UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

BVH- und Line-Space-Kombination zur Pathtracing-Beschleunigung

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Johannes Braun

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Kevin Keul, M.Sc. (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Dezember 2017

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Abstract

In dieser Arbeit werden Methoden und Maße getestet, nach denen beim Pathtracing eine Auswahl zwischen Line Space und Bounding Volume Hierarchie getroffen werden kann, die die Vorteile der beiden Datenstrukturen ausnutzen. Die Strukturen sind innerhalb der *Bounding Box* jedes Objekts (Objektlokal) definiert und jeder Line Space enthält in den Shafts jeweils eine Kandidaten-ID. Als Implementationsbasis dient ein eigenes C++ und OpenGL Framework, in dem das Pathtracing und die Line Space Generierung über Compute Shader stattfindet. Die Maße schließen die Wahrscheinlichkeitsverteilung, die Effektabhängigkeit, sowie einen Distanzgrenzwert ein und werden gegen verschiedene Szenen getestet. Die Ergebnisse zeigen in den meisten Situationen einen deutlichen Geschwindigkeitszuwachs bei teils nur geringen visuellen Unterschieden, wobei das Wahrscheinlichkeitsmaß die qualitativ hochwertigsten Bilder für den gegebenen Leistungszuwachs erbringt. Die grundlegenden Probleme des Line Space im Vergleich mit der BVH, nämlich der hohe Speicherverbrauch und die lange Generierungszeit, bleiben aber trotz der objektlokalen Struktur, der minimalen Datenmenge pro Shaft und der Compute Shader Implementierung, erhalten.

This thesis tests several methods and measures in pathtracing for selecting either the Line Space or the Bounding Volume Hierarchy data structure to make use of the advantages of both. The structures are defined locally around each object and each Line Space shaft contains one candidate ID each. All implementation is done as a C++ and OpenGL framework with compute shaders handling the pathtracing and Line Space generation. The measures include the probability distribution, the effect dependency, as well as a distance threshold and are tested against several different scenes. In most situations, the results show a noticeable increase in performance, partly only with minor visual differences, with the probability measure producing the highest quality images for a given performance. The fundamental problems of the Line Space concering the high memory consumption and a long generation time compared to the BVH still persist, despite the object local structure, a minimal amount of data per shaft and the compute shader implementation.

Inhaltsverzeichnis

1 Einleitung

Die Erzeugung digitaler Bilder, auch Bildsynthese genannt, steht in der Computergrafik im Mittelpunkt. Raytracing ist dabei eine essentielle Methode, bei der das Bild durch aus der Kamera verschossene Strahlen, sogenannte Rays, und deren nächste Schnittpunkte mit einer Szenengeometrie berechnet wird. Dies orientiert sich am Verhalten von realem Licht. *Pathtracing* als spezielle Form des Raytracing betrachtet dabei jeweils eine Abfolge von Strahlen pro Pixel, sog. Pfade. Da die Szene in den meisten Fällen aus einer großen Menge von Dreiecken besteht, ist eine gewisse Strukturierung der Daten vonnöten, durch die solche Schnittpunktkandidaten schneller gefunden werden können, als würde man mit jedem einzelnen testen. Nach aktuellem Stand der Technik ist die am weitesten verbreitete Datenstruktur die *Bounding Volume Hierarchy* (BVH), die die Szene und die enthaltenen Dreiecke in einen Baum aus *Bounding Boxes* einordnet. Dies verringert die Suchzeit, was in einer schnellen, exakten Schnittpunktsuche resultiert.

Ein neuer Ansatz, der ein anderes Paradigma verfolgt, ist der Line Space. Dieser wird immer auf eine einzelne Bounding Box angewendet und teilt deren Seiten in Bereiche auf. Solche *Patches* werden zwischen unterschiedlichen Seiten anschließend durch Volumen, den sog. *Shafts*, verbunden. Die Bounding Boxes können selbst wiederum in einer anderen Datenstruktur, wie einem Voxel Gitter, liegen. In den Shafts können letztendlich Informationen über die eingeschlossene Geometrie gespeichert werden. Beispielsweise ist dafür das vorderste im Shaft befindliche Dreieck geeignet. Ein solcher Line Space mit Shafts mit je einem Kandidaten verspricht durch die Unabhängigkeit der Polygonanzahl eine schnellere, jedoch ungenauere Schnittpunktabfrage als die BVH.

Die Grundfrage in dieser Arbeit ist die, in welchem Maße sich die beiden Datenstrukturen im Pathtracing kombinieren lassen, sodass die Vorteile beider die Nachteile der jeweils anderen Struktur überdecken können. Dahingehend wird eine Methode gewählt, mit der die Datenstrukturen nicht über die gesamte Szene, sondern über je ein Objekt gelegt werden, was dynamische Szenen ermöglicht.

Organisation Zuerst werden die Grundlagen der Bildsynthese nochmals genauer beleuchtet. Anschließend wird das Pathtracing eingeführt und beschrieben, welche Abhängigkeiten es dabei gibt, die später bei der Datenstrukturselektion aufgegriffen werden können. Im nächsten Schritt wird beschrieben, wie Bounding Volume Hierarchien aufgebaut sind, welche Möglichkeiten es gibt, sie über Algorithmen zu generieren, sowie die allgemeine und eine spezelle Methode der Traversierung der Hierarchie. Weiterhin wird die Strukturierung des genutzten Line Space erläutert. Im Anschluss daran wird auf die C++ und GLSL Implementation eingegangen. Dies schließt die Traversierung und den Aufbau der Datenstrukturen, aber auch die Struktur des gesamten genutzten Frameworks mit ein. Letztendlich folgt die Aufstellung der Selektionsmaße und die darauf aufgebaute Evaluation mit verschiedenen Testszenen.

2 Grundlagen

2.1 Bildsyntheseverfahren

Als Bildsynthese bezeichnet man in der Computergrafik das digitale Generieren von Bildern. Derzeit gibt es zwei Kategorien von Bildsyntheseverfahren: zum einen die in Spielen Anwendung findende Rasterisierung, zum anderen das eher in Nichtechtzeitanwendungen angewandte Raytracing.

2.1.1 Rasterisierung

Die Rasterisierung besteht üblicherweise daraus, dass für jedes Geometrieobjekt bestimmt wird, welche Pixel es füllt. Dies können Dreiecke, aber auch Punkte und Linien sein. Dabei werden erst die Geometrieeckpunkte, auch Vertices genannt, gesetzt und anschließend abhängig von der Art des Objekts verbunden. Im letzten Schritt werden die Pixel durch einem sog. *Fragment Shader*[1] mit Bilddaten, wie der Farbe des Pixels, gefüllt. Nachteil an der Rasterisierung ist, dass ein Fragment keine Informationen über die Restgeometrie der gerenderten Szene kennt. Somit können dies einbeziehende Effekte wie Reflexionen, Transmissionen, wie Lichtbrechung z. B. bei Glas, oder *Subsurface Scattering*, welches Durchdringen von Licht bei halbtransparenten Materialien wie Haut oder Wachs simuliert, nicht oder nur durch ungenaues Nachbearbeiten des gerenderten Bildes dargestellt werden.

2.1.2 Raytracing und Pathtracing

Die genannten Probleme umgeht man beim *Raytracing*, indem man im Gegensatz dazu Strahlen von der Kamera aus in die Szene verschießt, mit der Szenengeometrie schneidet und anschließend am Schnittpunkt die Pixelfarbe berechnet. Dies hat den Vorteil, dass man im Anschluss daran vom Schnittpunkt aus weitere Strahlen verschießen kann und dadurch grundsätzlich die Möglichkeit hat, Reflexionen und Transmissionen darzustellen. Der Grund, warum Raytracing nicht so häufig in Echtzeitanwendungen genutzt wird, ist die Komplexität der Schnittpunkttests, bei denen im schlechtesten Fall jedes Dreieck getestet werden muss. Abhilfe schaffen dabei eine Vielzahl an Datenstrukturen, wie sie in Abschnitt 2.2 beschrieben werden. Dadurch können einfachere Szenen mit wenigen Objekten auf stärkeren Grafikkarten in Echtzeit mit Raytracing gerendert werden. Komplexere dynamische Szenen wie Spiele sind aber weiterhin eine Herausforderung.

Eine Art des Raytracing ist das *Pathtracing*, bei dem das Bild durch statistische Bildsynthese erzeugt wird. Für jeden generierten Primärstrahl[2] wird dabei nach einem gefundenen Schnittpunkt immer maximal ein weiterer Strahl generiert, während naive Raytracing-Ansätze eine Aufteilung des Strahls erlauben. Dieser wird nach dem gleichen Prinzip weiter verfolgt, sodass dadurch ein *Pfad* entsteht. Ein

[1] Bezeichung von OpenGL, DirectX *Pixel Shader* genannt

[2] Strahl, der direkt von der Kamera aus startet

Schnittpunkt mit der Szene wird beim Pathtracing auch *Bounce* genannt. Ziel des Pathtracing ist es, die Lösung der sog. *Rendergleichung* [Kaj86]

$$L(x,w_o) = L_e(x,w_o) + \int_\Omega f_s(x,w_i,w_o)L(x,w_i)\langle w_i,n\rangle dw_i \tag{1}$$

anzunähern, die eine Aussage über die Strahldichte des ausgehenden Lichtes aus einem Weltpunkt x in Richtung w_o trifft. Die Gleichung bezieht sich immer auf eine Hemisphäre Ω über x, welche entlang der Oberflächennormalen ausgerichtet ist. Innerhalb dieser sind unendlich viele eingehende Lichtstrahlen w_i definiert, entlang welcher Licht mit einer Strahldichte von $L(x,w_i)$ einfällt. Die gesamte Bestrahlungsstärke, die in x anliegt, wird durch

$$E = \int_\Omega L\cos(\theta)d\Omega$$

berechnet [PJH16]. Mit $\cos(\theta) = \langle w_i,n\rangle$ folgt durch Gewichtung mit der materialabhängigen BSDF (Abschnitt 2.1.4) $f(x,w_i,w_o)$ die Rendergleichung. Emittiert ein gegebenes Material Licht in Richtung w_o, wird diese als Leuchtdichte $L_e(x,w_o)$ additiv hinzugerechnet.

Die Rendergleichung bezieht sich auf das Verhalten von Licht in der realen Welt, wobei Lichtpfade in Lichtquellen beginnen. Beim Pathtracing verhält es sich entgegengesetzt, die Strahlen starten also in der Kamera. Dies ist vonnöten, da bei uniformer Strahlverteilung in Lichtquellen durch die geringe Wahrscheinlichkeit, die Kamera zu treffen, eine hohe Varianz entstehen würde. In dem Kontext ist somit das Verhältnis von eingehendem Strahl w_i und ausgehendem Strahl w_o vertauscht.

2.1.3 Monte Carlo Integration

Das größte Problem bei der Berechnung der Rendergleichung ist die Integration über Ω, da digitale Systeme nur diskrete Datensätze berechnen können. Daher kann die Gleichung derzeit nur angenähert berechnet werden. Dazu eignet sich der Ansatz der *Monte Carlo Integration* [Ham60]. Diese sagt für ein beliebiges Integral aus, dass es mithilfe von N gleichverteilten Stichproben (*Samples*) $x_1,x_2,...,x_N \in \Omega$ nach

$$\int_\Omega g(x)\cdot p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} g(x_i) \tag{2}$$

angenähert werden kann. Dabei ist $p(x)$ die sog. Wahrscheinlichkeitsdichtefunktion[3], welche für ein x die Wahrscheinlichkeit angibt, mit der ein entsprechendes $g(x)$ daraus resultiert.

Mit $f(x) = g(x)\cdot p(x)$ kann nun umgeformt werden zu

$$\int_\Omega f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}. \tag{3}$$

[3]PDF, engl. *probability density function*

Wird x_i so gewählt, dass $p(x_i)$ stets signifikant ist, wird von *Importance Sampling* gesprochen [PJH16]. Dies kann im Falle des Pathtracing bedeuten, dass bspw. bei reflektierenden Materialien öfter Strahlen entlang des reflektierten Sichtstrahls verschossen werden. Bilder, die mit Importance Sampling gerendert wurden, weisen daher ein geringeres Rauschen auf als solche ohne.

Angewandt auf die Rendergleichung (1) lässt sich diese somit umformen zu

$$L(x,w_o) = L_e(x,w_o) + \frac{1}{N}\sum_{0}^{N}\frac{f_s(x,w_i,w_o)L(x,w_i)(w_i,n)}{p(w_o,w_i)}, \tag{4}$$

wobei die PDF $p(w_o,w_i)$ die Wahrscheinlichkeit darstellt, dass aus einem eingehenden Strahl mit Richtung w_o ein ausgehender Strahl w_i entsteht. Diese ist symmetrisch, sodass gilt $p(w_o,w_i) = p(w_i,w_o)$.

2.1.4 Bidirectional Scattering Distribution Function

Die sog. *Bidirectional Scattering Distribution Function*, kurz BSDF, beschreibt die Art und Weise, mit der eingehendes Licht von einer Oberfläche wieder abgestrahlt wird. Die meisten BSDFs beschreiben lediglich die BRDF (*Bidirectional Reflectance Distribution Function*), die nur das reflektierte Licht betrachten und nicht das transmittierte. Manche davon, wie *Phong* [Pho75] oder *Blinn Phong* [Bli77] haben keine physikalische Grundlage und beziehen die Energieerhaltung nicht in die Berechnung mit ein. Dem gegenüber stehen physikalisch basierte Verfahren wie das *Schlick-Modell* [Sch93] oder das für den Pathtracer in dieser Arbeit genutzte *Cook-Torrance-Modell* [Coo82]. Letzteres wurde entsprechend [Wal07b] mit einer BTDF (*Bidirectional Transmission Distribution Function)* in Kombination mit dem Lambertschen Reflektanzmodell zu einer gesamten BSDF erweitert, sodass insgesamt drei verschiedene Effekttypen unterschieden werden können (Tabelle 1). Abbildung 1 zeigt exemplarisch die jeweiligen Effekte und deren assoziierte BSDF.

Tabelle 1: Substitution der beim Pathtracing möglichen Effekte.

Diffus	Lambertsches Reflektanzmodell
Reflexion	Mikrofacetten BRDF
Transmission	Mikrofacetten BTDF

Da nur jeweils ein Strahl mit jedem Bounce beim Pathtracing weiter verfolgt wird, muss durch *Russian Roulette* (Abschnitt 2.1.5) eine zufällige Funktion von den genannten Effekten ausgewählt werden.

Lambertsches Reflektanzmodell Die Verfahren bauen auf Grundlage des vollständig diffusen *Lambertschen Beleuchtungsmodells* [Lam92] auf. Dieses Modell beschreibt eine Lichtabstrahlung, die unabhängig vom Betrachtungswinkel ist. Ausgehende Strahlen werden dabei mit $\cos(\theta)$ und der Abstrahlungsfläche A gewichtet.

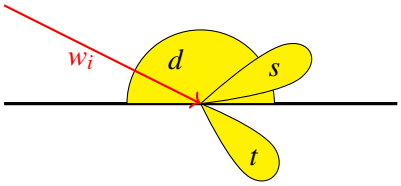


Abbildung 1: Schematische Darstellung einer BSDF mit der diffusen Reflexion d, der spekularen Reflexion s und der Transmission t bei eingehendem Strahl w_i.

θ ist der Winkel zwischen dem ausgehenden Strahl und der Oberflächennormalen. Die Gewichtung folgt aus dem Lambertschen Kosinusgesetz [Lam92], nach dem für eine Lichtstärke I gilt

$$I \sim A\cos(\theta). \tag{5}$$

Dies lässt sich dadurch erklären, dass bei flacherer Einstrahlung eine feste Leuchtdichte über eine größere Fläche verteilt ist. Eine gewichtete Hemisphäre bildet eine Kugel (Abbildung 2), wodurch dessen Fläche $A = 4\pi \cdot 0.5^2 = \pi$ beträgt. Mit $I = \cos(\theta) \cdot C \cdot I_L$ (C ist die Materialfarbe, I_L das eingehende Licht) folgt für die ausgehende Leuchtdichte L

$$L = \frac{C \cdot I_L}{\pi}. \tag{6}$$

Durch die Kosinus-gewichtete Verteilung auf der Kugeloberfläche der Abstrahlung folgt für die Wahrscheinlichkeitsdichte

$$p(w_i, w_o) = \frac{\cos(\theta)}{\pi}. \tag{7}$$

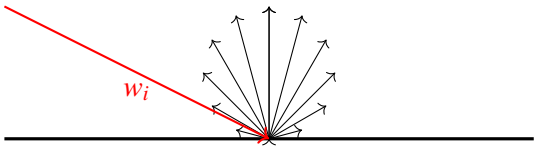


Abbildung 2: Kosinus-gewichtete Abstrahlung von einer Lambertschen Oberfläche.

Mikrofacetten-BRDF Die genutzten Modelle bauen auf physikalischen Messwerten auf und beschreiben den Reflexionsgrad in Form einer Approximation einer Funktion aus Wellenlänge und Einfallswinkel. Da die Wellenlänge durch Diskretisierung in Rot-, Grün- und Blauwerte keine Rolle spielt, wird sie hier nicht betrachtet. Die Funktion zur Berechnung der BRDF

$$f(i,o,n) = \frac{F(i,h_r)G(i,o,h_r)D(h_r)}{4|i \cdot n||o \cdot n|} \tag{8}$$

aus dem Einfallsvektor i, dem Reflektierten Vektor o und der Oberflächennormalen n besteht aus drei das Material definierenden Hauptkomponenten im Zähler. Hierbei

ist h_r der Halbvektor zwischen der Einfalls- und Reflexionsrichtung. Dieser wird im Folgenden äquivalent zur Mikrofacettennormalen m benutzt.

$F(i,h_r)$ ist der Fresnel-Term, der oft mithilfe von Schlicks Approximation [Sch93] durch

$$R(\theta) = R_0 + (1-R_0)(1-\cos\theta)^5$$

ausgedrückt wird. Dieser liegt in $[0,1]$ beschreibt den Reflexionskoeffizient $R(\theta)$ bei steilen Einfallswinkeln θ und wird höher, je größer der Winkel ist. Der Faktor R_0 ist der Basiswert bei senkrechter Betrachtung[4] mit dem eingehenden Refraktionsindex η_i und dem ausgehenden η_t. Dieser kann für jedes Material vorberechnet werden mit $R_0 = \left(\frac{\eta_i-\eta_o}{\eta_i+\eta_o}\right)^2$. Zusätzlich dazu wird in dieser Arbeit auch der Metall-Faktor einbezogen, da metallische Materialien im Gegensatz zu *dielektrischen* das eingehende Licht nicht durch Untergrundeffekte (Sub-Surface) diffus streuen, sondern entweder absorbieren oder reflektieren. Dies wird durch eine lineare Interpolation anhand des metallischen Faktors des Materials zwischen F und 1 bestimmt.

Der Geometrieterm $G(i,o,h_r)$ beschreibt die Selbstverschattung der Materialoberfläche durch Mikrofacetten und setzt sich im Falle der in der genutzten BSDF angewandten GGX-Verteilung [Wal07b] aus jeweils verschiedenen partialen Geometrietermen für Einfalls- und Relfexionsrichtung zusammen. Beide lassen sich durch den partiellen Geometrieterm

$$G_1(v,m) = X^+\left(\frac{v\cdot m}{v\cdot n}\right)\frac{2}{1+\sqrt{1+\alpha^2\tan^2\theta_v}}$$

berechnen. Hierbei ist X^+ eine boolesche Funktion, welche durch

$$X^+(x) = \begin{cases} 1 & \text{wenn } x > 0 \\ 0 & \text{andernfalls} \end{cases} \tag{9}$$

definiert ist. Die Terme für Einfalls- und Reflexionsrichtung werden durch $G(i,o,m) = G_1(i,m)G_1(o,m)$ zu einer gesamten geometrischen Verschattung zusammengefasst, wobei α die Rauheit des Materials ist und in $[0,1]$ liegt.

Mit dem Geometrieterm einhergehend ist die Verteilungsfunktion

$$D(m) = \frac{\alpha^2 X^+(m\cdot n)}{\pi\cos^4(\theta_m)(\alpha^2+\tan^2\theta_m)^2}.$$

Diese beschreibt die Wahrscheinlichkeit der Existenz einer Mikrofacettennormalen m bei einer Oberflächennormale n. Da Importance Sampling genutzt werden soll, können mit zwei gleichverteilten Zufallszahlen ξ_1 und ξ_2 aus dem Wertebereich $[0,1)$ die Hemisphärenwinkel

$$\theta_m = \arctan\left(\frac{\alpha\sqrt{\xi_1}}{\sqrt{1-\xi_1}}\right) \qquad \text{und} \qquad \phi_m = 2\pi\xi_2 \tag{10}$$

[4] engl. *Normal Response*

berechnet werden. Anschließend wird eine entlang der Normalenhemisphäre gerichtete, durch die Winkel definierte Mikrofacettennormale m berechnet, an der der einfallende Strahl reflektiert werdem kann, um einen reflektierten Vektor durch $o = 2|i \cdot m|m - i$ zu generieren. Dafür wird θ_m als Winkel zwischen m und n, und ϕ_m als radialer Winkel auf der Hemisphärengrundfläche genutzt (Abb. 3). Da m somit im Tangentialraum definiert ist, muss eine Transformation zu Weltkoordinaten durchgeführt werden.

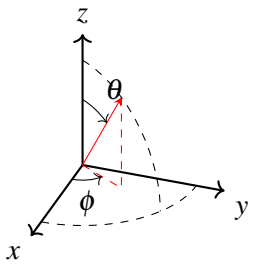


Abbildung 3: Die Hemisphärenwinkel ϕ und θ in 3D dargestellt.

Mikrofacetten-BTDF Die Transmissionsfunktion folgt einem ähnlichen Schema wie die BRDF zuvor. Der Reflektanz-Term wird um die Brechungsindizes η_i und η_o erweitert zu

$$f_t(i,o,n) = \frac{|i \cdot h_t||o \cdot h_t|}{|i \cdot n||i \cdot n|} \cdot \frac{\eta_o^2(1 - F(i,h_t))G(i,o,h_t)D(h_t)}{(\eta_i(i \cdot h_t) + \eta_o(o \cdot h_t))^2}. \tag{11}$$

Der resultierende ausgehende Strahl kann dabei nach der Berechnung der Hemisphärenwinkel des Importance Sampling mithilfe von

$$o = \left(\eta c - \text{sign}(i \cdot n)\sqrt{1 + \eta c^2 - 1}\right) m - \eta i$$

bestimmt werden, mit $c = i \cdot m$ und $\eta = \frac{\eta_i}{\eta_t}$. Wichtig für die allgemeine Verwendung einer BTDF ist der Umstand, dass für die Rendergleichung nicht mehr nur über die obere Hemisphäre, sondern nun über die gesamte Kugel integriert werden muss. In der Praxis spielt das durch die getrennte Betrachtung von Transmission und Reflexion, wiederum jeweils über die untere und obere Hemisphäre, jedoch keine Rolle.

Durch die Verwendung des Fresnel-Wertes F bei der BRDF und $1 - F$ bei der BTDF, werden Fehler, die durch Totalreflexionen bei flachen Einfallswinkeln entstehen, vermieden.

2.1.5 Russian Roulette

Beim Pathtracing muss nach jedem Bounce unterschieden werden, ob der resultierende neue Strahl reflektiert, transmittiert oder diffus gestreut wird. Dies geschieht

üblicherweise durch *Russian Roulette*, eine stochastische Methode, mit der auf Basis von je einer Wahrscheinlichkeit für jeden Effekt zufällig einer der drei Strahlen zur Weiterführung des Pfades ausgewählt wird. Die Wahrscheinlichkeiten werden durch k_d für den diffusen, k_s für den spekular reflektierten und k_t für den transmittierten Strahl beschrieben. Die Auswahl der Parameter ist abhängig von der genutzten BSDF und den Materialeigenschaften. In dieser Arbeit werden diese Werte nochmals mit der Emissionsstärke des Materials interpoliert, sodass bei einem vollständig emissiven Material keine weiteren Strahlen weiter verfolgt werden. Zusätzlich wird ein weiterer Wert k_e betrachtet, die bei einer Emissionsstärke in $[0,1)$ die Wahrscheinlichkeit eines Pfad-terminierenden Emissions-Samples wiederspiegelt. Somit ergeben sich die Wahrscheinlichkeiten wie folgt:

$$\begin{aligned} k_e &= \text{clamp}(e_{\text{Mat}}, 0, 1) \\ k_s &= (1-k_e) \cdot F \\ k_t &= (1-k_e) \cdot (1-F) \cdot t_{\text{Mat}} \\ k_d &= 1-k_e-k_s-k_t \end{aligned} \tag{12}$$

Dabei ist e_{Mat} die Emissionsstärke des Materials, welche für k_e mit der *clamp* Funktion auf ein Intervall $[0,1]$ limitiert wird. Weiterhin ist F der Fresnel-Wert aus der beschriebenen BRDF und t_{Mat} der Transmissionsfaktor des Materials.

2.2 Datenstrukturen

Wie beschrieben besteht das Hauptproblem bei echtzeitfähigen Raytracing-Methoden darin, die Schnittpunkte der Strahlen mit der Szenengeometrie zu finden. Bei der naiven Herangehensweise muss jedes Dreieck dabei einzeln getestet werden, wenn der vorderste Schnittpunkt gefunden werden soll. Zur Reduktion der zu testenden Kandidaten werden Datenstrukturen genutzt, die diese so eingeteilt, dass die Zugriffszeit verringert wird.

Heutzutage übliche Verfahren teilen die Szene räumlich ein. Bspw. wird sie in einer Uniform-Grid-Datenstruktur [uAW87] in gleichmäßige *Voxel*[5] unterteilt. Nachteil daran ist, dass kleine Objekte in großen Szenen eine verhältnismäßig schlechte Voxel-Abdeckung haben. So könnte ein kleines Objekt mit vielen Polygonen vollständig innerhalb eines Voxels liegen und somit keinen Geschwindigkeitsvorteil darbringen. Daher wurden mit der Zeit adaptive Datenstrukturen entwickelt. Ein Octree [uJSuMH10], allgemeiner auch *Recursive Grids* [Jev88], unterteilen rekursiv, sodass Voxel, die viele Polygone enthalten nochmals in 8^3 respektive $N \cdot M \cdot K$ weitere Voxel unterteilt werden können. Jedoch müssen abhängig von der Szene weiterhin leere Voxel getestet werden. Alternativen zum Voxel-Ansatz sind volumetrisch unregelmäßig unterteilte Datenstrukturen wie der *k*-d-Baum [Ben75] und die im nächsten Abschnitt beschriebene *Bounding Volume Hierarchy*.

2.2.1 Bounding Volume Hierarchy

Die Bounding Volume Hierarchy [Hav04] (BVH) teilt die Szenenprimitive räumlich so ein, dass die *Bounding Boxen* der Primitive in einer Baumstruktur eingetragen werden. Dabei sind die jeweiligen Elternknoten wiederum Bounding Boxen um alle Kindknoten. BVHs und daran angelehnte Strukturen sind momentan die am weitesten verbreiteten Datenstrukturen für das Raytracing. Es gibt mehrere Arten, eine BVH aufzubauen. Grundlegend unterscheidet man zwischen *Top-Down*, *Bottom-Up* und *Insertion-Based* Methoden, welche sich für jeweils andere Anwendungsfälle eignen.

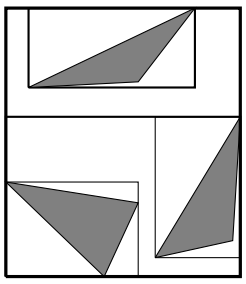


Abbildung 4: Aufbau einer BVH. Die unteren beiden Dreiecke sind zusammengefasst zu einer Bounding Box, die wiederum mit dem oberen Dreieck die gesamte Struktur bilden.

[5] Volumetric (3D) Pixel

Bottom-Up Eine *Bottom-Up*-Methode ist bspw. die *LBVH*[6] [Lau09], welche auf einer gleichmäßigen Unterteilung der Szene und eines daraus berechneten *Morton Codes* basiert. Durch direkte Indizierung ist diese Methode hoch parallelisierbar, resultiert aber aufgrund der uniformen Unterteilung in einem weniger optimalen Baum für die Traversierung, da sich bei Szenen mit dichteren Stellen[7] innerhalb einer Unterteilung eine einseitige Baumgewichtung ergeben kann. Die modernere *HLBVH*[8] beschleunigt das LBVH-Verfahren nochmals durch Hierarchisierung der Generierung auf Basis von Speicherkohärenz [Pan10].

Insertion-Based Bei *Insertion-Based*-Methoden beginnt man mit einem leeren Wurzelknoten, der schrittweise durch Einfügen von Objekten erweitert werden kann. Dies ermöglicht dynamisch verformbare Objekte und allgemein dynamische Szenen, hat aber ebenso den Nachteil eines weniger optimalen Baumes. Aufbauend auf selbst-balancierenden Binärbäumen entstand die Idee des über Baumrotationen nach *Surface Area Heuristic* (SAH) optimierten BVH [Ken08]. Daran anschließend entstanden dann Insertion Based BVHs [Bit13]. Diese nutzen eine Kostenfunktion aus den direkten Kosten eines einzufügenden Knotens und den entsprechenden induzierten Kosten. Erstere beschreiben die Kosten einer gesamten neu enstehenden Bounding Box, die das neue Objekt und Teile der bisherigen BVH mit einschließt. Die induzierten Kosten sind diejenigen, die die Oberflächendifferenz des Wurzelknoten nach Einfügen des neuen Knoten wiederspiegeln. Beide zusammenaddiert ergeben die totale Oberflächenzunahme der BVH. Dies ist ein Optimierungsproblem, mit dem versucht wird, ebendiese Kosten zu minimieren.

Top-Down Die am weitesten verbreiteten Generierungsmethoden sind *Top-Down*-Methoden, bei denen für jeden Knoten die dazugehörigen Kandidaten in immer zwei gleichwertige Bereiche aufgeteilt werden und daraus zwei neue Knoten erstellt werden, mit denen jeweils rekursiv weiter gearbeitet werden kann. Dies verspricht von den Grundmethoden die ausgewogensten Bäume, kann aber schlechter parallelisiert werden und ist statisch, was hinderlich für dynamische Szenen ist. Die Bestimmung der Wertigkeit kann bei einem naiven Ansatz durch Median-Partitionierung[9] oder durch Mittelwert-Partitionierung[10] stattfinden. Beide Methoden weisen jedoch suboptimale Unterteilungen auf, da bei der Median-Bestimmung die Kandidatenposition nicht beachtet wird, und bei der Mittelwert-Bestimmung keine Aussage über die Anzahl der Kandidaten in den jeweils resultierenden Partitionen getroffen wird. Dem kann entgegengewirkt werden durch bessere Gewichtungsmethoden wie die SAH-Kostenfunktion [Wal07a]. Durch die Möglichkeit, einen Kandidaten auch in mehreren Knoten zu haben, können die Bounding Boxen der Knoten kompakter

[6] Linear Bounding Volume Hierarchy

[7] Stellen, an denen sich hoch aufgelöste Objekte befinden

[8] Hierarchical Linear Bounding Volume Hierarchy

[9] Teilung beim $N/2$-ten Wert auf der größten Achse der gesamten Bounding Box

[10] Teilung anhand des des Durchschnittswerts auf der größten Achse der gesamten Bounding Box

werden, da sich der vollständige Kandidat nicht mehr innerhalb eines Knotens befinden muss. Dieser Ansatz wird *SBVH*[11] [Sti09] genannt und ist besonders in Szenen von Vorteil, die nicht-gleichmäßig unterteilte Objekte beinhalten. Neben den Binärbaum-Hierarchien gibt es auch Bäume mit mehreren Kindknoten pro Elternknoten. Diese *MBVH*s[12] [Ern08] resultieren in flacheren Bäumen und können mit einer Pro-Ebenen-Parallelisierung schneller aufgebaut werden. Dabei werden für jede Hierarchiestufe entsprechend Anzahl der neuen Kindknoten neue Threads für die Sub-Unterteilung benutzt. Aufgrund der hohen Auslastung in der ersten nicht-parallelisierten Root-Stufe ist diese Art eher für CPUs als für GPUs geeignet.

Traversierung Für das Finden eines Schnittpunkts in der BVH wird eine Methode verwendet, die *stackless traversal* genannt wird. Normalerweise wird beim Durchlaufen des Baumes ein *Stack*[13] aufgebaut, auf dem die durchlaufenen Knoten abgelegt werden.

Da die Abfrage über einen GLSL Compute Shader stattfindet, ist dies keine Option, da dort der lokale Speicher begrenzt ist und ein Stack mit konstanter Größe initialisiert werden muss. Dies verlangsamt den Algorithmus, bei dem in jeder Abfrage ein neuer Stack erstellt werden muss. Daher wird eine Variante gewählt, die ohne Stack auskommt [Áfr14]. Die implementierte Variante orientiert sich am gezeigten Code.

Der Bitstack wird durch Bit-Shifts verändert und gibt durch das *LSB*[14] den aktuellen Traversierungs-Status an. Ein Wert von 1 sagt aus, dass der derzeitige BVH-Knoten noch nicht vollständig getestet wurde, bspw. dadurch, dass von einem Knoten beide Kindknoten getroffen wurden, jedoch erst einer traversiert wurde. Eine 0 beschreibt einen vollständig traversierten Knoten. Trifft der Algorithmus auf einen Blattknoten, werden die darin referenzierten Kandidaten auf Schnittpunkte getestet. Anschließend wird iterativ auf dem Bitstack durch right-shifts zurück gegangen, während der aktuelle Knoten auf seinen Elternknoten gesetzt wird. Wird eine 1 gelesen, wird der Wert auf 0 gesetzt und sein verbliebener Kindknoten traversiert. Andernfalls wird zurückgegeben, ob ein Kandidat geschnitten wurde.

[11] Split Bounding Volume Hierarchy
[12] Multi Bounding Volume Hierarchies
[13] Container mit exklusivem Zugriff auf das oberste Element
[14] engl. für least significant bit. Das letzte Bit eines integralen Wertes (ganz rechts)

Algorithmus 1: *Stackless* BVH traversierung, orientiert an [Áfr14].

Input: A *ray* and a maximum testing distance *maxDistance*
Output: The nearest intersection with the BVH

```
1  hit ← none
2  bitstack ← 0
3  currentNode ← first node in BVH
4  while ray intersects Scene do
5      if currentNode is inner node then
6          leftHit ← intersects left child of currentNode
7          rightHit ← intersects right child of currentNode
8          if leftHit and rightHit then
9              bitstack ← (bitstack << 1) ∨ 1
10             currentNode ← left child of currentNode
11             continue
12         if leftHit and not rightHit then
13             bitstack ← bitstack << 1
14             currentNode ← left child of currentNode
15             continue
16         if not leftHit and rightHit then
17             bitstack ← bitstack << 1
18             currentNode ← right child of currentNode
19             continue
20     else
21         for candidate in all candidates in currentNode do
22             if ray intersects candidate and hitDistance < maxDistance then
23                 Hit ← candidate at hitDistance
24     while bitstack ∧ 1 = 0 do
25         if bitstack = 0 then
26             return hit
27         currentNode ← parent of currentNode
28         bitstack ← bitstack >> 1
29     currentNode ← right sibling of currentNode
30     bitstack ← bitstack ⊕ 1
31 return hit
```

2.2.2 Line Space

Anstelle der räumlichen Einteilung der Szene basiert der Line Space auf einer direktionalen Einteilung von Bounding Volumes in sogenannte *Shafts*. Dabei werden die Seiten der Räume in Patches unterteilt. Von jedem Patch aus wird anschließend jeweils ein Shaft zu jedem Patch auf allen anderen Seiten generiert (Abbildung 5). Welche Daten diese enthalten, ist implementationsbedingt und kann alle Primitive einschließen, die in dem Shaft liegen, oder einen einzelnen exemplarischen Datensatz.

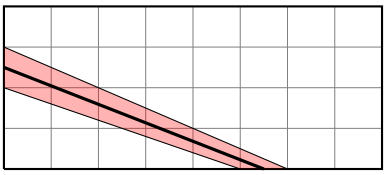


Abbildung 5: Darstellung eines 2D nicht-uniformen Line Space. Exemplarisch ist ein Shaft gekennzeichnet, sowie dessen Zentralstrahl.

Stand der Technik Die bisherigen Implementationen des Line Space bezogen sich meist auf eine globale Datenstruktur, die alle Szenenobjekte einschließt. Datenstrukturen, die dabei als Basis dienten, schließen Voxelbasierte Strukturen wie das Uniform Grid und Rekursive Grids ein [Keu17], bei denen in jedem Voxel jeweils ein Line Space generiert wird. Da das Generieren eines hochauflösenden Globalen Line Space Baumes ein langwieriger Prozess ist, wird eine *Maske* für jeden Voxel generiert und gespeichert. Diese enthält für jeden Shaft Informationen darüber, welche Kindknoten-Voxel geschnitten werden. Damit, und mit vorab konstruiertem Grid-Baum, der in den Knoten die Geometrie enthält, kann die zu testende Kandidatenmenge pro Shaft reduziert werden.

Aufbau In dieser Arbeit wird der Ansatz eines nicht-uniformen Objekt-lokalen Grid Line Space verwendet, der auf dem Aufstellen eines Grids aus $N_G \times M_G \times K_G$ *Voxeln* in der Objekt-Bounding-Box, welche jeweils einen Line Space mit den Patch Unterteilungen $N_L \times M_L \times K_L$ enthalten.

Dieser hat mehrere Vorteile gegenüber den ursprünglichen Implementationen mit uniformen Line Spaces der Größe N^3 in Uniform-Grids respektive Recursive Grids. Zum einen können die Szenenobjekte durch Objekt-Lokalität auch nach der Generierung dynamisch transformiert werden. Auch können Objekte gleichmäßiger unterteilt werden, da keine Stauchung oder Streckung der Line Spaces stattfindet, und somit nicht mehr Shafts generiert werden als nötig. Bspw. können sehr flache Objekte wie Böden mit einem Grid der Höhe 1 mit Line Spaces derselben Höhe unterteilt werden, was durch die Speicherkomplexität $O(N^4)$ eine große Reduktion im Speicherverbrauch erlaubt.

Die Komplexität kann aus der Formel für die Berechnung der gesamten Menge von Shafts

$$C = 2 \cdot (2k_x(k_y + k_z) + k_x^2 + 2k_y(k_x + k_z) + k_y^2 + 2k_z(k_x + k_y) + k_z^2)$$

hergeleitet werden. Dabei ist k_a die Anzahl von Patches auf der Volumenachse a einer Volumenseite der Höhe h_a und der Breite w_a mit $k_a = h_a \cdot w_a$. Dies resultiert bei exemplarischer kubischer Unterteilung N^3 mit $h_a = N$ und $w_a = N$ in der genannten Speicherkomplexität.

Weiterhin werden Line Spaces und deren umfassende Datenstrukturen nur dort erstellt, wo sie gebraucht werden, innerhalb der Bounding Boxen der Geometrien. Leere Räume in der Szene werden somit nicht betrachtet. Dadurch kann man entweder die Voxelauflösung der Geometrie erhöhen, oder den allgemeinen Speicherverbrauch senken.

In den Line Space Shafts wird dazu der Objekt-relative Index eines repräsentativen Dreiecks der Geometrie gespeichert. Dieser wird durch einen Schnittpunkttest mit einem Strahl durch die Zentren von Start- und End-Patch generiert und beschreibt die Ebene, die durch das vorderste getroffene Dreieck im Shaft aufgespannt wird. Als Hilfestellung beim Finden eines Schnittpunkts wird eine vorab für jedes Objektvolumen generierte BVH verwendet.

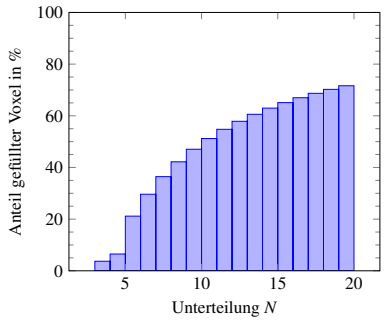


Abbildung 6: Anteil der gefüllten Voxel bei einem Würfel mit Voxel-Unterteilung N^3. Entspricht den innenliegenden Voxeln mit $f(N) = \frac{(N-2)^3}{N^3}$.

Für jeden Voxel wird getestet, ob er Kandidaten (Dreiecke) enthält. Durch Auslassen leerer Voxel kann je nach Objekt die Anzahl der Line Spaces nochmals reduziert werden (Abbildung 6). Wird ein leerer Voxel festgestellt, findet für diesen keine Line Space Generierung statt. Dieser Voxel kann bei der Schnittpunktabfrage übersprungen werden.

Generierung Die Unterteilungen sowohl des Grids als auch der Line Spaces wird durch Angabe der gewünschten Unterteilung der größten Volumenachse initiiert. Für verbliebenen Seiten S_M und S_K berechnen sich die Unterteilungen $M = \text{Subdiv}_M(N)$ und $K = \text{Subdiv}_K(N)$ mit $\text{Subdiv}_X(N) = N \cdot \frac{l_{S_X}}{l_N}$.

Für jeden Line Space gibt es 30 nutzbare Kombinationen aus Startseite und Endseite, da für jede der sechs Volumenseiten fünf Zielseiten existieren. Dadurch, dass die Startseite nicht als Zielseite verwendet wird, werden degenerierte Shafts ohne Volumeninhalt vermieden. Für jede Start-Ziel-Kombination gibt es darunter eine entgegengesetzte Kombination, somit kann man die Anzahl an zu testenden Kombinationen nochmals auf 15 halbieren, indem man für jeden Shaft sowohl das vorderste, als auch das hinterste zentrale Dreieck bestimmt. Dies halbiert auch die Generierungszeit, weil die Implementation der BVH-Traversierung zur Schnittpunktfindung mit dem Mittelstrahl in der Hinsicht erweiterbar ist.

Algorithmus 2: Line Space Shaft Generierung.

Input: A set of start-end-configurations $C = \{c_0, ..., c_{14}\}$.
Output: An array of all $shafts$ in the current Line Space.

1 **for** *configuration* c_i **in** C **do**
2 $\quad f_{s,i} \leftarrow$ start face in c_i
3 $\quad f_{e,i} \leftarrow$ end face in c_i
4 $\quad o_i \leftarrow o(f_{s,i}, f_{e,i})$
5 $\quad numShafts \leftarrow |c_i|$
6 $\quad$ **for** *index* **in** o_i **to** $o_i + numShafts$ **do**
7 $\quad\quad p_s \leftarrow$ patch at $index/(w_{f_{e,i}} \cdot h_{f_{e,i}})$
8 $\quad\quad p_e \leftarrow$ patch at $index\%(w_{f_{e,i}} \cdot h_{f_{e,i}})$
9 $\quad\quad midRay \leftarrow$ center ray between p_s and p_e
10 $\quad\quad beginIndex \leftarrow o_i + s(p_s, f_{e,i}) + e(p_e)$
11 $\quad\quad shafts[beginIndex \cdot 2] \leftarrow$ nearest intersection of $midRay$
12 $\quad\quad shafts[beginIndex \cdot 2 + 1] \leftarrow$ farthest intersection of $midRay$
13 **return** $shafts$

Die Schleife in Zeile 6 in Algorithmus 2 kann durch die Unabhängigkeit der Schnittpunkttests voneinander pro Kombination parallelisiert auf der GPU ausgeführt werden.

Für diesen Algorithmus sind mehrere essentielle Funktionen implementiert: der Basis-Offset $o(f_s, f_e)$, die Start-Indizierung $s(p_s, f_e)$ und die End-Indizierung $e(p_e)$. Diese Funktionen berechnen sich nach vordefinierten Schemata für eine lineare Indizierung der Patches und Faces auf den Bounding Boxes der Line Spaces. Die Faces sind in einem Array $F = (x_+, x_-, y_+, y_-, z_+, z_-)$ definiert, wodurch sich der Array Index direkt aus einem gegebenen Face berechnen lässt. Für ein gegebenes Face $a_\sigma \in F$ beschreibt a die Achse, entlang der die Normale des Faces zeigt, und σ sagt aus, ob die Normale in die positive $(+)$ oder negative $(-)$ Richtung der Achse zeigt.

Die Shafts für jede Start-End-Kombination (f_s, f_e) werden in jeweils gleich großen Speicherblöcken gespeichert. Zur Indizierung einer dieser Speicherblöcke wird ein basis-Offset $o(f_s, f_e)$ benötigt, der aussagt, wo im gesamten Speicher der erste Shaft dieses Bereichs steht. Dieser berechnet sich durch

$$o(f_s, f_e) = \sum_{m=0}^{c-1} |m|,$$

wobei $c \in (0, ..., 15)$ die derzeitige Konfigurations-ID und $|m|$ die Anzahl von Shafts in der Konfiguration m ist.

Die Struktur innerhalb des Speicherblocks ist nochmals aufgeteilt in kleinere Blöcke, die jeweils für ein Patch p_s alle davon ausgehenden Shafts zu einem festen anderen Face f_e enthalten. Der Offset innerhalb eines gesamten Blocks wird mithilfe der vertikalen und horizontalen Indizes $i_v(p_s)$ resp. $i_h(p_s)$, sowie der Höhenunterteilung des Start-Faces h_{f_s} und den Unterteilungs-Dimensionen des End-Faces w_{f_e} und f_{f_e} durch

$$s(p_s, f_e) = (i_v(p_s) + i_h(p_s) \cdot h_{f_s}) \cdot w_{f_e} \cdot h_{f_e}$$

berechnet. Der Index des Shafts innerhalb des Start-Shaft-Blocks letztendlich ist

$$e(p_e) = i_v(p_e) + i_h(p_e) \cdot h_{f_e}.$$

Schnittpunktabfrage Die Schnittpunktabfrage besteht aus zwei Schritten: der Traversierung des Grids und der Shaft-Indizierung je Line Space. Bei einer Objekt-lokalen Struktur kann ersteres innerhalb einer Schleife über alle Objekte der Szene stattfinden, oder während der Traversierung einer beliebigen darüberliegenden Datenstruktur, die die Bounding Boxes der Objekte enthält. Dafür muss jedes Mal der globale Strahl in einen Objekt-lokalen Referenzstrahl durch Multiplikation der Richtung und des Startpunktes mit der invertierten Modellmatrix transformiert werden. Hierbei kann ein Algorithmus wie in Code 3 gezeigt angewandt werden.

Algorithmus 3: Traversierung des Line Space grids.

Input: A *mesh*, a *ray* relative to the *mesh* transformation and a grid resolution $R = (N_G, M_G, K_G)^T$.
Output: A *hit* containing whether there was any hit, as well as the candidate if so.

```
1  hit ← none
2  voxelSize ← 3D size of a voxel in the current mesh bounding box
3  step ← sign(ray direction)
4  delta ← voxelSize·step / ray direction
5  pos ← (ray origin − min bounds)/voxelSize
6  if ray intersects bounding box of mesh then
7      idx ← clamp pos between 0 and R − 1
8      stepSign ← 3D step component wise greater than 0
9      tnext ← (stepSign + (1 − 2 · stepSign) · (pos − idx)) ∗ delta
10     while idx inside grid do
11         lsIndex ← idx_x + R_x · idx_y + R_x · R_y · idx_z
12         axis ← axis of smallest component in lsIndex
13         tempHit ← hit from line space at lsIndex
14         if tempHit is valid then
15             hit ← tempHit
16             break
17         idx_axis ← idx_axis + step_axis
18         tnext_axis ← tnext_axis + delta_axis
19 return hit
```

Die Berechnung von *tnext* in Zeile 9 zur Vermeidung zusätzlicher if-Abfragen basiert auf einer Zusammenfassung von $a = x$ für $k = n \leq 0$ und $a = 1 - x$ für $k = n > 0$ zu $a = (k + (1 - 2k) \cdot x)$. Zusätzlich dazu kann durch diese Rechnung auf beschleunigte 128-bit 4-Komponenten-Vektor-Rechnung zurückgegriffen werden. Soll nur getestet werden, ob ein Schnittpunkt existiert, kann in Zeile 15 der *hit* zurückgegeben werden. Angelehnt ist der Algorithmus an [uAW87].

Bei der anschließenden Indizierung wird der Element-Index im Buffer entsprechend der Indizierung bei der Generierung berechnet. Dabei werden Start- und Endseite zeitgleich mit dem Schnittpunkttest mit der Bounding Box bestimmt. Die daraufhin benötigten Patch-Indizes können jeweils bei Angabe von Eintrittsseite und Eintrittsposition mit Code 4 berechnet werden. Anschließend können mit einem Strahl-Ebenen-Schnittpunkttest mit dem durch den Shaft indizierten Dreieck die Schnittpunktdaten berechnet werden. Aufgrund der halbierten Seitenkombinationszahl und der sich daraus ergebenden Buffer-Adjazenz von vorderstem und hinterstem Schnittpunkt pro äquivalentem Shaft-Paar, kann für einen Strahl mit invalider Konfiguration eine inverse Konfiguration substituiert werden.

Algorithmus 4: PATCHINDEX in einem Line Space mit Axialen Patch Dimensionen **d**, der Höhenachse **h** und der Breitenachse **v** auf diesem Patch.

Input: The rays entry face f and the 3D entry point vector $\mathbf{p}$ on that face.
Output: A 2D vector containing the patch index.

1 $baseIndices \leftarrow (\langle \frac{\mathbf{p}}{\mathbf{d}}, \mathbf{v} \rangle, \langle \frac{\mathbf{p}}{\mathbf{d}}, \mathbf{h} \rangle)$
2 **if** $f < 3$ **then**
3 | **return** *baseIndices*
4 **else**
5 | $patchMaximum \leftarrow (\langle (N_L, M_L, K_L)^T, \mathbf{v} \rangle - 1, \langle (N_L, M_L, K_L)^T, \mathbf{h} \rangle - 1)$
6 | **return** $patchMaximum - baseIndices$

3 Konzeption

3.1 Implementation

Das Basis-Framework, welches für diese Arbeit genutzt wird, ist mit C++17 implementiert und baut auf OpenGL 4.5 und vor allem auf darin enthaltenem *Direct-State-Access* (DSA) auf. Normalerweise müssen Objekte wie Buffer, Texturen o.Ä., bzw. deren Namen (Ausgedrückt durch einen 32-bit unsigned Integer), an eine begrenzte Anzahl von *Binding Points* gebunden werden, bevor sie verändert oder genutzt werden können. Funktionen, die für DSA implementiert wurden, erhalten anstelle des Binding Points den Namen des Objektes, wodurch viele Operationen vereinfacht werden. Dies ermöglicht es, ohne die Limitierung der Objekt Bindings komplexere Systeme zu erstellen.

Das standardmäßig aktivierte Renderverfahren ist deferred Rendering. Dabei werden Objektinformationen in 6 Texturen, wie in Tabelle 2 beschrieben, gerendert. Anschließend wird diese Menge von Texturen, auch *GBuffer* genannt, in einem weiteren Rendervorgang zu einem Finalen Bild zusammengestellt. Mithilfe weiterer anfolgender Renderdurchgänge sind auch Post-Processing-Effekte möglich.

Tabelle 2: Zieltexturen für das deferred Rendering.

Textur Nr.	Gespeicherte Informationen
0	Basisfarbe und IOR
1	Rauheit, Metalness, Transmissionsfaktor und Emissionsfaktor
2	Oberflächennormale in Weltkoordinaten
3	View-Space-Position
4	World-Space-Position
5	Oberflächennormale in Bildschirmkoordinaten

Grundlegender Aufbau Der Szenenorganisation dient ein komponentenbasierter Szenengraph. Objekte wie Meshes, Lichter oder Kameras werden als Komponenten repräsentiert, die jeweils zu einem Knoten hinzugefügt werden können. Soll eine bestimmte Funktion aufgerufen werden, kann durch den Graphen über einen pre-order-Iterator[15] traversiert werden. Alternativ dazu sind bestimmte Funktionen wie der Render-Aufruf rekursiv definiert, wodurch die Funktion bei jedem Knoten zuerst bei ihm selbst, dann bei allen Komponenten und letztendlich bei allen Kindknoten aufgerufen wird.

Meshes und Lichter werden beim Durchlaufen des Graphen nicht direkt gerendert. Stattdessen werden sie in je eine Render-Liste pro Typ eingefügt und im Anschluss gemeinsam gerendert. Dies hat den Vorteil, dass der G-Buffer, der für

[15] Zuerst wird der Elternknoten besucht. Anschließend werden alle Kindknoten rekursiv durchlaufen.

deferred Rendering genutzt wird, direkten Zugriff auf die Daten hat, um zum Beispiel die Lichter zu aktualisieren. Es ist jedoch auch möglich, den Szenengraph vollständig ohne Farben zu rendern, was beispielsweise in Shadow Maps benötigt wird. Das Aktualisieren von Komponenten kann auch aus externer Quelle durch ein Nachrichtensystem geschehen. Dazu registriert sich ein Objekt beim globalen Nachrichtenempfänger mit einer Nummer und wird beim Erhalt einer Nachricht, die diese Nummer trägt, benachrichtigt. Die Nummer ist ein integraler 32-Bit-Wert und kann infolge dessen auch durch eine Kombination aus vier Buchstaben erzeugt werden.

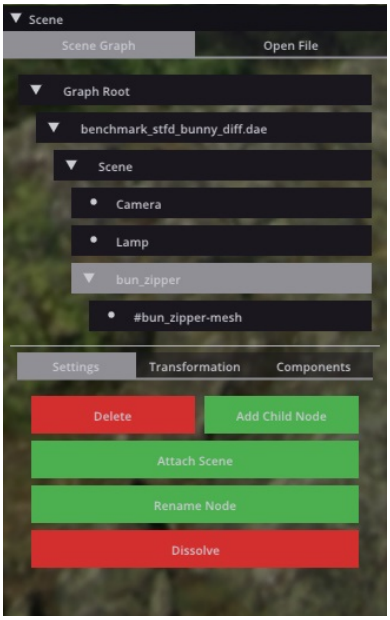


Abbildung 7: Ansicht und Möglichkeiten zur Veränderung des Szenengraphen über ImGui.

Zur Steuerung der Komponenten, sowie zum Ändern des Szenengraphen oder von externen Einstellungen wird ImGui (Abb. 7) mit teils eigenen Erweiterungen wie einem Tab-System, aber auch fremden Erweiterungen benutzt.

Pathtracing Das Pathtracing wird über eine eigene Klasse geregelt. Intern wird ein OpenGL Compute Shader benutzt, welcher anpassbare Work-Group-Größen durch die `GL_ARB_compute_variable_group_size` Erweiterung hat. Als Work-Group bezeichnet man in GPGPU[16]-Anwendungen eine Anzahl an Programmen, die parallel ausgeführt werden.

Es wird mit jedem Renderaufruf immer eine einstellbare Anzahl von Samples gerendert und auf das finale Bild aufgerechnet. Diese inkrementelle Methode erlaubt eine flüssigere Darstellung und die Möglichkeit, sich bei kleinen Szenen aufgrund geringerer Renderzeiten besser umher zu bewegen.

[16]General Purpose GPU Programming

Da die Szene im Ausgangszustand nicht im passenden Format für den Pathtracer vorliegt, ist ein Zwischenschritt nötig. Für diese Aufgabe ist der *Scene Collector* verantwortlich, welcher bei einer Aktualisierung den gesamten Szenengraphen traversiert (siehe Abbildung 8). Je nachdem, welche Komponenten der dabei in jedem Schritt betrachtete Knoten enthält, wird eine eigene *Collector-Einheit* damit beauftragt, diese einzeln zu verarbeiten. Es sind momentan vier Einheiten implementiert: eine für Geometrien und Materialien, und jeweils eine für Lichter, Kameras und den Szenenhintergrund. Letzterer ist meist einzigartig und besteht lediglich aus einer Farbe oder einer Cube-Map.

Tabelle 3: Nachrichtenkategorien bei einer Szenenänderung.

Graph Transformation	Die Transformation eines beliebigen Szenenknoten wurde verändert
Mesh Transformation	Die Transformation eines Szenenknoten, der Geometrie enthält, wurde verändert
Nodes	Der Szenengraph wurde verändert. Es wurden Knoten hinzugefügt, verschoben oder gelöscht
Camera	Die Hauptkamera wurde verändert
Light	Ein beliebiges Licht wurde verändert
Material	Ein beliebiges Material wurde verändert
Skybox	Die Skybox wurde verändert

Für jede dieser Objekte, wie auch für weitere Szenenänderungen, existieren eigene Aktualisierungsnachrichten, die der Collector erhält und weiter verarbeiten kann (Tabelle 3). Anhand dieser Nachrichten können auch nur Teile der gesamten Szene aktualisiert werden. Dies fällt gerade deshalb ins Gewicht, weil die Einheit, die Geometrien verarbeitet, auch die lokalen Datenstrukturen aufbaut. Wird also eine Geometrie deformiert, muss diese Einheit aufgerufen werden um die Strukturen zu aktualisieren, was nicht innerhalb eines Rendervorgangs abgeschlossen werden kann. Das Aufstellen der Datenstrukturen beginnt mit der Generierung der BVH, und wird anschließend mit dem damit aufgebauten Line Space Grids abgeschlossen.

Die Nachrichtenverarbeitung wird über eine *polling*-Methode durchgeführt. Das bedeutet, dass beim Erhalt der Nachricht lediglich ein *Flag* gesetzt wird. Dieses wird beim nächsten Aufruf der implementierten `collect` Methode abgefragt und erst dort werden die zu aktualisierenden Objekte festgelegt. Dies verhindert ein mehrfaches Abarbeiten der gesamten Szene bspw. beim Löschen des Szenengraphen. Am Ende des Abarbeitens wird eine neue Nachricht für den Pathtracer gesendet, durch welche dieser zurückgesetzt wird und er mit dem nächsten Aufruf von Sample 0 an von Neuem rendert. Das Bild, welches der Pathtracer rendert, wird in eine Textur geschrieben, die im Anschluss auf ein bildschirmfüllendes Dreieck gerendert, oder als Bild gespeichert werden kann.

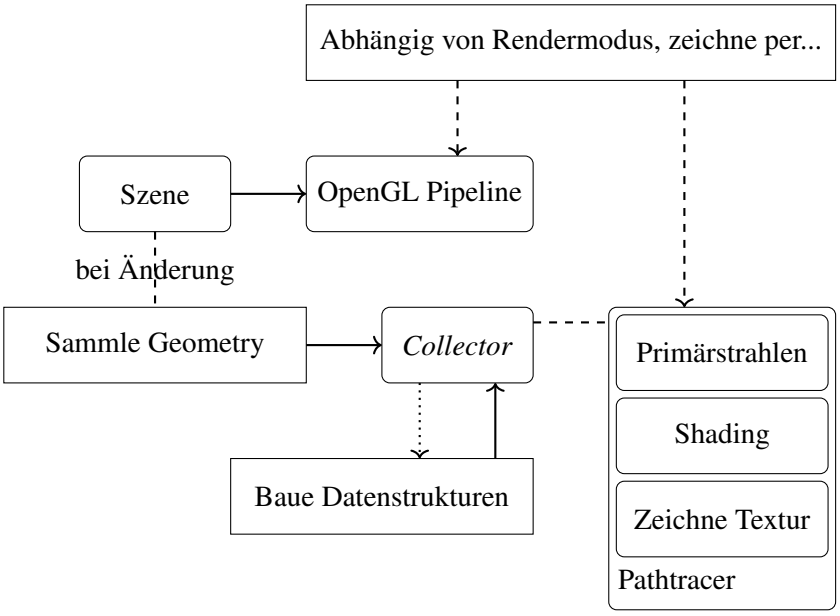


Abbildung 8: Genutzte OpenGL- und Pathtracing-Framework-Struktur. Die *Szene* enthält sämtliche Geometrie, Transformationen, Materialien, Lichter und Kameras und wird über den *Collector* in ein zum Pathtracing geeignetes Format umgewandelt.

Dem Compute Shader liegen die gesammelten Szenendaten in Shader Storage Buffern vor. Dabei wird auch Nutzen von den OpenGL Erweiterungen für *Bindless Graphics* `GL_NV_shader_buffer_load` und `GL_ARB_bindless_texture` gemacht, welche es ermöglicht, die Buffer anstelle von *Binding Points* durch Speicheradressen auszulesen. Diese können auch als 64-Bit-Integer in einem `struct` in einen Buffer geschrieben werden und erlauben somit sehr flexible Strukturen. Bspw. lässt sich dadurch eine Geometrie durch mehrere Buffer-Adressen für Material, BVH-Daten, Line-Space-Daten und Geometrieeckpunkte darstellen.

Auch die Line Space Grids sind so realisiert, dass ein Grid einen Buffer mit Line Spaces enthält, welche jeweils wiederum Buffer für ihre Shafts enthalten. Dazu sind speziell beim Pathtracing Geometrien instanziiert. D.h. ein solches Objekt referenziert lediglich einen geteilten Geometriespeicher, der auch die Datenstrukturen enthält, und speichert zusätzlich dazu instanzspezifische Zusatzinformationen wie das Material oder die Objekttransformation. Um für die Evaluation eine Selektion der genutzten Datenstruktur zu ermöglichen, muss es eine Schnittstelle zwischen Schnittpunktabfrage und Schnittpunkttest geben. Diese bekommt als Eingabe einen Strahl und Informationen zur derzeitigen Rendersituation (z. B. die derzeitige Anzahl von Bounces oder die bisher aufmultiplizierte PDF). Der Rückgabewert ist ein *Hit* Objekt, welches neben den Geometrie- und Dreiecksindizes auch zwei der drei baryzentrischen Koordinaten enthält. Dadurch können Schnittpunkte mit minimalem Speicher ohne Genauigkeitsverlust abgefragt werden.

Bounding Volume Hierarchy Der in dieser Arbeit genutzte BVH basiert auf einer beschleunigten Version der *Surface Area Heuristic*. Der Algorithmus für die Maßfindung läuft pro Achse a wie folgt ab. Zuerst werden die Kandidaten in N auf der Achse gleich verteilte *Bin*s verteilt. Die Indizierung der Bins geschieht pro Kandidat durch dessen *Zentrumsposition* c, aufgrund der geringeren einzuordnenden Menge im Vergleich bspw. zu Dreiecken. Berechnet wird der Index durch $i = k \cdot (c - \min_{\text{bounds}}(a))$, mit $k = \frac{N}{\text{size}_{\text{bounds}}(a)}$. Anschließend werden für jedes k die bins aufgeteilt, sodass die Kostenfunktion

$$C(K) = A_{<k} \cdot n_{<k} + A_{\geq k} \cdot n_{\geq k}$$

minimal wird. Hierbei bezeichnet $A_{<k}$ den Flächeninhalt der durch die Kandidaten der Bins $\{0,...,k-1\}$ aufgespannten Bounding Box. $A_{\geq k}$ wird äquivalent durch die Kandidaten der Bins $\{k,...,N-1\}$ berechnet. Sowohl $A_{<k}$ als auch $A_{\geq k}$ werden nochmals mit der Anzahl der Kandidaten $n_{<k}$ und $n_{\geq k}$ gewichtet. Die Achse und Ebene mit den geringsten Kosten bestimmt, wie die Kandidaten am Ende aufgeteilt werden.

3.2 Datenstrukturselektion

Es gilt zu untersuchen, welche Situationen durch Line Space Nutzung einen Leistungsgewinn bei einer Qualität bieten, die einem nur mit der BVH gerenderten Bild nahekommt. Dabei sind Phasen beim Pathtracing vorteilhaft, in denen Ungenauigkeiten vernachlässigt werden können. Die verschiedenen Selektionsmaße können durch eine dedizierte GUI (Abbildung 9) dynamisch eingestellt werden.

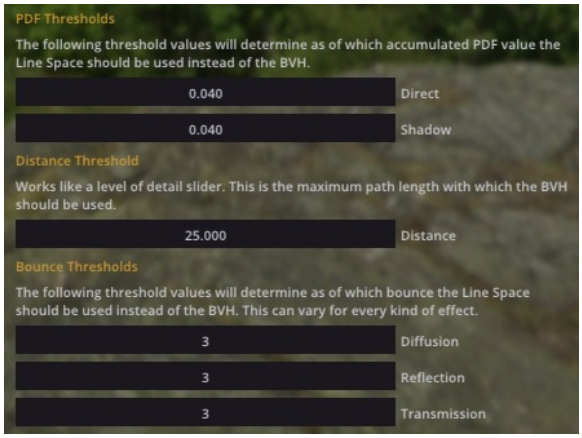


Abbildung 9: GUI mit allen Einstellungen zur Line Space Kontrolle.

Distanz-Grenzwert Einen Grenzwert bei fernen Objekten zu haben, um den Detailgrad zur Erhöhung der Performanz zu reduzieren, findet man heutzutage in der Spieleentwicklung und Szenenkomposition wieder. Dort auch unter dem

Begriff *Level of Detail (LOD)* bekannt, kann dies auf verschiedene Arten genutzt werden. So wird dazu bspw. die Geometrieauflösung bei weit entfernten Objekten verringert. Anders herum kann es aber auch für Texturen genutzt werden, um bei naher Betrachtung einen höheren Detailgrad zu gewährleisten.

Im Kontext der distanzabhängigen Datenstrukturwahl werden mit jeder durchlaufenen Indirektion jedes Pfades die jeweiligen überbrückten Distanzen aufaddiert. Übersteigt die Pfadlänge den Grenzwert, kann, durch eine kleinere Darstellung des Objekts und einen daraus folgenden niedrigeren Bedarf an Genauigkeit, anstatt des BVHs der Line Space für die Schnittpunktabfrage verwendet werden.

Wie auch bei den anfangs genannten Anwendungsfällen kann diese Betrachtung auch hier bei größeren Szenen von Vorteil sein. So können durch die Instanziierung bspw. viele Baummodelle im Hintergrund dargestellt werden, wobei die Rendergeschwindigkeit weniger beansprucht wird, als würde man die gesamte Szene mit BVHs rendern. Liegt eine kleine Szene mit wenig Objekten vor, ist kein hoher Geschwindigkeitsvorteil aus diesem Ansatz zu erwarten.

Wahrscheinlichkeitsdichte Die Wahrscheinlichkeitsdichtefunktion aus 2.1.3 ist ein elementarer Bestandteil des Pathtracing, um eine Aussage darüber zu treffen, wie wichtig ein gegebener Strahl für das Gesamtbild ist. So haben raue Materialien eine geringere PDF als hoch reflektierende. Da gerade bei rauen Reflexionen und Transmissionen eine Streuung der Strahlen in einer Objektunschärfe resultiert, ist es dort nicht vonnöten, genaue Kantendarstellung zu haben. Folglich kann früher im Pfad über den Line Space gerendert werden.

Neben der direkten Abhängigkeit von der PDF des jeweils aktuellen Schnittpunkts, kann eine stärkere Gewichtung erlangt werden, indem die gesamten PDF-Werte des Pfades aufmultipliziert betrachtet werden. So kann bei rauen Materialien nochmals früher der Line Space genutzt werden, wohingegen bei glatten Materialien eine höhere Genauigkeit gewährleistet wird. Diese letztere Variante wird für diese Arbeit getestet und ist einstellbar durch einen invertierten *Qualitätswert*. Eine Qualität von 1, äquivalent zu einem PDF-Grenzwert von 0, bedeutet demnach, dass immer mit der BVH gerendert wird, wohingegen bei 0 nach den Primärstrahltests nur noch der Line Space benutzt wird, da jede PDF kleiner oder gleich dem Grenzwert 1 ist. Dazu gibt es eine Unterscheidung zwischen Schnittpunkttests von Schattenstrahlen und solchen von Sekundärstrahlen.

Effekte Anhand der BSDFs aus Abschnitt 2.1.4 kann zusätzlich eine Unterscheidung für jeden der drei Rendereffekte getroffen werden. Hier geschieht dies durch einen festen Bounce-Anzahl-Grenzwert. Liegt dieser unter dem Effektabhängigen globalen Grenzwert, der die Gesamtanzahl von Bounces pro Pfad beschränkt, so wird ab dann für den Rest des Pfades direkt der Line Space für den Schnittpunkttest gewählt, unabhängig von den bisher genannten Kriterien. Somit ist es bspw. möglich, in Bereichen, in denen vorherige Grenzwerte nicht greifen, weiteren Leistungsgewinn zu erreichen.

4 Evaluation

Die Evaluation der verschiedenen Grenzwertkonzepte ist durch die hohe Anzahl von verschiedenen Konfigurationen aufgrund der kombinatorischen Zusammenarbeit der Methoden nicht in vollem Umfang möglich. Daher werden spezielle Fälle ausgewählt, die repräsentativ eine Gesamtheit der Effizienz der Methoden darstellen können. Für jeden Effekt werden verschiedene Bounce-Thresholds getestet, die im Allgemeinen eine Übersicht über die Leistungsunterschiede geben können. Dazu wird auch ein Vergleich zwischen verschiedenen Polygonmengen pro Objekt gezogen. Als Modelle dienen zum einen der *Stanford Bunny* (69k Dreiecke) und der *Stanford Dragon* (1.132.830 Dreiecke) aus der 3D-Scan Repository des Computer Graphics Laboratory der Stanford University[17], zum anderen das *Sponza*-Modell von Crytek[18] als Architekturelle Testszene mit 262267 Polygonen. Bei allen Tests wurde in HD (1280x720 Pixel) auf einer NVidia Titan X, einem Intel Core i7 6700 und mit 32GB RAM gerendert. Die Line Spaces hatten eine Auflösung von 8x8x8.

4.1 Diffuse Bounce-Grenzwerte

Zuerst wird die Abhängigkeit der Performanz von einem Diffusen Bounce-Grenzwert und weiterhin auch der Einfluss der Schnittpunktsuche beim Schattentest getestet. Das Modell ist der hoch aufgelöste Stanford Dragon. Die Ergebnisse zeigen merkliche zeitliche Unterschiede zwischen Line-Space- und BVH-Schattentests. Im Vergleich zum Unterschied bei Sekundärstrahlen ist dieser jedoch hinnehmbar, sollte ein Fokus auf korrekte Schattendarstellung liegen. Der Schatten bei direkter Nutzung des Line Space (Abb. 10a) weist harte Kanten und eine starke Artefaktbildung auf. Es ist daher nur dann lohnenswert, wenn entweder die Lichtquellen einen großen Radius haben und somit die Schatten sehr weiche Ränder haben, oder wenn die Objekte weit entfernt sind. Beim Shading ist der Line Space früher nutzbar als beim Schatten, da die Diffuse Streuung harte Kanten vermeidet und somit die Artefaktbildung überdecken kann. Die BVH alleine (Abb. 10f) ist in diesem Beispiel unter Betrachtung der Renderzeiten nicht lohnenswert, da sie nur etwa 50% der Leistung aufweist. Die größten Unterschiede finden sich in sich selbst verschattenden Arealen wie dem Maul des Drachen.

[17] `http://graphics.stanford.edu/data/3Dscanrep/`
[18] `http://www.crytek.com/cryengine/cryengine3/downloads`

(a) Nur Line Space

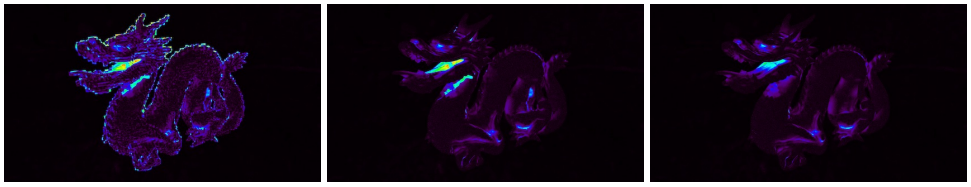
(b) Primärstrahlen mit BVH, Indirektionen mit Line Space

(c) Primärstrahlen und Schatten mit BVH, sonst Line Space

(d) Nur BVH

Bild	Samples pro Sekunde	Millisekunden pro Sample	MRays/s
10a	47.1	21.2	93
10b	30.3	32.9	73
10c	21	47.6	59
10d	9.7	103.7	20

(e) Renderzeiten für 1500 Samples.



(f) Unterschiede zwischen BVH-Referenzbild 10d und Line-Space-Intermediärbildern (vlnr.) 10a, 10b und 10c.

Abbildung 10: Vergleich beim Rendern von diffusen Materialien. Modell: Stanford Dragon (1.132.830 Dreiecke). Line Space Grid Auflösung: 4x8x6.

4.2 Reflexive Bounce-Grenzwerte

Im Gegensatz zu diffusen Materialien bilden sich auf metallischen Materialien keine Schatten. Daher wird eine explizite Behandlung von Schattenstrahlen in diesem Abschnitt außen vor gelassen.

Grundlegend weisen Materialien mit metallischen Reflexionen ein Verhalten ähnlich dem von diffusen Materialien auf. So sind die Artefakte des Line Space eher im Vergleich sichtbar, da keine sichtbaren harten Kanten entstehen. Ein durch den Line Space hervorgerufener Fehler ist aber bei Selbstreflexionen im Objekt ersichtlich. Bspw. sind diese Artefakte beim Drachen in den Kanten der Beine, sowie zwischen den Windungen sichtbar.

Im Kontext dieser Evaluation wurden die Bilder 11a, 11b und 11c mit aufsteigenden Bounce-Grenzwerten gerendert. Letzteres dient dabei als Referenzbild, da bei höherem Grenzwert kein signifikanter Unterschied erkennbar ist.

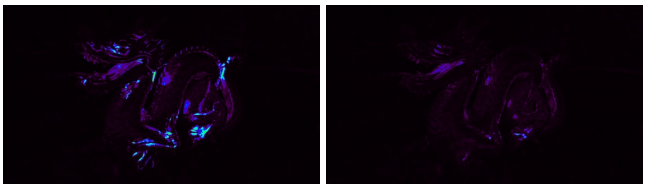
(a) Primärstrahlen mit BVH, Indirektionen mit Line Space

(b) Line Space ab dem zweiten Bounce

(c) Line Space ab dem dritten Bounce

Bild	Samples pro Sekunde	Millisekunden pro Sample	MRays/s
11a	27	37.1	58
11b	16.4	60.7	30
11c	9.5	104.5	20

(d) Renderzeiten in HD (1280x720) auf einer NVidia Titan X, Intel Core i7 6700, 32GB RAM, mit 4x8x6 Grid aus 8x8x8 Line Spaces für 1500 Samples.



(e) Unterschiede zwischen Referenzbild 11c und Line-Space-Intermediärbildern (vlnr.) 11a und 11b.

Abbildung 11: Vergleich beim Rendern von reflektierenden Materialien (Rauheit 0.092). Modell: Stanford Dragon (1.132.830 Dreiecke). Line Space Grid Auflösung: 4x8x6.

Die Unterschiede zwischen den Bildern 11a und 11c sind zwar sichtbar, aber dennoch ist ersteres immernoch geeignet, wenn die Renderzeit wichtiger als die Qualität ist. Der Geschwindigkeitsvorteil liegt knapp bei einem Faktor von 3 bei

dahingehend vernachlässigbaren qualitativen Unterschieden. Spielt die Qualität eine übergeordnete Rolle, so eignet sich Bild 11b durch eine geringere Renderzeit ebenso eher als Bild 11c. Die gewählte Materialrauheit spielt ebenso eine Rolle bei der Vermeidung von Artefakten. Je rauer das Material, desto weniger sichtbar sind Artefakte in reflektierten Objekten. Schon eine Rauheit von $0,0092$ reicht aus, um den Großteil der Reflexionsartefakte zu vermeiden.

4.3 Transmissive Bounce-Grenzwerte

Bei transmittierenden Bounces kommt im Vergleich zu anderen Materialeigenschaften hinzu, dass allgemein mehr Bounces benötigt werden, um schwarze Flächen zu vermeiden. Diese entstehen dadurch, dass die Pfade innerhalb des Objekts aufhören.

(a) Primärstrahlen mit BVH, Indirektionen mit Line Space

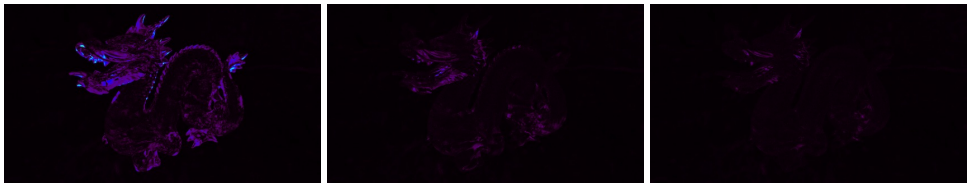
(b) Line Space ab dem zweiten Bounce

(c) Line Space ab dem dritten Bounce

(d) Line Space ab dem vierten Bounce

Bild	Samples/s	ms/Sample	MRays/s
12a	23	43.5	69
12b	14	71	42
12c	9.5	105.3	28.5
12d	8.3	120.6	25

(e) Renderzeiten in HD (1280x720) auf einer NVidia Titan X, Intel Core i7 6700, 32GB RAM, mit 4x8x6 Grid aus 8x8x8 Line Spaces für 1500 Samples.



(f) Unterschiede zwischen Referenzbild 12d und Line-Space-Intermediärbildern (vlnr.) 12a, 12b und 12c.

Abbildung 12: Vergleich beim Rendern von transmittierenden Materialien (Rauheit 0.092). Modell: Stanford Dragon (1.132.830 Dreiecke). Line Space Grid Auflösung: 4x8x6.

Da sie dabei weder den Hintergrund noch ein anderes diffuses Objekt treffen, wird für diese Pfade die Farbe schwarz in das Bild geschrieben, da das Glasmaterial selber, inklusive der vollständigen Reflexionen durch flaches Auftreffen der Strahlen,

keine Lichtfarbe empfängt.

Somit ist Bild 12b das Optimum bezüglich der Zeit pro Qualität. Für Vorschauzwecke oder performantes Rendering ist jedoch auch Bild 12a schon nutzbar, da die Unterschiede nur im Vergleich mit Bildern einer höheren Qualität sichtbar sind.

Es zeigt sich ein ähnliches Verhalten wie bei den anderen Bounce-Typen auf. Wie schon bei der Reflexion empfängt das implementierte Glasmaterial keine Schatten und weist dahingehend somit keine Nachteile auf. Signifikante Unterschiede lassen sich lediglich in Bezug auf Bild 12a feststellen, bei dem der Line Space direkt nach dem direkten Shading benutzt wird. Mit mehr Bounce-Grenzwerten können die Unterschiede vernachlässigt werden.

4.4 PDF-Abhängigkeit

Das Ziel der folgenden Szene ist es, die Abhängigkeit der Line Space Nutzung von der PDF, aber auch von der Polygonanzahl des Objektes, zu analysieren. Der Stanford Bunny hat mit etwa 69.000 Dreiecken nur ungefähr 6% verglichen mit der Anzahl von Polygonen im Stanford Dragon. Der erste Test bestand daraus, erneut zu prüfen, wie stark sich die Datenstrukturen beim Rendern von Schatten unterscheiden. Dabei wurde das Modell einmal mit BVH- und einmal mit Line Space Schatten gerendert. Die Bilder 13a und 13b zeigen geringe Unterschiede, wie sich in 13c erkennen lässt.

Dies sind die gleichen Schattenartefakte, wie sie auch schon in Abschnitt 4.1 aufgetreten sind. Jedoch lassen sich in diesem Durchlauf keine zeitlichen Unterschiede feststellen, sodass für direkte Schatten der BVH in Bild 13a bessere Ergebnisse liefert. Neben der Polygonabhängigkeit in den Schattentests wurden vier weitere Bilder mit verschiedenen PDF-Grenzwerten gerendert. Wie in 13e bis 13h zu sehen ist, unterscheiden sich verschiedene Grenzwerteinstellungen nur wenig voneinander. Die Unterschiede entstehen, wie schon beim Dragon, besonders in den Kantenbereichen des Modells, stechen jedoch nicht so stark hervor. Die Differenzbilder 13i bis 13k zeigen nur geringe Unterschiede, die bei Bedarf an höherer Rendergeschwindigkeit vernachlässigt werden können. Der Zeitvorteil des Line Space liegt bei 26% bei einem Grenzwert von 0.2 gegenüber 1.0. Bedarf es nun einer hohen Rendergeschwindigkeit, bei der eine geringe qualitative Reduktion hinnehmbar ist, empfiehlt es sich somit meistens, einen eher geringeren Grenzwert bei der Auswahl zu nutzen.

(a) Primärstrahlen und Schatten mit BVH, Indirektionen mit Line Space

(b) Primärstrahlen mit BVH, Indirektionen und Schatten mit Line Space

(c) Unterschiede zwischen beiden Methoden

Bild	Samples pro Sekunde	Millisekunden pro Sample	MRays/s
13a	54.2	18.4	82
13b	54.2	18.4	82
thesh. $= 0.2$	54.2	18.4	82
thesh. $= 0.8$	48.4	20.7	72
thesh. $= 0.9$	45.3	22.1	67
thesh. $= 1.0$	40.3	24.8	59

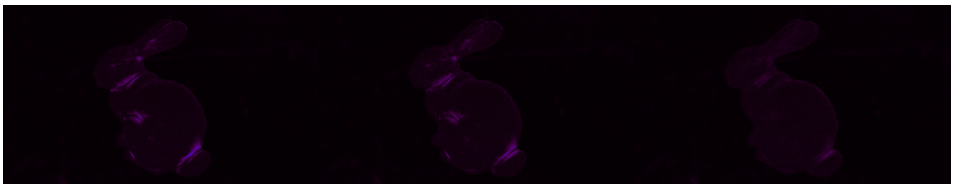
(d) Renderzeiten für 1500 Samples.

(e) PDF-Grenzwert 0,2

(f) PDF-Grenzwert 0,8

(g) PDF-Grenzwert 0,9

(h) PDF-Grenzwert 1,0



(i) Differenz 13e und 13h

(j) Differenz 13f und 13h

(k) Differenz 13g und 13h

Abbildung 13: Vergleich beim Rendern von diffusen Materialien zwischen dem Referenzbild 13h und (vlnr.) 13f, 13g und 13h. Modell: Stanford Bunny (69k Dreiecke). Line Space Grid Resolution: 8x7x8.

4.5 Architekturelle Szenen

Die Schwierigkeit beim Rendern von architekturellen Szenen besteht in der hohen Anzahl von möglichen Indirektionen der Pfade. Speziell bei der Sponza-Szene entstehen diese besonders hinter den Vorhängen in der unteren Etage und fordern mehr Leistung des Renderers. Ein weiteres Problem ist die Aufteilung der Objekte und die damit einhergehenden Ungenauigkeiten beim uniform aufgeteilten Line Space . Dadurch können sich bei zu geringer Auflösung Artefakte zwischen nah beieinander liegenden Objekten, wie den Bannern in Sponza, bilden (Siehe Abb. 14). Globale nichtrekursive Datenstrukturen wie das Uniform Grid hätten zusätzlich noch den Nachteil, dass viele leere Räume existieren, die dennoch traversiert werden müssen. Aus diesen Gründen ist es wiederum besonders wichtig, diese Szenen in Geschwindigkeitstests mit einzubeziehen. Zu Testzwecken wurde ein PDF-Grenzwert von 0.7 bei maximal 3 BVH-Bounces von 6 Gesamtbounces kombiniert. Dazu liegt die Distanzgrenze bei ungefähr der Hälfte der Szenentiefe. Diese Einstellungen sollen im Nahfeld dafür sorgen, dass durch den BVH mehr Details sichtbar sind und schärfere Reflexionen dargestellt werden, während weiter entferntere Objekte ungenauer gerendert werden sollen.

Samples pro Sekunde	Millisekunden pro Sample	MRays/s
0.94	1065	13

Abbildung 14: Sponza, gerendert mit einem PDF Grenzwert von 0.7, einer Distanzgrenze von 50m (ca. die Hälfte der Szenendimension) und einem diffusen Bounce-Grenzwert von 3 für 6 Globale Bounces insgesamt. Die Renderzeiten entsprechen 100 Samples.

Das entstandene Bild weist eine starke Artefaktbildung auf, die auf einen zu geringen Line Space Bias zurückzuführen sein könnten. Dieser Bias ist verantwortlich dafür, wie weit der Strahlursprung beim Test gegen einen Line Space entgegen der Richtung verschoben wird, um durch Bounds-Schnittpunkttests einen Eintrittspunkt für die Shaft-Indizierung zu berechnen.

Auch die Geschwindigkeit ist verglichen an den MRays/s Messwerten gegenüber den anderen Szenen niedrig ausgefallen und hat sich nach weiteren Tests nicht

signifikant verändert. Grund dafür könnte die Objektlokale Datenstruktur sein, die über die naive Methode, jedes Objekt einzeln zu testen, implementiert wurde. Dies kann dazu führen, dass die Tests einzelner Objekte insgesamt länger dauern als die Tests in den Datenstrukturen der Objekte.

4.6 Generierungszeit

Trotz der Tatsache, dass die BVH auf der CPU und der Line Space auf der GPU konstruiert wird, ist sowohl die Generierungszeit, als auch der Speicherverbrauch des Line Space weit höher als der der BVH. Dadurch, dass ein vorheriger Aufbau der BVH vonnöten ist, um den Line Space zu generieren, ist ein direkter Vergleich dahingehend nicht sinnvoll. Dennoch bleibt dieser Nachteil immernoch ein wichtiger Entscheidungsgrund bei der Auswahl der Datenstruktur, wenn der Fokus auf einer dynamischen Generierung einer solchen liegt. Tabelle 4 zeigt die deutlichen Unterschiede, die zwischen einem Faktor 20 und 50 liegen, auf.

Modell	**Line Space**	**BVH**
Stanford Bunny	18.805	0.363
Stanford Dragon	85.326	4.606

Tabelle 4: Generierungszeiten in Sekunden. Zeit des Line Space zusätzlich zur vorher benötigten BVH-Generierung.

Im Kontext des Speicherverbrauches sind die Nachteile des Line Space ebenso deutlich. Belegt der BVH etwa weniger Speicher bei kleineren Meshes, ist der Line Space unabhängig von der Polygonanzahl und behält bei durchschnittlichen Auflösungen einen vergleichsweise hohen Speicherverbrauch.

Modell	**Line Space Grid Auflösung**	**Line Space**	**BVH**
Stanford Bunny	8x7x8	220.2	4.066
Stanford Dragon	4x8x6	94.371	47.958

Tabelle 5: Speicherverbrauch der Datenstrukturen in MB.

5 Fazit

In der Evaluation wurden verschiedene Schwerpunkte bei den Selektionsmethoden gesetzt, die jeweils andere Aspekte der Datenstrukturen beleuchten sollten. Neben dem allgemeinen Distanzgrenzwert, der auf Basis der Pfadlänge eine Auswahl trifft, wurde sowohl allgemein, als auch spezifisch über effektbezogene Auswahlmethoden selektiert. Dazu dienten drei verschiedene Szenarien der Szenengestaltung als Grundlage: Objekte mit einer geringen, sowie solche mit einer hohen Polygonzahl und eine architekturelle Szene.

Die Tests der verschiedenen allgemeinen effektbezogenen Methoden, bei denen ab einem festen Bounce-Grenzwert der Line Space genutzt wird, zeigen Resultate, die zum Großteil für eine Nutzung des Line Space sprechen. Vor allem bei nicht-diffusen Materialien, die keine Schatten erhalten, kann eine Mehrleistung von zwischen 50% und 150% erwartet werden, ohne einen hohen Qualitätsverlust zu erhalten. Da Schatten nur bei größeren Meshes eine merkliche Leistungsverringerung zur Folge haben, kann im Normalfall dieser über die BVH gerendert werden, um keine stark sichtbaren Artefakte zu erhalten.

Weiterhin ist auch die indirekte und von der PDF abhängige Effektabhängigkeit vorteilhaft. Dadurch ist wieder ein Leistungszuwachs von bis zu 40% möglich. Die letzte, distanzgewichtete Methode ist eher unterstützend und kann bei weit entfernten Objekten Anwendung finden. Dadurch, dass dabei ein absoluter Grenzwert gesetzt wird, haben hinter der gesetzten Distanz andere Gewichtungen keine Einwirkungen mehr. Durch die Natur der Testmethoden wirken die Leistungsdifferenzen nicht additiv, es bedarf somit eines Abwägens der einzelnen Methoden, um eine hinsichtlich der Rendergeschwindigkeit optimale Kombination zu finden. Dies ist abhängig von der zu rendernden Szene.

Anzumerken bleibt weiterhin die lange Generierungszeit des Line Space, die wiederum nochmals abhängig von einem vorherigen Aufbau der BVH ist. Dadurch ist eine dynamische Neugenerierung noch nicht in Echtzeit möglich.

5.1 Anwendungsmöglichkeiten

Die Ergebnisse zeigen einen Nutzwert bei Szenen, die viele Instanzen des selben Meshes enthalten, wodurch die Generierungszeit und vor allem der Speicherverbrauch insgesamt nicht zu stark ins Gewicht fällt. Aufgrund der Selbstreflexion von Objekten und der dabei entstehenden Artefakte ist der Ansatz in der derzeitigen Form mehr für die Nutzung bei konvexen Meshes geeignet. In Räume, die als ein vollständiges zusammenhängendes Mesh vorliegen, sind durch die uniforme Voxelauflösung Artefakte wahrscheinlicher, wie sich an den unregelmäßigen Oberflächen in Sponza zeigen lässt.

Durch den Distanzgrenzwert können auch weit entfernte Objekte schnell gerendert werden, da die entstehenden Artefakte je nach Unterteilung kleiner als die Pixel sind, die das Objekt einnimmt. Dadurch entsteht eine Art Level of Detail. In Kombination mit den Mesh-Instanzen kann dies bspw. ein Wald sein, bei dem die

Bäume einzeln rein mit dem Line Space gerendert werden können und nur durch Skalierungen und Rotationen schon zufällig verteilt aussehen können.

Allgemein bestätigt dies die Nutzbarkeit des Ansatzes in bestimmten Situationen. Gerade dann, wenn man schnell eine Art Vorschaubild rendern will, bei dem Qualität eine untergeordnete Rolle im Vergleich zur Renderzeit spielt. Im Falle der Schattendarstellung ist der Line Space nur selten von Vorteil, bspw. wenn die Lichtquelle einen großen Radius hat und die Schatten sehr weiche undefinierte Kanten haben, bei denen die Artefakte nicht sichtbar sind.

5.2 Voraussicht

Die Implementation ist nur eine Art allgemeines *Proof-of-Concept* und nicht ein hinsichtlich der Geschwindigkeit optimiertes Framework. Hier wäre eine sehr spezialisierte nur für die Tests ausgelegte Basis vorteilhaft. Damit einhergehend ist die Implementation der BVH nicht in jeder Hinsicht dem aktuellen Stand der Technik entsprechend nach der besten Methode aufgebaut, da auch dabei eine gewisse Generalisierung im Framework-Kontext zur Anpassung an die Gegebenheiten vonnöten war.

Da das Framework ursprünglich lediglich für die normale Rasterisierung angedacht war und somit der Aspekt des Raytracings erst im Nachhinein implementiert wurde, ist die Schnittstelle zwischen dem Szenengraphen und den Datenstrukturen eher hinderlich. Hier gäbe es objektlokale Ansätze, wie bspw. die Speicherung der Datenstrukturen innerhalb eines Buffers, der einen geteilten Speicherbereich für Mesh Indizes und Vertices einschließt. Neben der allgemeinen Beschleunigung durch einen geringeren Bedarf an Aktualisierungs-Callbacks, kann somit vielleicht zusätzlich eine Speicherkohärenz positiven Einfuss auf die Rendergeschwindigkeit haben.

Dynamische Szenen sind auch nur hinsichtlich der Transformationen der gesamten Objekte möglich. Verformbare Meshes sind durch die lange Generierungszeit des Line Space nicht sinnvoll. Dahingehend würden schnellere Generierungsmethoden vorteilhaft sein, sind aber allein durch die hohe Menge von Shafts schwer zu finden.

Bei den Tests selber könnte auch ein zufallsbasierter fließender Distanzwert interessante Ergebnisse aufzeigen, da die Grenze zwischen Line Space und BVH nicht mehr so stark sichtbar wäre.

Literatur

[Áfr14] A. T. Áfra und L. Szirmay-Kalos. *Stackless Multi-BVH Traversal for CPU, MIC and GPU Ray Tracing*. Computer Graphics Forum, vol. 33, 129–140. Wiley Online Library, 2014.

[Ben75] J.-L. Bentley. *Multidimensional binary search trees used for associative searching*. Communications of the ACM, vol. 18, no. 9, 509–517, September 1975.

[Bit13] J. Bittner, M. Hapala und V. Havran. *Fast insertion-based optimization of bounding volume hierarchies*. Computer Graphics Forum, vol. 32, 85–100. Wiley Online Library, 2013.

[Bli77] J. F. Blinn. *Models of light reflection for computer synthesized pictures*. ACM SIGGRAPH Computer Graphics, vol. 11, 192–198. ACM, 1977.

[Coo82] R. L. Cook und K. E. Torrance. *A reflectance model for computer graphics*. ACM Transactions on Graphics (TOG), vol. 1, no. 1, 7–24, 1982.

[Ern08] M. Ernst und G. Greiner. *Multi bounding volume hierarchies*. IEEE/EG Symposium on Interactive Ray Tracing 2008, 35–40. IEEE, 2008.

[Ham60] J. M. Hammersley. *Monte carlo methods for solving multivariable problems*. Annals of the New York Academy of Sciences, vol. 86, no. 3, 844–874, 1960.

[Hav04] H. J. Haverkort. *Results on geometric networks and data structures*. PhD thesis, 2004.

[Jev88] D. Jevans und B. Wyvill. *Adaptive voxel subdivision for ray tracing*. University of Calgary, 1988.

[Kaj86] J. T. Kajiya. *The rendering equation*. SIGGRAPH Computer Graphics, vol. 20, no. 4, 143–150, August 1986.

[Ken08] A. Kensler. *Tree rotations for improving bounding volume hierarchies*. IEEE/EG Symposium on Interactive Ray Tracing 2008, 73–76. IEEE, 2008.

[Keu17] K. Keul, N. Klee und S. Müller. *Soft shadow computation using precomputed line space visibility information*. Václav Skala-UNION Agency, 2017.

[Lam92] J. H. Lambert. *Lamberts Photometrie*, vol. 1. ursp. *Photometria, sive De mensura et gradibus luminis, colorum et umbrae*, 1760. W. Engelmann, 1892.

[Lau09] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke und D. Manocha. *Fast BVH construction on GPUs*. Computer Graphics Forum, vol. 28, 375–384. Wiley Online Library, 2009.

[Pan10] J. Pantaleoni und D. Luebke. *HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry*. Proceedings of the Conference on High Performance Graphics, 87–95. Eurographics Association, 2010.

[Pho75] B. T. Phong. *Illumination for computer generated pictures*. Communications of the ACM, vol. 18, no. 6, 311–317, 1975.

[PJH16] M. Pharr, W. Jakob und G. Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[Sch93] C. Schlick. *A customizable reflectance model for everyday rendering*. Fourth Eurographics Workshop on Rendering, 73–83. Paris, France, 1993.

[Sti09] M. Stich, H. Friedrich und A. Dietrich. *Spatial splits in bounding volume hierarchies*. Proceedings of the Conference on High Performance Graphics 2009, 7–13. ACM, 2009.

[uAW87] J. Amanatides und A. Woo. *A fast voxel traversal algorithm for ray tracing*. In Eurographics '87, 3–10, 1987.

[uJSuMH10] M. Seiler, J. Spillmann und M. Harders. *A threefold representation for the adaptive simulation of embedded deformable objects in contact*. Journal of WSCG, vol. 18, no. 1-3, 89–96, February 2010.

[Wal07a] I. Wald. *On fast construction of sah-based bounding volume hierarchies*. IEEE Symposium on Interactive Ray Tracing, 2007, 33–40. IEEE, 2007.

[Wal07b] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. *Microfacet models for refraction through rough surfaces*. Proceedings of the 18th Eurographics Conference on Rendering Techniques, EGSR'07, 195–206, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.