5 eine Bildrate von 9 FPS, während EEVEE 34 FPS erreichte. In Unreal Engine 5 lag die Bildrate unterhalb der für sinnvolle Echtzeitanwendungen erforderlichen Geschwindigkeit.

Es gab sowohl zeitliche als auch ressourcenabhängige Aspekte, die die Performance in Testszenario 2 stärker beeinflussten als in Testszenario 1.

Eine detaillierte Analyse der Materialeigenschaften sowie der direkten und indirekten Beleuchtung ist in diesem Zusammenhang von Bedeutung, da diese Systeme eng miteinander verknüpft sind und Einfluss auf die Performance haben. Alle Objekte weisen Reflexionseigenschaften auf (siehe Abbildung 7.2). Es sind jedoch keine Objekte mit Subsurface Scattering oder Brechung in der Szene vorhanden.

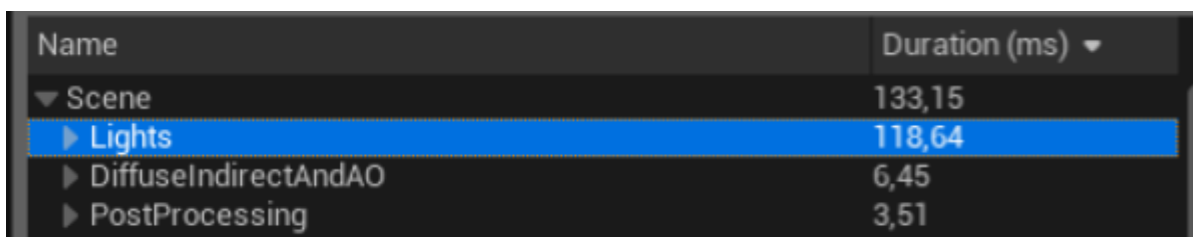


Abbildung 7.2: Alle reflektierenden Objekte mit ihren Reflexionseigenschaften in Testszenario 2
Quelle: Eigene Darstellung

Die Deaktivierung der Screen Space Reflections (SSR) in EEVEE zeigte nur geringfügige Verbesserungen in der Performance. Die Latenz lag vor der Deaktivierung bei durchschnittlich 120 ms und danach bei 112 ms. Ähnlich verhielt es sich mit der Deaktivierung der Ambient Occlusion, die ebenfalls keine signifikante Verbesserung der Performance bewirkt. Diese Beobachtungen sind in allen Räumen des Testszenarios gleich.

Bei EEVEE führte das Deaktivieren von sämtlichen Lichtquellen innerhalb des Raumes - außer der Sonne und indirekter Beleuchtung - durch HDRI zu einer deutlichen Verringerung der Latenz von 112 ms auf 23 ms und erhöhte die Bildrate von 34 FPS in allen Räumen auf 126 FPS. Grund hierfür könnte sein, dass Shadow Maps 67 Lichtquellen pro Pixel auf Schattierung überprüfen müssen.

In der Unreal Engine 5 zeigte sich ein ähnliches Bild, wobei auch hier die Anzahl der Lichtquellen den größten Einfluss auf die Performance hatte. Durch die Analyse der Latenz in der Unreal Engine 5 kann auch hier genau ermittelt werden, welche Elemente die Berechnung der Bilder am meisten beeinflussen. Von insgesamt 133 ms, die ein Frame benötigt, entfallen 118 ms allein auf die Schattenberechnungen (Abbildung 7.3 blau hervorgehoben).



Name	Duration (ms)
Scene	133,15
▶ Lights	118,64
▶ DiffuseIndirectAndAO	6,45
▶ PostProcessing	3,51

Abbildung 7.3: GPU Visualizer zeigt die zeitliche Zusammensetzung eines Bildes in Unreal Engine 5 an
Quelle: Eigene Darstellung

Die Zeitspanne, die für die Berechnung von Global Illumination aufgewendet wird (Abbildung 7.3), beträgt lediglich 6,45 ms und macht somit nur einen geringeren Anteil aus.

Für das Testszenario 2 beträgt der VRAM-Verbrauch in EEVEE 5,2 GB von 8 GB (Maximaler VRAM der verwendeten Hardware). Bei Unreal Engine 5 ist der VRAM vollständig ausgelastet. Das führt dazu, dass die visuelle Qualität darunter leidet und nicht alle Texturen zur Darstellung geladen werden können (die visuelle Qualität kann in den Grafiken im Anhang beurteilt werden).

Um eine optimale Leistung zu erzielen, ist es entscheidend, sicherzustellen, dass die Berechnungen für die direkte Beleuchtung von Objekten, indirekte Beleuchtung und Schatten gleichermaßen effizient sind. Andernfalls könnte ein System das andere ausbremsen und die gesamte Echtzeit-Performance beeinträchtigen.

8 Fazit

Die zentralen Forschungsfragen der Bachelorarbeit waren die verschiedenen Techniken zur Lichtberechnung in Echtzeit-Render-Engines zu analysieren und hinsichtlich ihrer Performance und visuellen Qualität zu überprüfen.

Bei dem Vergleich der Echtzeit-Render-Engines EEVEE und Unreal Engine 5 hat sich gezeigt, dass die Unterschiede in den technischen Methoden bei der Erzeugung von fotorealistischen Bildern in Echtzeit einen erheblichen Unterschied in visueller Qualität und Performance zeigen.

EEVEE schafft es durch die Verwendung von Rasterisierung in besonders hoher Geschwindigkeit Materialeigenschaften verschiedener Oberflächen zu erfassen und die Objekte dann in Echtzeit darzustellen. Dafür müssen Abstriche in der Qualität bezüglich Schatten und indirekter Beleuchtung, die wesentlich für die Darstellung von fotorealistischen Bildern sind, gemacht werden. Übergangsschatten und die Beziehung der Beleuchtung von Objekten zueinander werden vernachlässigt. EEVEE eignet sich hervorragend für die Vorschau von Materialeigenschaften, interaktiven Objekten und Szenen, bei denen eine hohe Detailgenauigkeit nicht erforderlich ist.

Unreal Engine 5 verwendet zusätzlich zur Rasterisierung das Raytracing. Die Kombination beider Techniken ermöglicht eine hochwertige visuelle Qualität von Szenen und Objekten. Durch die akkuratere Beachtung von physikalischen Gesetzmäßigkeiten bietet Unreal Engine 5 eine fotorealistische Darstellung, bei der komplexe Lichtberechnungen vorgenommen werden, die Schatten, indirekte Beleuchtung und Materialeigenschaften berücksichtigen. Die Engine ist gut geeignet für Computerspiele und im Filmbereich. Raytracing, ohne den Einsatz von Rasterisierung, würde ein realistischeres Ergebnis liefern, ist aber noch zu rechenintensiv, selbst mit der Unterstützung von Hardwarebeschleunigung.

Die Bachelorarbeit hat gezeigt, dass sehr viele unterschiedliche Faktoren Einfluss haben und zu berücksichtigen sind, damit eine fotorealistische Darstellung in Echtzeit erfolgen kann.

9 Ausblick

Die Welt der Echtzeit-Render-Engine hat in den letzten Jahren enorme Fortschritte gemacht und die Ansprüche an visuelle Qualität und Realismus in Videospielen, Simulationen und virtuellen Welten steigen ständig. Um diesen Ansprüchen gerecht zu werden, spielen die effizienten Berechnungen von physikalischen Lichtphänomenen eine entscheidende Rolle. In diesem abschließenden Kapitel wollen wir einen Blick auf mögliche Zukunftstechnologien werfen.

9.1 Zukunft von Echtzeit-Render-Engines

Eine der wichtigsten Voraussetzungen für die Weiterentwicklung von Echtzeit-Render-Engines ist die Anwendung von leistungsfähiger Hardware. Mit der Einführung von Grafikkarten, die speziell für Raytracing optimiert sind, wie beispielsweise die NVIDIA RTX-Serie, hat sich die Leistungsfähigkeit erheblich verbessert. Die Möglichkeiten von Hardwareunterstützung könnten in der Zukunft zu noch effizienteren Render-Prozessen führen.

Eine Richtung für die Zukunft von Raytracing in Kombination mit schneller werdender Hardware ist die Integration verschiedener Raytracing-Techniken. Aktuell sind Render-Engines wie Global Illumination und Schatten als eigenständige Systeme implementiert. Wir könnten jedoch Fortschritte sehen, bei denen diese Technologien zu einer einheitlichen Lösung zusammengeführt werden. Diese Integration könnte zu effizienten Render-Prozessen führen.

9.1.1 Spektrales Rendern

Mit steigender Hardware-Kapazität kann es in Zukunft möglich sein, die Betrachtungsweise des Lichts mit Hilfe der geometrischen Optik um die elektromagnetischen Eigenschaften des Lichts zu ergänzen. Konzepte und Ansätze gibt es bereits von Musbach [42] und Steinberg Et al.[43]. Hierbei wird Licht als sich ausbreitende Welle modelliert, bei der die Welleneigenschaften, wie Wellenlänge, Amplitude und Frequenz berücksichtigt werden. Die Wellendarstellung ermöglicht eine detaillierte Simulation von Lichtphänomenen, wie Interferenz, Beugung, Streuung und Polarisation.

9.2 Künstliche Intelligenz

Eine vielversprechende Richtung für zukünftige Echtzeit-Algorithmen zur Bildverbesserung besteht darin, verstärkt maschinelles Lernen einzusetzen. Durch den Einsatz von neuronalen Netzwerken können Algorithmen entwickelt werden, die komplexe Muster im Rauschen erkennen und hochwertige Ergebnisse erzielen. Die Integration von maschinellem Lernen in Algorithmen kann zu einer erheblichen Verbesserung der Bildqualität bei der Verwendung von Raytracing führen.

Zukünftige Forschung könnte sich darauf konzentrieren, Denoising-Algorithmen zu entwickeln, die sich an unterschiedliche Szenarien und Anforderungen anpassen können.

Aber auch im Bezug zu Raytracing kann maschinelles Lernen helfen, das Verfolgen von Strahlen zu optimieren und die Berechnungszeit zu verkürzen. Durch den Einsatz von maschinellem Lernen können Algorithmen entwickelt werden, die den Prozess effizienter gestalten und die Anzahl der Berechnungen für Strahlen reduzieren. Ansätze gibt es hierfür, wie z.B. von Thomas Müller et al. [44].

Alle diese Systeme können dabei helfen, eines Tages physikalisch akkurate Lichtberechnungen in Echtzeit durchführen zu können, die von der Realität nicht mehr zu unterscheiden sind.

10 Literaturverzeichnis

- [1] M. Pharr, W. Jakob, und G. Humphreys, *Physically based rendering: from theory to implementation*, Fourth edition. Cambridge, Massachusetts: The MIT Press, 2023.
- [2] T. Möller, E. Haines, und N. Hoffman, *Real-time rendering*, Fourth edition. Boca Raton: CRC Press, Taylor and Francis Group, 2018.
- [3] C. Cinotti, „The Rendering Pipeline | WebGPU | Video“, 16. Januar 2023.
<https://carmencinotti.com/2023-01-16/how-to-render-a-webgpu-triangle-series-part-six-video/> (zugegriffen 10. August 2023).
- [4] Emmett Kilgariff, Henry Moreton, Nick Stam, und Brandon Bell, „NVIDIA Turing Architecture In-Depth“, *NVIDIA Technical Blog*, 14. September 2018.
<https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/> (zugegriffen 11. August 2023).
- [5] G. Laguna, „NVIDIA Unveils Quadro RTX, World’s First Ray-Tracing GPU“, *NVIDIA Newsroom*.
<http://nvidianews.nvidia.com/news/nvidia-unveils-quadro-rtx-worlds-first-ray-tracing-gpu> (zugegriffen 12. August 2023).
- [6] „ISO/IEC 2382 - 2015-05 - Beuth.de - <https://www.beuth.de/de/norm/iso-iec-2382/235389372>“.
<https://www.beuth.de/de/norm/iso-iec-2382/235389372> (zugegriffen 24. August 2023).
- [7] P. Scholz, *Softwareentwicklung eingebetteter Systeme: Grundlagen, Modellierung, Qualitätssicherung*. in Xpert.press. Berlin Heidelberg New York: Springer, 2005.
- [8] J. D. Foley, *Computer graphics: principles and practice*, Bd. 12110. Addison-Wesley Professional, 1996.
- [9] R. W. Boyd, *Radiometry and the detection of optical radiation*. in Wiley series in pure and applied optics. New York: Wiley, 1983.
- [10] E. Hering und G. Schönfelder, *Sensoren in Wissenschaft und Technik: Funktionsweise und Einsatzgebiete*. in Praxis. Wiesbaden: Vieweg + Teubner, 2012.
- [11] Cmglee, *Comparison of photometric units: luminous flux, luminous intensity, luminous exitance, luminance and illuminance (purple) and radiometric units: radiant flux, radiant intensity, radiant exitance, radiance and irradiance (red) by CMG Lee based on <http://slideshare.net/chinkitkit/chapter-1-radiometryandphotometry-slide-2>*. Zugegriffen: 27. August 2023. [Online]. Verfügbar unter:
https://commons.wikimedia.org/wiki/File:Photometry_radiometry_units.svg
- [12] J. T. Kajiya, „THE RENDERING EQUATION“, Bd. 20, Nr. 4, 1986.
- [13] Blender, „Diffuse BSDF“.
<https://docs.blender.org/manual/en/latest/render/shader/shader/diffuse.html#diffuse-bsdf> (zugegriffen 20. August 2023).
- [14] B. Burley und W. D. A. Studios, „Physically-based shading at disney“, gehalten auf der Acm Siggraph, vol. 2012, 2012, S. 1–7.
- [15] B. Burley, „Physically-Based Shading at Disney“.
- [16] „Global Illumination and Path Tracing - <https://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing/introduction-global-illumination-path-tracing.html>“.
<https://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing/introduction-global-illumination-path-tracing.html> (zugegriffen 27. August 2023).
- [17] Henrik, „upload.wikimedia.org/wikipedia/commons/8/83/Ray_trace_diagram.svg - https://upload.wikimedia.org/wikipedia/commons/8/83/Ray_trace_diagram.svg“.
https://upload.wikimedia.org/wikipedia/commons/8/83/Ray_trace_diagram.svg (zugegriffen 16. August 2023).
- [18] L. David, „REAL vs CGI“, *Behance*, Januar 2020.
<https://www.behance.net/gallery/90722129/REAL-vs-CGI> (zugegriffen 16. August 2023).

- [19] Blender, „EVEE Introduction“.
<https://docs.blender.org/manual/en/latest/render/eevee/introduction.html> (zugegriffen 10. August 2023).
- [20] Blender, „Introduction Cycles“.
<https://docs.blender.org/manual/en/latest/render/cycles/introduction.html> (zugegriffen 11. August 2023).
- [21] Blender, „Principled BSDF“.
https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html (zugegriffen 1. August 2023).
- [22] Blender, „Screen Space Reflections“.
https://docs.blender.org/manual/en/latest/render/eevee/render_settings/screen_space_reflections.html (zugegriffen 29. August 2023).
- [23] Blender, „Ambient Occlusion“.
https://docs.blender.org/manual/en/latest/render/eevee/render_settings/ambient_occlusion.html#ambient-occlusion (zugegriffen 29. August 2023).
- [24] Blender, „Indirect Lighting“.
https://docs.blender.org/manual/en/latest/render/eevee/render_settings/indirect_lighting.html (zugegriffen 29. August 2023).
- [25] Blender, „Irradiance Volumes“.
https://docs.blender.org/manual/en/latest/render/eevee/light_probes/irradiance_volumes.html (zugegriffen 29. August 2023).
- [26] E. Eisemann, M. Schwarz, U. Assarsson, und M. Wimmer, *Real-time shadows*. CRC Press, 2011.
- [27] Epic Games, Inc., „Lumen Global Illumination and Reflections“.
<https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/> (zugegriffen 26. August 2023).
- [28] Epic Games, Inc., „Lumen Technical Details“.
<https://docs.unrealengine.com/5.2/en-US/lumen-technical-details-in-unreal-engine/> (zugegriffen 26. August 2023).
- [29] Epic Games, Inc., „Mesh Distance Fields“.
<https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/MeshDistanceFields/> (zugegriffen 28. August 2023).
- [30] J. C. Hart, „Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces“, *Vis. Comput.*, Bd. 12, Nr. 10, S. 527–545, 1996.
- [31] Flafla2, „Raymarching Distance Fields: Concepts and Implementation in Unity -
<https://adrianb.io/2016/10/01/raymarching.html>“.
<https://adrianb.io/2016/10/01/raymarching.html> (zugegriffen 26. August 2023).
- [32] Epic Games, Inc., „Hardware Ray Tracing“.
<https://docs.unrealengine.com/5.1/en-US/hardware-ray-tracing-in-unreal-engine/> (zugegriffen 28. August 2023).
- [33] Scott, „Viewport FPS“. 29. Mai 2023. Zugegriffen: 6. August 2023. [Online]. Verfügbar unter:
<https://github.com/ScottishCyclops/viewport-fps>
- [34] Epic Games, Inc., „Render Resource Viewer“.
<https://docs.unrealengine.com/5.2/en-US/render-resource-viewer-in-unreal-engine/> (zugegriffen 12. August 2023).
- [35] *Blender, Eevee and Unreal Two incredible Free programs . A basic comparison of the two renderers.*, (6. Dezember 2020). Zugegriffen: 25. August 2023. [Online Video]. Verfügbar unter:
<https://www.youtube.com/watch?v=QHEbIODBGBA>
- [36] A. Botero, „Blender Datasmith Export“. 4. August 2023. Zugegriffen: 9. August 2023. [Online]. Verfügbar unter: <https://github.com/0xafbf/blender-datasmith-export>
- [37] Blender, „Light Objects“. https://docs.blender.org/manual/en/latest/render/lights/light_object.html (zugegriffen 9. August 2023).
- [38] Epic Games, Inc., „Physical Lighting Units“.
<https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/LightingAndShadows/PhysicalLightUnits/> (zugegriffen 10. August 2023).

- [39] A. V. Arecchi, R. J. Koshel, und T. Messadi, *Field guide to illumination*. in SPIE field guides, no. FG11. Bellingham, Wash: SPIE, 2007.
- [40] Jürgen Furrer, „ies library“. <https://ieslibrary.com/en/browse/manufactur/bega#ies-0118f5f48d259f005b568eb221f94771> (zugegriffen 9. August 2023).
- [41] „blender/intern/cycles/kernel/light/point.h at main · blender/blender“, *GitHub*. <https://github.com/blender/blender/blob/main/intern/cycles/kernel/light/point.h> (zugegriffen 12. August 2023).
- [42] A. Musbach, G. W. Meyer, F. Reitich, und S. H. Oh, „Full Wave Modelling of Light Propagation and Reflection“, *Comput. Graph. Forum*, Bd. 32, Nr. 6, S. 24–37, 2013, doi: <https://doi.org/10.1111/cgf.12012>.
- [43] S. Steinberg, R. Ramamoorthi, B. Bitterli, E. d’Eon, L.-Q. Yan, und M. Pharr, „A Generalized Ray Formulation For Wave-Optics Rendering“, 2023, doi: 10.48550/ARXIV.2303.15762.
- [44] T. Müller, F. Rousselle, A. Keller, und J. Novák, „Neural control variates“, *ACM Trans. Graph.*, Bd. 39, Nr. 6, S. 1–19, Dez. 2020, doi: 10.1145/3414685.3417804.

11 Anhang

Anhang 1: Visuelle Artefakte im Tisch und Boden, durch die maximale Auslastung des VRAMs in Unreal Engine 5



12 Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel:

Analyse und Implementierung von Shader Techniken zur Lichtberechnung in Echtzeit-Render-Engine

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Datum

Unterschrift