

**F A C H H O C H S C H U L E W E D E L**  
- University of Applied Science -

**BACHELOR-THESIS**

in der Fachrichtung  
Medieninformatik

Thema:

**Machine Learning acceleration in Raytracing**

Eingereicht von: Jonas Sorgenfrei  
Minf101767  
Op de Wisch 1  
25436 Moorrege  
Tel. (+49) 04122 / 83420  
E-Mail: minf101767@fh-wedel.de

Erarbeitet im: 7. Semester

Abgegeben am: 19. August 2019

Referent (FH Wedel): Prof. Dr. Christian-A. Bohn  
Fachhochschule Wedel  
Feldstraße 143  
22880 Wedel  
Tel. (+49) 04103 / 804840  
E-Mail: bo@fh-wedel.de



# **Machine Learning acceleration in Raytracing**

*Forschungsarbeit zur Beschleunigung des Raytrace-Prozesses durch maschinelles Lernen*

2019, Bachelor-Thesis von Jonas Sorgenfrei, Fachhochschule Wedel

## **Zusammenfassung:**

*In dieser Arbeit wird die Anwendbarkeit und Effizienz von Modellen des maschinellen Lernens auf den geometrischen Abschnitt des Raytracing/Pathtracing untersucht.*

*Es wird erforscht, welchen Einfluss ein maschinell trainiertes Modell auf den Raytrace-Algorithmus nehmen kann und welche Genauigkeit dieses zurückliefert. Im Anschluss wird diskutiert, ob sich anhand von Zeit und Genauigkeit eine praktische Anwendbarkeit eines solchen Modells im Raytrace-Prozess realisieren lassen würde und welche Vor- und Nachteile daraus entstehen könnten.*



# Inhaltsverzeichnis

Abbildungsverzeichnis .....	VII
Listing-Verzeichnis .....	IX
Tabellenverzeichnis .....	X
Abkürzungsverzeichnis .....	XI
<b>1. Einleitung.....</b>	<b>1</b>
<b>1.1. Motivation.....</b>	<b>1</b>
<b>1.2. Zielsetzung .....</b>	<b>5</b>
<b>1.3. Struktur.....</b>	<b>7</b>
<b>2. Grundlagen.....</b>	<b>8</b>
<b>2.1. Raytracing .....</b>	<b>8</b>
<b>2.2. PBRT Rendering System .....</b>	<b>14</b>
<b>2.3. Acceleration Alogrithms.....</b>	<b>15</b>
2.3.1. Bounding Volume Hierarchies .....	16
2.3.2. Kd-Trees .....	18
<b>2.4. Maschinelles Lernen.....</b>	<b>20</b>
<b>2.5. Tensorflow.....</b>	<b>27</b>
<b>3. Datensatz Erstellung und Analyse.....</b>	<b>29</b>
<b>3.1. Datensatz für Szenengeometrie Schnitttests.....</b>	<b>29</b>
3.1.1. Testszene & Geometrie.....	29
3.1.2. Raw Daten .....	31
3.1.3. Datenteilung und Datenverteilung .....	32
<b>3.2. Datensatz für Sichtbarkeitstests.....</b>	<b>37</b>
3.2.1. Testszene & Geometrie.....	38
3.2.2. Raw Daten .....	39
3.2.3. Datenverteilung .....	40
<b>3.3. Feature Engineering .....</b>	<b>44</b>
3.3.1. Datensatz für Szenengeometrie Schnitttests.....	45
3.3.2. Datensatz für Sichtbarkeitstests .....	47
<b>4. Hardware.....</b>	<b>48</b>
<b>5. AI Modelle, Training &amp; Evaluierung .....</b>	<b>50</b>
<b>5.1. Datensatz für Szenengeometrie-Schnitttests.....</b>	<b>51</b>
5.1.1. Linearer Regressor .....	51
5.1.2. Lineares Klassifizierungsmodell .....	54
5.1.3. Neuronal Network .....	60
<b>5.2. Datensatz für Sichtbarkeitstests.....</b>	<b>69</b>
5.2.1. Lineares Klassifizierungsmodell .....	69

5.2.2. Neuronales Netzwerk.....	70
<b>6. Geschwindigkeit, Genauigkeit, Auswertung .....</b>	<b>76</b>
<b>7. Praktische Anwendbarkeit &amp; Ergebnisbewertung .....</b>	<b>79</b>
<b>Literaturverzeichnis .....</b>	<b>84</b>

# Abbildungsverzeichnis

<b>Abbildung 1</b> Render-Ergebnisse 3D-Komposition gerendert mit dem Cinema 4D Renderer.....	2
<b>Abbildung 2</b> Komposition einer 3D Szene im Cinema 4D Viewport.....	2
<b>Abbildung 3</b> Wolken Simulation in Houdini gerendert mit Mantra.....	3
<b>Abbildung 4</b> Rauch Simulation gerendert mit V-Ray Next .....	3
<b>Abbildung 5</b> Arnold GPU (Beta) Raytracer (Echtzeit-Vorschau).....	4
<b>Abbildung 6</b> Renderergebnis Pixar RenderMan .....	4
<b>Abbildung 7</b> Raytracing-Prinzip. Bildquelle: Wikipedia [6] .....	9
<b>Abbildung 8</b> Sichtbarkeit und Schatten beim Raytracing. Bildquelle: Wikipedia [6]	11
<b>Abbildung 9</b> Geometrische Herleitung des Lichtanteils, der von der Lichtquelle zum Punkt P ausgesendet wird. Bildquelle: Physical Based Rendering [1]....	11
<b>Abbildung 10</b> Rekursives Raytracing. Bildquelle: Wikipedia [6].....	12
<b>Abbildung 11</b> Disney Moana Island Szene Quelle: Walt Disney Animation Studios (WDAS) [7].....	15
<b>Abbildung 12</b> Bounding Volume Hierarchie Quelle: Physical Based Rendering [1]17	
<b>Abbildung 13</b> Kd-Tree Sturktur Quelle: Physical Based Rendering [1].....	18
<b>Abbildung 14</b> Tensorflow Framework Hierarchie Bildquelle: tensorflow.org .....	27
<b>Abbildung 15</b> Killeroo-Simple Scene.....	29
<b>Abbildung 16</b> Plot Ursprungskoordinaten der Strahlen; Kompletter Datensatz.....	32
<b>Abbildung 17</b> Plot Richtungsvektoren der Strahlen; Kompletter Datensatz .....	33
<b>Abbildung 18</b> Plot Verteilung positive & negativer Schnitttests der Strahlen; Kompletter Datensatz .....	34
<b>Abbildung 19</b> Oben Plot Ursprungskoordinaten, Unten Plot Richtungsvektoren der Strahlen; Kleiner Datensatz .....	35
<b>Abbildung 20</b> Plot Verteilung positive & negativer Schnitttests der Strahlen; Kleiner Datensatz .....	36
<b>Abbildung 21</b> Raytracer Demo Bild .....	38
<b>Abbildung 22</b> Szenengeometrien für Sichtbarkeitstests .....	38
<b>Abbildung 23</b> Sichtbarkeitstest Punktverteilungen; Oben Punkt A; Unten Punkt B	40
<b>Abbildung 24</b> Sichtbarkeitstest; Links Richtungsvektor (x,y,z) Verteilung, Rechts Punktabstandsverteilung.....	40
<b>Abbildung 25</b> Sichtbarkeitstest Verteilung Sichtbarkeit .....	41
<b>Abbildung 26</b> Sichtbarkeitstest Verteilung Sichtbarkeit; Szene-ID 3 & Zufällige Verteilung auf Kugeloberflächen .....	42
<b>Abbildung 27</b> Sichtbarkeitstest Verteilung Sichtbarkeit; Szene-ID 3 & Zufällige Verteilung auf zueinander zu gewendeten Kugeloberflächen .....	43

<b>Abbildung 28</b> Vektor v im $\mathbb{R}^3$ Bildquelle: Direction cosine, en.Wikipedia.org.....	46
<b>Abbildung 29</b> GPU Auslastung beim Modell-Training .....	49
<b>Abbildung 30</b> Linearer Regressor Konvergenz-Kurve, Links ID 1, Rechts ID 2 .....	52
<b>Abbildung 31</b> Lineares Klassifiziermodell Konvergenz-Kurve; Links ID 3, Rechts ID 6 .....	55
<b>Abbildung 32</b> Lineares Klassifiziermodell ID 8; Links Konvergenz-Kurve, Rechts ROC Kurve .....	57
<b>Abbildung 33</b> Lineares Klassifiziermodell ID 9; Links Konvergenz-Kurve, Rechts ROC Kurve .....	58
<b>Abbildung 34</b> Linear Klassifiziermodell ID 13; Links Konvergenz-Kurve, Rechts ROC Kurve .....	59
<b>Abbildung 35</b> Konvergenzkurven Deep Neural Network Klassifizierungs-Modell ID 5, 7 &10 .....	62
<b>Abbildung 36</b> Deep Neural Network Klassifizier-Modell ID 10 ROC Kurve .....	64
<b>Abbildung 37</b> Deep Neural Network Classifier Modell-ID 11; Links Konvergenz-Kurve, Rechts ROC Kurve .....	65
<b>Abbildung 38</b> Deep Neural Network Klassifizier-Modell ID 12; Links Konvergenz-Kurve, Rechts ROC Kurve .....	66
<b>Abbildung 39</b> Deep Neural Network Klassifizier-Modell ID 14 und 15 Konvergenzkurven & ROC-Kurve .....	67
<b>Abbildung 40</b> Lineares Klassifizierungsmodell ID 02_01; Links Konvergenz-Kurve, Rechts ROC-Kurve .....	70
<b>Abbildung 41</b> Lineares Klassifizierungsmodell ID 02_03; Links Konvergenz-Kurve, Rechts ROC-Kurve .....	71
<b>Abbildung 42</b> Deep Neural Network Klassifizierungs-Modell ID 02_03 - 02_05; Oben Konvergenz-Kurven, Unten ROC-Kurven .....	72
<b>Abbildung 43</b> Deep Neural Network Klassifizierungs-Modell ID 02_06 & 02_07; Oben Konvergenz-Kurven, Unten ROC-Kurven .....	73
<b>Abbildung 44</b> Auswertung Keras Referenzimplementierung; Links 2 Layer, Rechts 4 Layer.....	75

# **Listing-Verzeichnis**

<b>Listing 2.1</b> Tensorflow Erstellen, Trainieren und nutzen eines Neuronalen Netzwerkes .....	28
<b>Listing 5.1</b> Tensorflow Linearer Regressors Pseudocode.....	51
<b>Listing 5.2</b> Tensorflow Lineares Klassifizier-Modell Pseudocode.....	54
<b>Listing 5.3</b> Tensorflow Deep Neural Network Klassifizier-Modell Pseudocode .....	60
<b>Listing 5.4</b> Keras Modell Struktur .....	75

# Tabellenverzeichnis

<b>Tabelle 1</b> Confusion Matrix .....	26
<b>Tabelle 2</b> Übersicht Szenenparameter Testszene Datensatz für Schnitttests .....	30
<b>Tabelle 3</b> Übersicht Parameter-Verteilung im kompletten Datensatz .....	34
<b>Tabelle 4</b> Übersicht Parameterverteilung im kleinen Datensatz .....	37
<b>Tabelle 5</b> Szenen Bounding Box Ausmaße für Sichtbarkeitstests .....	39
<b>Tabelle 6</b> Kugel Verteilung Szene-ID 3 (Szene-ID 1 & 2 sind äquivalent) .....	43
<b>Tabelle 7</b> Verwendete Computer Hardware .....	48
<b>Tabelle 8</b> Modell-Parameter Linearer Regressor Modell ID 1 & 2 .....	52
<b>Tabelle 9</b> Datensatzteilung; Szenengeometrieschnitttests, Absolute Mengen Größen .....	54
<b>Tabelle 10</b> Modell-Parameter Linear Klassifiziermodell ID 3, 4 & 6 .....	55
<b>Tabelle 11</b> Modell-Parameter lineares Klassifiziermodell ID 8.....	57
<b>Tabelle 12</b> Modell-Parameter lineares Klassifiziermodell ID 9.....	58
<b>Tabelle 13</b> Modell-Parameter lineares Klassifiziermodell ID 13.....	59
<b>Tabelle 14</b> Parameter Deep Neural Network Klassifizier-Modell ID 5, 7 &10.....	62
<b>Tabelle 15</b> Auswertung Deep Nerual Network Klassifizierungs-Modell ID 5, 7 &10	63
<b>Tabelle 16</b> Parameter Deep Neural Network Klassifizier-Modell ID 11, 12, 14 und 15	65
<b>Tabelle 17</b> Datensatzteilung; Sichtbarkeitstests, Absolute Mengen Größen .....	69
<b>Tabelle 18</b> Parameter Modell-Training für Modell ID 02_01 & 02_02 .....	69
<b>Tabelle 19</b> Parameter Modell-Training für Modelle ID 02_03 - 02_07 .....	71
<b>Tabelle 20</b> Auswertung Deep Nerual Network Klassifizierungs-Modell für ID 02_03 - 02_05.....	72
<b>Tabelle 21</b> Ausschnitt Schnitttest Datensatz (Exportiert aus der pbrt Killeroo-Szene [1])	77
<b>Tabelle 22</b> Raytracing Modell Confusion Matrix .....	81

# Abkürzungsverzeichnis

Abkürzung	Bedeutung
<b>ML</b>	Maschinelles Lernen (engl. Machine Learning)
<b>KI (engl. AI)</b>	Künstliche Intelligenz (engl. Artificial Intelligence)
<b>PBR</b>	Physically Based Rendering
<b>PBRT</b>	Physical Based Raytracer; verweis auf den in [1] beschriebenen Raytracer
<b>RMSE</b>	Root Mean square error (dt. Wurzel der mittleren quadratische Abweichung)
<b>FPS</b>	Bilder pro Sekunde (engl. Frames per Second)
<b>GI</b>	Globale Beleuchtung (engl. Global Illumination)



# 1.

## Einleitung

Im Rahmen dieser Bachelor-Thesis soll erforscht werden, in wieweit sich maschinelles Lernen in den Raytrace Prozess integrieren lässt.

Der Raytrace-Algorithmus ist eine Methode um fotorealistische computergenerierte Bilder zu generieren, dabei werden Strahlen rekursiv durch den Raum verfolgt und überprüft ob diese Objekte im Raum treffen.

### 1.1. Motivation

In der Produktions-Pipeline bei der Erzeugung von computergenerierten Bildern und Animationen ist der letzte Schritt<sup>1</sup> die Erstellung eines Bildes aus den 3D-Daten bzw. Szenen-Kompositionen.

Insbesondere werden in diesem Schritt alle Elemente der Computergrafik kombiniert wie Modellierte-Meshes, Animationen/Simulationen, Beleuchtung, atmosphärische Effekte etc. Dadurch, dass in diesem Schritt alle Komplexitäten der einzelnen Bereiche kombiniert werden, wird eine große Rechenpower genutzt und generell liegt in diesem Schritt ein Bottle-Neck vor.

---

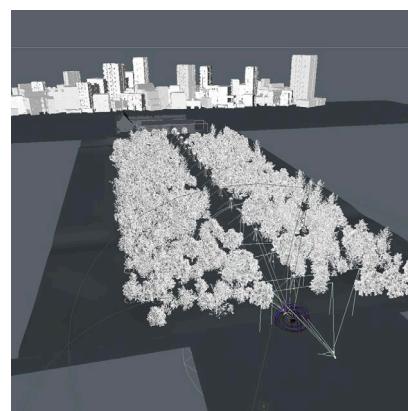
<sup>1</sup> Vor dem finalen Ergebnis wird in der Regel zusätzlich noch Compositing und finale Bildkorrektur durchgeführt. Auf diesen Schritt wird hier nicht weiter eingegangen, da dies nicht in dem Bereich des Raytrace-Algorithmus liegt.



**Abbildung 1** Render-Ergebnisse 3D-Komposition gerendert mit dem Cinema 4D Renderer

Abbildung 1 zeigt ein paar Beispiele computergenerierter Bilder, welche mit dem in Maxon Cinema 4D integrierten Renderer erstellt wurde. Bei dieser Szene handelt es sich um eine 1-minütige Animation einer Kamerafahrt gefolgt von einer Auto-Simulation. Es können mehrere Lichtquellen, Schatten und Reflektionen erkannt werden. Im Hintergrund kann zusätzlich Nebel erkannt werden.

Abbildung 2 gibt einen Überblick über die Szenengeometrie der Szene, es kann eine sehr komplexe und detaillierte Flora Geometrie auf beiden Seiten der Straße erkannt werden.



**Abbildung 2** Komposition einer 3D Szene im Cinema 4D Viewport

Die Szene wurde in den Abbildungen mit einer Auflösung von 5760x1080 Pixeln gerendert und einer Framerate von 25 Bildern pro Sekunde. Die Dauer der Erstellung eines Einzelbildes Betrug zwischen 4 und 10 Minuten. Bei einer Frame-Rate von 25 FPS für die einminütige Animation betrug die gesamte Renderzeit für die 1500 Bilder knapp 140 Stunden.



**Abbildung 3** Wolken Simulation in Houdini  
gerendert mit Mantra

**Abbildung 4** Rauch Simulation gerendert  
mit V-Ray Next

Der Mantra Renderer, integriert in der Houdini Software von Sidefx, hat für die Erstellung der Wolkensimulation in Abbildung 3 ca. 2 Stunde gebraucht. Das exportierte Bild hat eine Auflösung von 1280x720. Bei diesem Beispiel wurden volumetrisches Rendering angewendet, welches den Raytrace-Prozess verkompliziert und sich als einer entsprechend längeren Renderzeit auswirkt.

Abbildung 4 zeigt eine weitere Szene, die mit V-Ray gerendert wurde. Bei diesem Render handelt es sich um einen der meist genutzten Renderer in der Hollywood-Filmindustrie. Für diese HD Szene (1920x1080) benötigte V-Ray Next etwas mehr als eine Minute für den Render-Vorgang, es ist jedoch zu beachten, dass das Ergebnis ohne Multi-Sampling erstellt wurde und ein starkes Bildrauschen auf dem Boden zu sehen ist. Des Weiteren wurde im Vergleich zu der Szene in Abbildung 1 eine weit weniger komplexe Szenengeometrie verwendet. Das Volumen ist jedoch ähnlich der Wolken in Abbildung 3.

Wie in der Einleitung beschrieben, erhöht sich in der Regel die Renderdauer je nach Szenen Geometrie und Qualitätseinstellung des Renderers.



**Abbildung 5** Arnold GPU (Beta) Raytracer  
(Echtzeit-Vorschau)



**Abbildung 6** Renderergebnis Pixar  
RenderMan

Abbildung 5 zeigt eine Szene die mit der Beta Version des Arnold GPU Renderers von Solid Angle erstellt wurde und standardmäßig in Maya integriert ist. Diese Szene kann mit dem interaktiven Renderer mit einer Framerate von ca. 1 FPS bei einer Auflösung von 960x540 gerendert werden. Es ist zu beachten, dass globale Beleuchtungsphänomene dabei berücksichtigt werden. Es ist jedoch das starke Rauschen in der Vorschau zu verzeichnen. Jedoch ist dies ein gutes Werkzeug für eine schnelle Echtzeitvorschau von Beleuchtungseinstellung, Shadereinstellungen und Texturierung.

Abbildung 6 zeigt eine ähnliche Szene wie in Abbildung 5. Diese Szene wurde in 2 Minuten Renderzeit mit dem von Pixar entwickelten Renderer Renderman erstellt. An den Schatten, Reflektionen und Refraktionen kann auch hier der Einsatz von GI erkannt werden.

Sämtliche Szenen wurden selber erstellt und im Rahmen dieser Arbeit gerendert. Insbesondere um die zeitlichen Unterschiede der unterschiedlichen Bildgenerierungen und Softwareimplementierungen aufzuzeigen abhängig, von Szenenkomplexität und GI-Effekten. Das Rendering sämtlicher Szenen wurde mit dem in Abschnitt 4 beschrieben Hardware-System durchgeführt.

Allgemein kann aus diesen Beispielen erkannt werden, wie hoch die Renderdauer bei der Erstellung dieser Bilder sein kann

Zwar haben große Firmen ganze Renderfarmen und bessere Hard- und Software für diesen Prozess, jedoch sind auch generell die Szenen entsprechend komplexer. Um das Licht möglichst realistisch zu simulieren, müssen extrem viele geometrische Tests durchgeführt werden. Einfach gesagt wird dabei die Sichtbarkeit unterschiedlicher Punkte untereinander getestet. Eine detaillierte Beschreibung

befindet sich in Absatz 2.1. Viele dieser geometrischen Tests sind dabei sehr ähnlich und unter Umständen sogar redundant.

## 1.2. Zielsetzung

Maschinelles Lernen findet heutzutage Einzug in viele Bereich [2], z.B. Diagnoseverfahren, Bild- und Videoanalysen sowie Sprach- & Texterkennung um einige Beispiele zu nennen. Insbesondere für die automatische Verarbeitung und Analyse von großen Datenmengen wird maschinelles Lernen als „Schlüsseltechnologie“ [2] gesehen.

Im Gegensatz zu bisherigen Ansätzen der Integration von maschinellem Lernen in den Raytrace Prozess, soll in dieser Arbeit nicht die Anwendung des maschinellen Lernens auf ein bereits bestehendes Ausgabe-Bild untersucht werden [3], sondern dies direkt in dem Geometrie-Teil (Schnitt-Test) des Raytracing-Prozesses Anwendung finden.

Insbesondere soll untersucht werden, ob sich ein Modell, welches auf maschinellem Lernen beruht, auf eine Szene trainieren lässt, um eine Aussage darüber zu treffen, ob ein beliebiger Strahl die Szenen Geometrie trifft bzw. ob die Sichtbarkeit zwischen Punkten gegeben ist. Dadurch sollen zum Beispiel sog. Schattenstrahlen schneller bestimmt werden können. Es sollen in dieser Arbeit verschiedene Methoden zur Generierung solcher Modelle untersucht werden, um eine Aussage treffen zu können, ob sich diese Modelle für den Prozess eignen und wie sich die Qualität unter diesen unterscheidet.

Als grundlegende Struktur und Referenz für den Prozess der Raytracing-Implementation soll der im Buch „Physical Based Rendering“ (Phar et. All) [1] beschriebene Raytracer dienen. In diesem wird eine generelle Struktur für den Raytracing Prozess beschrieben. Es werden sehr viele Thematiken moderner Raytracer diskutiert und es werden Implementations-Referenzen geliefert. Die Struktur des beschriebenen Raytracers wurde auch als Basis für den von Pixar entwickelten Renderer RenderMan genutzt, als dieser „kürzlich als physical based raytracer neugeschrieben wurde“<sup>2</sup>. Dieser zählt zu einem der in der Industrie meist verwendeten Renderern gehört.

---

<sup>2</sup> Siehe [http://www.pbr-book.org/3ed-2018/Introduction/A\\_Brief\\_History\\_of\\_Physically\\_Based\\_Rendering.html](http://www.pbr-book.org/3ed-2018/Introduction/A_Brief_History_of_Physically_Based_Rendering.html)

Im Rahmen dieser Thesis wird die Hypothese aufgestellt, dass ein Modell aus dem Bereich des maschinellen Lernens auf eine gegebene Geometrie trainiert werden kann, um die geometrische Berechnungen zu beschleunigen.

Ein Beispiel für aktuelle Forschungen bezüglich der Integration von maschinellem Lernen in den Render-Prozess ist z.B. die Filterung von starkem Rausch in partiellen Render-Ergebnissen mittels eines erlernten Netzwerk, das die fehlenden Pixel berechnet bzw. durch Filterung korrigiert. [3]

Zum Beispiel kann mit diesem Ansatz ein Bild wie das der Abbildung 5 verbessert werden.<sup>3</sup>

In dieser Arbeit soll konkret ein eher geometrischer Ansatz untersucht werden, ob ein Modell die Schnitttests mit der Szenen-Geometrie bzw. die Sichtbarkeit von Punkten im Raum in einer überschaubaren Zeit erlernen kann, um im Folgenden Strahlen zu Klassifizieren.

Das Ziel dieser Arbeit ist es, anhand verschiedener trainierter Modelle zu untersuchen, ob ein Modell gefunden werden kann, das ein gutes Ergebnis für die Schnitttests liefert. Mit Hilfe dieses soll die Einsatzfähigkeit in dem Raytrace Prozess evaluiert werden anhand der gelieferten Genauigkeit.

Dabei sollen zeitliche Komponenten wie das Erzeugen von Trainingsdaten und das Trainieren des Netzwerkes einbezogen werden sowie die Genauigkeitsabweichung die diese im Vergleich zu den bisherigen geometrischen Verfahren liefert.

Im Rahmen dieser Überlegung wird vor Beginn der Arbeiten folgende Thesis aufgestellt, die es zu untersuchen gilt:

Ein Modell aus dem Bereich des maschinellen Lernens kann eine Szenen-Geometrie Approximation anhand von Strahlen erlernen und mithilfe dieses den Raytrace-Algorithmus an verschiedenen Stellen beschleunigen bei gleichbleibender Szenengeometrie.

---

<sup>3</sup> Dieser als Denoising [3] bezeichnete Ansatz von Chakravarty et al. ist inzwischen in fast allen namhaften Render Engines integrierter. Darunter zum Beispiel Arnold, Vray und Mantra. Insbesondere beim Path Tracing, wie der Monte Carlo Integration bietet die ein großen Geschwindigkeitsvorteil, da die Anzahl der Berechnung bei dem eigentlichen Tracing Prozess minimiert werden kann. Eines der bekanntesten und meist genutzten Frameworks (basierend auf [3]) ist das von NVIDIA entwickelte Optix mit dem AI-Accelerated Denoiser. Siehe <https://developer.nvidia.com/optix-denoiser>

### **1.3. Struktur**

Diese Arbeit ist in 7 Kapitel unterteilt. Das erste Kapitel dient dazu einen Überblick über das Thema aufzuzeigen und die Problemstellung sowie die Motivation für diese zu erläutern.

Das 2. Kapitel soll eine spezifischere Einführung in die Thematik bieten. Insbesondere wird in diesem der Raytracing-Algorithmus erklärt und es wird auf den Teil des Prozess eingegangen, der durch diese Forschungsarbeit betroffen ist.

Des Weiteren werden in diesem Kapitel verwendete Begriffe im Kontext des maschinellen Lernens definiert und erklärt.

In Abschnitt 2.5 wird das Framework (Tensorflow) erläutert, welches verwendet worden ist, um verschiedene Varianten von Modellen (des maschinellen Lernens) zu erstellen, zu trainieren und auszuwerten. Dies wurde im Rahmen eines schnellen Prototypings und dementsprechend schnelleren Ergebnissen in der Programmiersprache Python durchgeführt.

In Kapitel 3 werden die verwendeten Datensätze beschrieben, die exportiert wurde. In den jeweiligen Unterabschnitten wird zuerst auf die generelle Geometrie der jeweiligen Szene eingegangen und welche Besonderheiten diese aufweist. Es wird beschrieben, wie die exportierten Daten aussehen, wie diese unterteilt sind und wie sie in Inputdaten für die weiteren Schritte konvertiert wurden. Es wird analysiert welche Verteilung innerhalb dieser Daten vorliegt.

Kapitel 4 geht auf die verwendete Hardware ein, die bei der Forschung verwendet wurde und wie sich die Leistung und Auslastung über die Testreihen verhält.

In Kapitel 5 werden die Strukturen der Modelle und Parameter der einzelnen Testdurchläufe vorgestellt. Für jeden Durchlauf wird der Lernfortschritt über die Trainingsdauer gezeigt und das Training sowie das finale Resultat jedes Modells abhängig der veränderten Trainingsparameter analysiert. In den jeweiligen Unterabschnitten sind die Modelle ja nach Modell-Kategorie geordnet beschrieben und anhand der Auswirkung von Parameter Änderungen verglichen.

Kapitel 6 beschreibt das Fazit, welches aus den Testreihen des vorherigen Kapitel gezogen wurde. Insbesondere wird auf die Genauigkeit und die damit verbundene Dauer des Trainings eingegangen.

Kapitel 7 bewertet die praktische Anwendbarkeit der Ergebnisse in bestehenden Raytracern und beschreibt einige Ideen und Hypothesen, die aus der Forschungsarbeit entstanden sind.

# 2.

## Grundlagen

Dieser Abschnitt führt in die Thematik der synthetischen Bilderstellung mithilfe von Raytracing ein und gibt einen Überblick über das Thema des maschinellen Lernens. Es werden Begrifflichkeiten definiert, die in den folgenden Hauptteilen der Arbeit verwendet werden.

### 2.1. Raytracing

Raytracing ist der grundlegende Algorithmus, der in fast allen professionellen Render-Softwares integriert ist, um 3D-Szenen zu rendern die in Softwares wie Houdini, Maya, Cinema 4D oder 3ds Max erstellt wurden um dadurch ein Pixelbild zu erstellen. Zu den bekanntesten und meist verwendeten professionellen Render-Softwares gehören derzeit Arnold, V-Ray, Render-Man, Octane sowie Redshift [4]. Abbildung 1 bis Abbildung 6 in Abschnitt 1.1 zeigen einige Render-Ergebnisse dieser Softwares, die diverse Techniken verwenden, jedoch letztendlich auf den Raytrace-Algorithmus zurückzuführen sind.

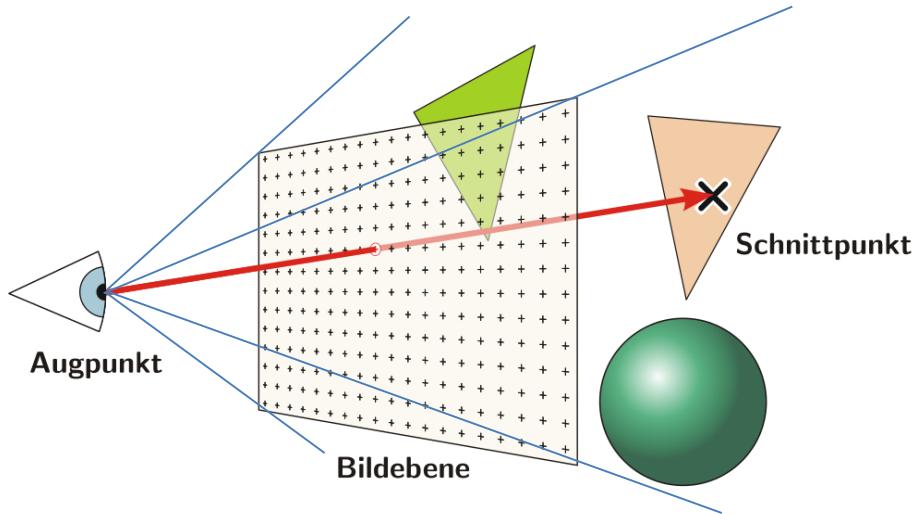
Insbesondere wird die Reflexion, Refraktion und Absorption von Lichtstrahlen simuliert um ein möglichst realistisches Bild zu generieren.

Je nach gewünschter Genauigkeit (mit anderen Worten, je physikalisch korrekter die Annäherung sein soll) kann sich die Berechnungsdauer gravierend erhöhen, sodass an dieser Stelle ein Bottle-Neck der synthetischen Bildgenerierung vorliegt.

Dieser Abschnitt soll einen Überblick über den Raytrace-Algorithmus und die damit verbundene synthetische Bilderstellung liefern. Für eine tiefere Beschreibung des Algorithmus, der Implementation und Anwendung ist das Buch „Physical Based Rendering: From Theory to implementation“ von Matt Pharr und Wenzel Jakob [1] sehr zu empfehlen und diente mit dem Buch „An Introduction to Raytracing“ von Andrew S. Glassner et Al. [5] als Hauptquelle für diesen Abschnitt.

Der Raytrace-Algorithmus ist generell die Verfolgung von Strahlen in der Szene. Dabei wird der Strahl z.T. von Objekten reflektiert oder beeinflusst diese bzw. wird von diesen beeinflusst.

In vielen Systemen wird von einem Augpunkt ausgegangen, von dem aus die Szene betrachtet wird. Dieser wird oft als Kameraobjekt angesehen.



**Abbildung 7** Raytracing-Prinzip. Bildquelle: Wikipedia [6]

Wie Abbildung 7 zeigt, kann das resultierende Bild als eine Bildebene betrachtet werden, die sich vor dem Augpunkt befindet und die Informationen aus der dahinterliegenden Szene beinhaltet, die das Sichtvolumen einschließt. Das Sichtvolumen ist durch die blauen Linien definiert, die vom Augpunkt durch die 4 Ecken der Bildebene gehen. Von der Kamera (Augpunkt) werden Sichtstrahlen in die Szene verfolgt.

Ein Strahl ist geometrisch definiert als:

$$r(\lambda) = O + \lambda \vec{D}$$

**Formel 2.1** Geometrische Definition Strahl/Halbgerade; Parametrische Form

In der Formel 2.1 beschreibt  $O$  den Ursprungspunkt des Strahls und  $\vec{D}$  den Richtungsvektor. Der skalare Parameter  $\lambda$  besitzt einen Wertebereich von  $[0, \infty]$  und beschreibt die Skalierung des Richtungsvektors. (siehe Kapitel 2.5 in [1])

Im einfachsten Fall wird beim Raytracing der Augpunkt als Ursprungspunkt genutzt und der Richtungsvektor ist der Vektor, der vom Augpunkt durch den jeweiligen Bildpunkt auf der Bildebene geht. Um ein Bild zu erstellen, muss der Algorithmus über das gesamte Bild iterieren. In Abbildung 7 ist einer dieser Strahlen in Rot dargestellt.

An dieser Stelle beginnt der wichtigste Teil des Algorithmus, denn es muss festgestellt werden, welche Objekte der Strahl in der Szene schneidet. Wenn ein Objekt der Szene durch eine implizite Funktion beschrieben werden kann, kann dieser Schnitttest per Substitution der Strahlgleichung gelöst werden.

$$(O_x + \lambda \vec{D}_x)^2 + (O_y + \lambda \vec{D}_y)^2 + (O_z + \lambda \vec{D}_z)^2 - r^2 = 0$$

**Formel 2.2** Substitution der Strahlgleichung mithilfe der impliziten Kugelgleichung

Diese quadratische Gleichung wird dann für den Parameter Lambda gelöst, z.B. mithilfe eines Gleichungssystems. Sollte das Gleichungssystem nicht mit reellen Zahlen lösbar sein, so schneidet der Strahl die Kugel nicht. Typischerweise wird der Schnittpunkt mit dem geringsten Abstand zur Kamera betrachtet, der mithilfe des Parameters Lambda ermittelt werden kann.

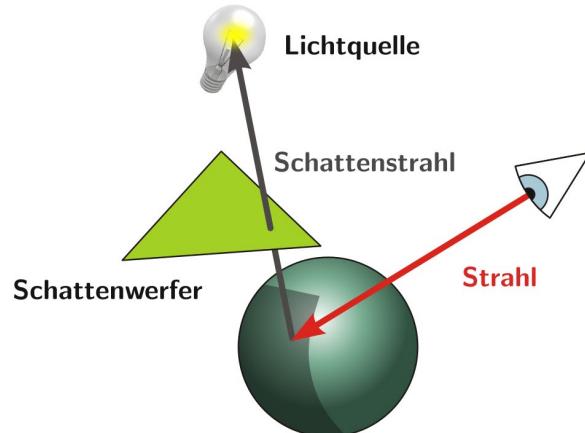
Für jeden Schnittpunkt müssen dabei noch weitere Parameter wie z.B. die Oberflächennormale und das Material des Objektes bestimmt werden, die für die folgenden Lichtberechnungen benötigt werden.

Wie durch die Objekte genau iteriert wird, um den Schnittpunkt mit dem kleinsten Abstand zu finden, wird genauer in Abschnitt 2.3 erklärt, da dies der Abschnitt ist, der durch diese Forschung auf mögliche Beschleunigung untersucht werden soll.

Damit Objekte in der Szene sichtbar sind, muss ein Raytracer Lichtquellen simulieren und dessen Einfluss auf die Objekte berechnen. Dies ist deshalb entscheidend, da die Menge an Licht bestimmt werden soll, die von dem berechneten Punkt in Richtung Augpunkt (Abbildung 9;  $\omega_o$ ) geht und entsprechend des verwendeten Farbmodells in Farben umgerechnet werden kann. (z.B. RGB-Farbe)

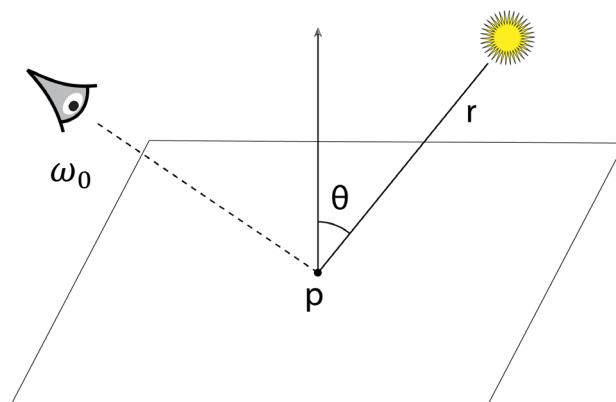
Da es sich bei Licht um eine Form von elektromagnetischer Strahlung handelt und einen Energie-Transport darstellt, können Beobachtungen und Gesetze aus der Radiometrie in diesem Schritt Anwendung finden. Um zu bestimmen, ob und wieviel Einfluss eine Lichtquelle auf den Punkt hat, muss die Sichtbarkeit untereinander geprüft werden. Für diese Prüfung kann wieder ein Strahl (Schattenstrahl) zu den

Lichtquellen verfolgt werden (Abbildung 8) und es kann auf Verdeckung von Objekten zwischen der Lichtquelle und dem Objekt geprüft werden<sup>4</sup>.



**Abbildung 8** Sichtbarkeit und Schatten beim Raytracing. Bildquelle: Wikipedia [6]

Als weiteren Punkt muss beachtet werden, dass die Energie über die Länge eines Strahls abnimmt, da es sich in den meisten Szenen nicht um ein Vakuum handelt, in dem sich der Strahl befindet. Dadurch können atmosphärische Effekte wie Nebel und Rauch generiert werden. Die Energie, die von einem Licht an einem Punkt  $P$  ankommt, nimmt mit der quadratischen Distanz ( $r$  in Abbildung 9) ab.



**Abbildung 9** Geometrische Herleitung des Lichtanteils, der von der Lichtquelle zum Punkt  $P$  ausgesendet wird. Bildquelle: Physical Based Rendering [1]

---

<sup>4</sup> In der realen Welt gibt es keine Punktlichtquellen, sondern lediglich geometrische Objekte die Licht emittieren. Der einfacheitshalber wird hier an dieser Stelle eine Approximation durch eine Punktlichtquelle angenommen, welche Licht gleichmäßig in alle Richtungen ausstrahlt.

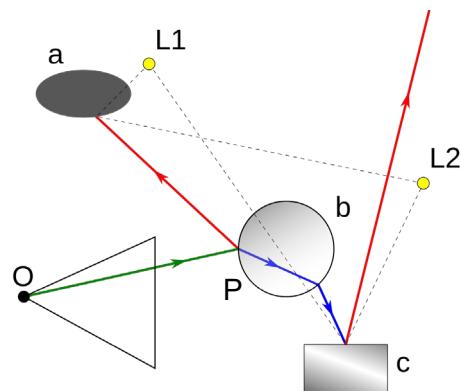
Des Weiteren kann gezeigt werden, dass das Licht, welches den Punkt  $P$  (bzw. den kleinen Oberflächenausschnitt um  $P$ ) erreicht, proportional zu  $\cos(\theta)$  ist.  $\theta$  beschreibt den Winkel zwischen der Oberflächennormalen an dem Punkt  $P$  und dem Vektor der vom Punkt  $P$  zu der Lichtquelle zeigt.

Wenn eine Sichtbarkeit zwischen dem Objekt und der jeweiligen Lichtquelle besteht, kann mithilfe der Oberflächeneigenschaften (z.B. Material, Medium, Emissions Charakteristik) bestimmt werden, wie sich das Licht auf dieses auswirkt. Dadurch kann bestimmt werden, welcher Lichtanteil in Richtung der Kamera geht (Abbildung 9;  $\omega_0$ ).

Zur Berechnung dieses Anteils der ausgehenden Energie von einer eingehenden Richtung (Lichtquelle zu Punkt  $P$ ) kann die Bidirectional Reflectance Distribution Function (BRDF) verwendet werden. Bei der BRDF handelt es sich um eine 4-dimensionale Funktion. Eine allgemeinere Form stellt die Bidirectional Scattering Distribution Function (BSDF)<sup>5</sup> dar.<sup>6</sup>

Bei mehreren Lichtquellen kann der jeweilige Anteil aufaddiert werden, um den Gesamtanteil an Licht für den betrachteten Ausschnitt (Punkt  $P$ ) zu bestimmen.

Neben dem beschriebenen direkten Lichttransport bietet ein Raytracer den Vorteil die globale Beleuchtung zu beachten. Daher auch den indirekten Lichttransport, der durch indirekte spiegelnde Reflexionen und Transmissionen stattfindet.



**Abbildung 10** Rekursives Raytracing. Bildquelle: Wikipedia [6]

<sup>5</sup> Von der BSDF gibt es eine Reihe von Varianten, z.B. wird für die Berechnung der Lichtstreuung bei Volumen die Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF) verwendet.

<sup>6</sup> Die genaue Definition der BRDF oder BSDF ist, für den in dieser Arbeit behandelten, geometrischen Abschnitt nicht direkt relevant und es wurde auf eine genauere Beschreibung verzichtet. Diese kann in Kapitel 5.4ff, 8 und 9 [1] gefunden werden.

Abbildung 10 zeigt den rekursiven Strahlenverlauf. Objekt *a* stellt ein opakes, nicht reflektierendes Objekt dar.

Objekt *b* stellt dabei ein ideal semi-transparentes und spiegelndes Objekt in der Szene dar, welches einen Brechungsfaktor ungleich 1 besitzt.

Objekt *c* stellt ein spiegelndes Objekt dar. *L*<sub>1</sub> und *L*<sub>2</sub> sind jeweils Punktlichtquellen.

Ein Strahl, der von Augpunkt *O* verfolgt wird, trifft in diesem Beispiel auf das Objekt *b*. Von dem Schnittpunkt aus wird zum einen ein Reflexions- und zum anderen ein Transmissions-Strahl<sup>7</sup> weiterverfolgt. Da der gesamte Lichtanteil aus der Reflexion und der Transmission bestimmt wird, wird die direkte Beleuchtung an dieser Stelle nicht bestimmt werden.

Der reflektierte Strahl ist in rot dargestellt, der transmittierte Strahl in blau. Jeder dieser Strahlen wird im folgenden Schritt rekursiv weiterverfolgt. Für Objekt *a* und *c* wird zusätzlich der direkte Lichtanteil für jede Lichtquelle bestimmt (die gestrichelten Linien).

Der an dem initialen Schnittpunkt vorhandene Lichtanteil ergibt sich somit aus den indirekten Anteilen der weiteren Szenen Geometrien. An diesem Beispiel wird für die Bestimmung des Lichtanteils an der Position *P* der initiale Sichtstrahl (grün), 2 reflektierte Strahlen (rot), 2 transmittierte Strahlen (blau) sowie 4 Schattenstrahlen (grau gestrichelt) benötigt.

Abbildung 5 und Abbildung 6 zeigen jeweils Beispiel von spiegelnden und semi-transparenten Objekten.

Diese Beobachtung führt zu der Rendering Equation in Formel 2.3:

$$L_0(P, \omega_0) = L_e(P, \omega_0) + \int_{S^2} f(P, \omega_0, \omega_i) L_i(P, \omega_i) |\cos\theta_i| d\omega_i$$

### Formel 2.3 Rendering Equation

Der ausgehende Lichtanteil von Punkt *P*  $L_0(P, \omega_0)$  in die Richtung  $\omega_0$  ergibt sich aus der Summe des emittierenden Lichtes der Oberfläche  $L_e(P, \omega_0)$  und der einfallenden Lichtanteile  $L_i(P, \omega_i)$  aus allen Richtungen der Halbkugel  $S^2$  um den Punkt *P* skaliert durch die BSDF  $f(P, \omega_0, \omega_i)$  und den Kosinus Term.

---

<sup>7</sup> Um den reflektierten bzw. transmittierten Strahl zu bestimmen, wird das Snelliussche Brechungsgesetz verwendet, mithilfe der Reflexions-/Brechungindizes und der Oberflächennormalen an Punkt *P*.

An Formel 2.3 kann erkannt werden, dass aufgrund des Integrals über die Halbkugel das Problem in der Regel nicht analytisch gelöst werden kann, sodass dies durch verschiedene Ansätze approximiert werden muss<sup>8</sup>.

Trotz dieser Approximationen verbleibt in der Regel eine große Anzahl an Schnitttests, die im Rahmen des Raytracing Algorithmus durchgeführt werden.

Generell kann erkannt werden, dass viele der Strahlen sehr ähnlich sind bzw. es sogar in einigen Fällen identische Strahlen geben könnte.

Aus diesem Grund ist es elementar, dass die Implementation der Schnitttests, mit der Geometrie effizient ist. Neben verschiedenen Datenstrukturen zur Beschleunigung (siehe Abschnitt 2.3), wird der Render-Prozess durch Ausnutzung der Hardware wie besondere CPU-Befehlsinstruktionen, Multi-Threading oder GPU-Berechnungen beschleunigt.

Wie in der Einleitung beschrieben, soll diese Arbeit einen Ansatz untersuchen, um genau diesen Schritt des Algorithmus zu beschleunigen.

Dies soll durch das Training eines Modells mit vorhandenen Strahlen und Schnitttest Auswertungen geschehen. Im Rahmen dieser Arbeit soll das Modell lediglich unterscheiden, ob ein gegebener Strahl die Szenengeometrie trifft oder nicht bzw. in einem zweiten Ansatz, ob zwischen zwei Punkten in der Szene eine Sichtbarkeit besteht. Auf der praktischen Ebene soll dieser Ansatz helfen, z.B. Schattenstrahlen zu klassifizieren.

Eine detailliertere (spezifischere) Ausgabe des Modells, wie Geometrie (-ID) oder Schnittpunkt-Parameter wären auch denkbar, aber für diese wird angenommen, dass es zu komplex für ein Modell werden würde und sich somit nicht eignen würde.

## 2.2. PBRT Rendering System

Physical based Rendering ist eine Technik, bei der Algorithmen verwendet werden, die versuchen, das Render-Ergebnis möglichst nahe zu der realen physikalischen Welt zu erzeugen. Insbesondere ist damit eine sehr komplexe Lichtberechnung verbunden die sich sehr stark an der realen Radiometrie orientiert.

Das pbrt System ist ein kompletter Raytracer dessen Implementierung in dem Buch „Physical Based Rendering: From Theory to Implementation“ von Matt Pharr und Wenzel Jakob [1] beschrieben ist. Das System implementiert alle modernen Render

---

<sup>8</sup> Eine tiefere Analyse dieser Methoden würde den Rahmen dieses Überblicks sprengen und wurde deshalb hier vernachlässigt.

Techniken die auch in den meisten kommerziellen Raytracer zu finden sind.

Es handelt sich dabei um einen objekt-orientierten Ansatz mit C++ als Implementationssprache. Das System ist darauf ausgelegt eine einfache Schnittstelle für neue Implementationen zu bieten und sich dadurch einfach erweitern lässt.

In den Basis Klassen können die in Absatz 2.1 beschriebenen Teile eines Standard Raytracer gefunden werden. (z.B. Camera, Shape, Material etc.)

Im Rahmen dieser Arbeit wurde lediglich eine kleine Komponente beim pbrt ergänzt: ein Modul zum Export von Schnitttests aus einer Szene in eine CSV-Datei. Insbesondere musste dafür beachtet werden, dass das pbrt System hoch parallel arbeitet (bzw. arbeiten kann) und dies beim Schreiben in die CSV-Datei als atomarer Vorgang beachtet werden muss.

### 2.3. Acceleration Alogrithms

In einem naiven Ansatz, wird für jeden Strahl der getestet werden muss die gesamte Szenengeometrie durchgegangen werden. Wenn jedoch Szenen vorliegen wie aktuelle Produktionsstandards von Firmen wie Disney kann schnell erkannt werden, dass dieser Ansatz wenig effizient ist. Abbildung 11 zeigt ein Beispiel aus dem 2016 erschienenen Disney Film Vaiana (engl. Moana).



**Abbildung 11** Disney Moana Island Szene  
Quelle: Walt Disney Animation Studios (WDAS) [7]

Die Szene repräsentiert laut Autoren viele Herausforderungen bei aktuellen Produktionen. Die Szene beinhaltet viele Instanz Objekte (Bäume, Blätter etc.) in einer sehr großen Szene und komplexen Lichttransport (z.B. beim Wasser und den

Wolken). Eine genauere Beschreibung der Szene kann in dem 2018 vom Walt Disney Animation Studio veröffentlichten Paper [7] zu der Szene gefunden werden.

Durch die in Abschnitt 2.1 beschriebene Rekursion, die beim Raytracing stattfindet, kann es in solchen Szenen extrem zeitaufwändig werden, wenn für jeden Strahl die komplette Szenen Geometrie getestet werden muss. Es ist festzustellen, dass jeder Strahl in der Regel nur wenige Objekte in der Szene schneidet und die anderen weit verfehlt. Um Szenen in solchen Dimensionen rendern zu können, wird eine Verbesserung dieser Schnitttests beim Raytracing benötigt.

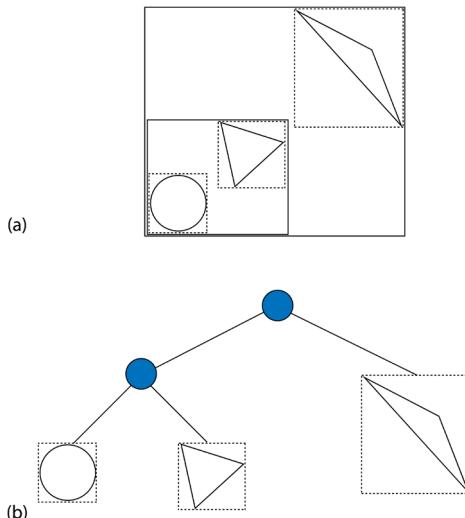
Sogenannte Beschleunigungsstrukturen (engl. Acceleration structures) sollen den Raytrace Prozess beschleunigen, indem eine effizientere Traversierung der Szene bei den Schnitttests gewählt wird. Anstelle jeweils alle Objekte der Szene testen zu müssen, kann mithilfe dieser Beschleunigungsstrukturen, die Anzahl der Schnitttests auf eine logarithmische Komplexität reduziert werden. Diese Komponente ist eine der Komponenten im „Herzen des Raytracers“ (Kapitel 4.2 [1]) und ist essentiell entscheidend für die finale Laufzeit dieses. Dabei ist es das Ziel, schnell und gleichzeitig Gruppen von Primitiven auszuschließen.

Zwei der meist genutzten Beschleunigungsstrukturen sind die Bounding Volume Hierarchie und Kd-Trees. Beide Ansätze erstellen eine neue Szenenhierarchie für die Traversierung, die jeweils darauf ausgelegt ist, die „Kosten“ (Berechnungsaufwand und -zeit) der gesamten Schnitttests zu minimieren.

Beide Strukturen und deren Implementierung sind im Detail in Kapitel 4.3 und 4.4 in dem Buch „Physical Based Rendering: From Theory to Implementation“ [1] beschrieben. In den folgenden beiden Unterabschnitten werden diese kurz vorgestellt.

### 2.3.1. Bounding Volume Hierarchies

Bounding Volume Hierarchies (BVHs) sind dem Bereich der Objekt Unterteilungsalgorithmen zuzuordnen. Dabei werden Objekte absteigend in kleinere disjunkte Teilmengen aufgeteilt.



**Abbildung 12** Bounding Volume Hierarchie Quelle: Physical Based Rendering [1]

Abbildung 12 (b) zeigt ein Beispiel einer Bounding Volume Hierarchie. Von der Wurzel geht ein Blatt und ein weiterer Knoten ab, von dem mittleren Knoten gehen weitere zwei Blätter ab.

Dabei werden die Primitive (Geometrien wie z.B. Meshes) der Szenen in den Blättern gespeichert und jeder Knoten speichert eine Bounding Box<sup>9</sup> (gestrichelte Boxen Abbildung 12 (a)) seiner Kind-Knoten. Dadurch kann bei einer Traversierung ein Teilbaum übersprungen werden, wenn die Bounding Box dieses nicht vom aktuell untersuchten Strahl geschnitten wird. Die Bounding Box des Wurzelknotens enthält demnach die komplette Szene.

Zusammengefasst besteht der Prozess bei der Konstruktion aus 3 Schritten. Zuerst wird die Hüllengeometrie für jedes Objekt der Szene bestimmt. Mithilfe dieser Information kann ein binärer Baum mit Hilfe verschiedener Teilungsalgorithmen konstruiert werden, der Knoten mit den Bounding Boxen und Blätter mit den Geometrien enthält. Die Verbindung basiert auf Pointern. In einem Folge Schritt wird diese Struktur in eine effizientere Struktur (depth-first array layout [1]) umgewandelt, die sich besser für das interne Speicherlayout des Computers eignet.

---

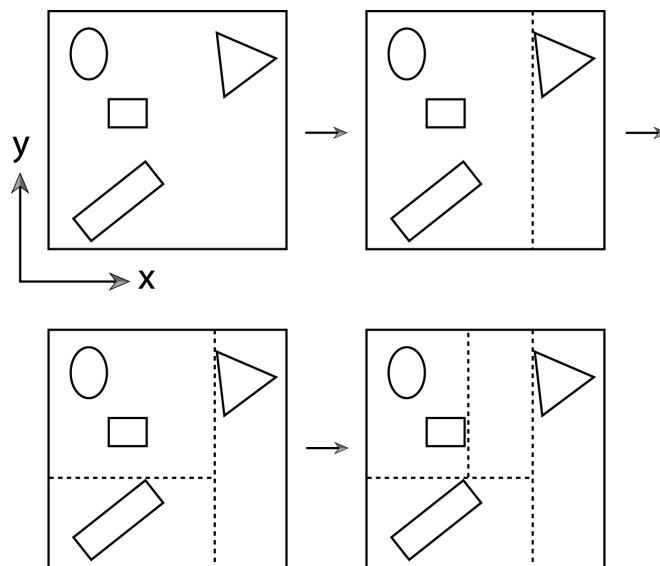
<sup>9</sup> Ein Hüllkörper (engl. Bounding Volume) ist ein einfacher geometrischer Körper, der ein oder (in diesem Fall) mehrere komplexere Objekte umschließt. Bounding Boxen sind eine Teilmenge der Bounding Volume. Eine gute Übersicht über diesen Bereich bietet das Buch „Computational Geometry – Algorithms and Applications“ [13], insbesondere die konvexe Hülle in Kapitel 11 ist oft im Rahmen der Computer Grafik anzutreffen.

Bei einer Traversierung wird beim Wurzelknoten begonnen und dann werden die Kindknoten untersucht. Um die Traversierung effizienter zu machen, kann dieser abhängig von der Richtung des Strahls den Kindknoten wählen, der je nach angewendetem Teilungsalgorithmus eher getroffen wird, basierend auf den geometrischen Informationen.

Am Ende des folgenden Abschnittes werden die Vor- und Nachteil dieser Struktur im Vergleich zu der Kd-Tree Struktur diskutiert.

### 2.3.2. Kd-Trees

Kd-Trees sind dem Bereich der räumlichen Unterteilungsalgorithmen (vgl. Binary Space Partitioning (BSP) Trees) zuzuordnen, die den 3D Raum in disjunkte Regionen unterteilen und dabei speichert, welchen Regionen Primitive schneiden. Bei der eigentlichen Traversierung müssen nur die Primitive getestet werden, die die Regionen schneiden. Die Aufteilung des Raumes bei der Konstruktion wird mithilfe von Schnitt-Ebenen durchgeführt.



**Abbildung 13** Kd-Tree Sturktur Quelle: Physical Based Rendering [1]

Auch dieser Algorithmus beginnt mit einer Bounding Box, die die gesamte Szene umschließt (Abbildung 13; oben links).

Es wird ein Schwellwert festgelegt, der angibt wie viele Primitive sich maximal in einem Unterschabschnitt befinden dürfen. Überschreitet die aktuelle Anzahl an Primitiven in einer Region diesen Schwellwert, wird die Region (3D Raum) mittels einer Ebene (2D) geschnitten und die Primitive werden auf die jeweiligen

Unterregionen je nach Zugehörigkeit aufgeteilt (Abbildung 13; oben rechts & unten links). Es kann dabei vorkommen, dass die Teilungsebene so gewählt wird, dass Geometrien in beiden Teilregionen liegen; in diesem Fall werden diese mit beiden Teilregionen assoziiert und können demnach mehrfach in der Struktur vorkommen. Dieser Prozess wird rekursiv fortgeführt bis entweder jedes Blatt die entsprechend definierte maximal Anzahl an Primitiven beinhaltet oder eine maximale Rekursionstiefe erreicht ist. Die Schnittebene ist immer rechtwinklig zu einer der drei Koordinaten Achsen<sup>10</sup>.

Beim Traversieren, wird jede Region getestet. Bei positivem Schnitttest mit den Knoten wird ermittelt, welcher Kindknoten zuerst traversiert werden soll und dieser Prozess nach der Reihe durchgeführt. Auch dieser Algorithmus kann so konstruiert werden, dass beim Traversieren, die Kindknoten zuerst getestet werden, die geometrisch wahrscheinlicher getroffen werden (depth-first, front-back Traversierung).

Eine wichtige positive Eigenschaft der BVHs gegenüber der Kd-Trees ist, dass jedes Primitiv lediglich einmal in der Hierarchie vorkommt.

Ein Nachteil dieser Struktur ist, dass ein Primitiv mehrere räumliche Unterteilungen schneidet und daher unter Umständen mehrfach getestet werden müsste. Ein weiterer Nachteil bei den BVHs ist die benötigte Speichergröße der Struktur, insbesondere bei Szenen wie der, die in der Einleitung von Abschnitt 2.3 vorgestellt wurde, kann dieses eine bestimmte Relevanz haben, die beachtet werden muss. Die Kd-Tree Struktur liefert in der Regel schneller Schnitt-Test-Ergebnisse zurück ist aber komplizierter und weniger effizient zu konstruieren als die BVHs und benötigt daher eine längere Konstruktionsdauer. Des Weiteren sind Kd-Trees stärker anfällig bezüglich numerischer Robustheit aufgrund von Rundungsfehlern.

---

<sup>10</sup> Eine weitere bekannte Variation des BSP Baums ist der Octree. Hierbei wird die aktuelle Box durch 3 Ebenen, die zu den Achsen rechtwinklig sind, geteilt und in 8 Unterregionen bei jedem Schritt zerlegt.

## 2.4. Maschinelles Lernen

Dieser Abschnitt beschreibt, was maschinelles Lernen ist und definiert bestimmte Begriffe, die im Laufe der weiteren Arbeit wiederverwendet werden. Es ist zu beachten, dass zum Großteil auf die englischen Begriffe zurückgegriffen wurde, da diese sich in der Praxis durchgesetzt haben.

Das Prinzip des maschinellen Lernens basiert auf der Generierung von Wissen aus Erfahrung. Ein solches System beruht auf der Entwicklung von Lernalgorithmen aus Beispielen, woraus ein komplexes Modell entwickelt werden kann.

Mit Hilfe dieses erlernten Wissens kann das System auf neue und potentiell unbekannte Daten angewendet werden. Bei Problemen die zu komplex sind, um sie analytisch zu beschreiben, es jedoch viele Beispieldaten gibt, bietet sich maschinelles Lernen als Mittel an.

Einen guten Überblick über das Thema bietet das Buch „Deep Learning“ von Goodfellow et al. [8] sowie eine praxisorientierte Analyse kann in der 2018 vom Fraunhofer-Institut veröffentlichten Studie „Maschinelles Lernen Eine Analyse zu Kompetenzen, Forschung und Anwendung“ [2] gefunden werden .

### Labels & Features

Bei Features handelt es sich um die Inputparameter. Diese werden beim maschinellen Lernen als Eingangsparameter genutzt, um damit zu arbeiten. Als Label wird die Ausgabe (auch mehrere Ausgaben sind möglich) bezeichnet die geliefert wird, ausgehend von den (Input)-Features.

Dies kann mit einer Funktion  $f(x_i, x_{ii}) = y$  verglichen werden, bei der  $x_i$  und  $x_{ii}$  die Features sind und  $y$  das Label. Features und Labels sind die Datenstruktur die verwendet wird, um ein Netzwerk zu trainieren. Nach abgeschlossenem Training stellen die Features die (unbekannten) Parameter dar, mit denen das Netzwerk arbeiten kann. Generell gesagt steigt je nach Komplexität des Problems in der Regel auch die Anzahl an Features.

### Modell

Ein Modell beschreibt die Beziehung zwischen Features und Labels. Ein Model wird im Rahmen des maschinellen Lernens mit Features trainiert, die ein Label besitzen und lernt dabei diese Beziehung. Anhand der erlernten Beziehungen kann das Modell Schlussfolgerungen bei unbekannten/neuen Features ziehen, die kein Label

besitzen. Das Erlernen der Beziehung geschieht durch die Minimierung des Verlustes (s.u.) und wird als empirische Risikominimierung bezeichnet. [9]

Es gibt verschiedene Arten von Modellen, wovon im Rahmen dieser Arbeit hauptsächlich zwei Typen von Modellen unterschieden werden:

### **Regression/Klassifikation**

Ein lineares Regressionsmodell gibt kontinuierliche Werte zurück, während ein Klassifikationsmodell diskrete Werte (z.B. Kategorien) zurück gibt. Ein Klassifikationsmodell basiert in der Regel auf logistischer Regression. Dabei wird eine Sigmoid Funktion ( $y = \frac{1}{1+e^{-\sigma}}$ ) verwendet, um die Prognose ( $\sigma$ ) eines linearen Modells in den Wertebereich [0,1] zu bringen, was als Wahrscheinlichkeitswert interpretiert werden kann.

Außerdem wird in dieser Arbeit zwischen Regressions-/Klassifikationsmodell und neuronalem Netzwerk unterschieden.

Bei einem linearen Modell werden die einzelnen Parameter mithilfe von angelernten Skalaren verbunden und die Beziehung definiert. Hierbei muss die Zusammengehörigkeit von Werten vordefiniert sein (z.B. Feature Kreuzungen (Multiplikationen), Funktionen wie Sinus/Kosinus, Potenzfunktionen).

Dies ist in der Regel die schnellste Möglichkeit des maschinellen Lernens. Jedoch führt dies bei einer großen Anzahl an Features sehr schnell zu komplexen, nicht überschaubaren Zusammenhängen, die schwer manuell modellierbar sind.

An dieser Stelle können Neuronale Netze ein Vorteil bieten. Neuronale Netze besitzen eine weitere Struktur zwischen Inputs und dem Output. Es handelt es sich um sogenannte Neuronen, die sich in verschiedenen Layern befinden und die Inputs verbinden. Jedes Neuron besitzt zusätzlich zu den verschiedenen Gewichten der Verbindungen Aktivierungsfunktionen, die ebenfalls über das Training veränderte Werte bekommen. Diese Aktivierungsfunktionen führen zu dem Verlust der allgemeinen Linearität des Modells.

### **Trainings-Set/Validation-Set/Test-Set**

Um sicherzustellen, dass ein Modell generalisierbar bleibt, werden die Testdaten in drei Teilmengen unterteilt. Dabei wird das Training-Set direkt zum Trainieren des

Modells genutzt. Das Validation-Set wird genutzt, um ein Trainiertes Modell zu evaluieren und damit dem Overfitting (Spezialisierung des Models bezüglich der Testdaten) entgegenzuwirken. Das Test-Set wird nach abgeschlossenem Training zum manuellen Evaluieren des Models in einem zweiten Schritt genutzt. Der Hauptunterschied zwischen dem Validation- und Test-Set ist, dass das Validation Set bereits während der einzelnen Trainingsschritte genutzt wird. Hierbei beinhaltet die Schnittmenge aus Training- und Validation-Set generell 80-90% der Testdaten, während das Test-Set den kleineren Anteil, also die verbleibenden 10-20% darstellt.

## **Loss**

Als Verlust (engl. Loss) wird der Verlust durch schlechte Prognosen des Modells bezeichnet. Dieser kann beim Training mithilfe der etikettierten Trainings- und Validations-Daten bestimmt werden. Das Ziel des Trainings ist es, einen geringen Verlust über den Durchschnitt aller Werte zu erhalten.

### **Squared Loss (L2 loss) / Mean Square Error**

Beim Squared Loss handelt es sich um eine Verlust Funktion, die die Differenz zwischen dem Label und der Prognose quadriert.

Die mittlere quadratische Abweichung (engl. Mean Square Error; MSE) bezeichnet den durchschnittlichen quadratischen Verlust pro Stichprobe über den gesamten Datensatz. Die mittlere quadratische Abweichung ist folgendermaßen definiert (Kapitel 8, [10]):

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - \text{prognose}(x))^2$$

#### **Formel 2.4 Mean Square Error**

$D$  beschreibt den Datensatz mit  $N$  Stichproben  $\{(x_0, y_0), \dots, (x_N, y_N)\}$ , wobei  $x$  die Menge an Features und  $y$  das Label darstellt.

Squared Loss wird als Loss Funktion für lineare Regressionsmodelle genutzt.

## **Log Loss**

Der Logarithmische Verlust (engl. abk. Log Loss) wird als Verlustfunktion für logarithmische Regression wie bei Klassifiziermodellen verwendet.

Die Log Loss Funktion ist folgendermaßen definiert:

$$LogLoss = \sum_{(x,y) \in D} -y * \log(\text{prognose}(x)) - (1 - y) * \log(1 - \text{prognose}(x))$$

### **Formel 2.5 Log Loss**

$D$  beschreibt den Datensatz mit  $N$  Stichproben  $\{(x_0, y_0), \dots, (x_N, y_N)\}$  wobei  $x$  die Menge an Features und  $y$  das Label darstellt. Hierbei ist zu beachten, dass das Label  $y$  entweder 1 oder 0 ist. Die Prognose ( $\text{prognose}(x)$ ) entspricht einem Wert zwischen 0 und 1.

### **Konvergenzkurven**

Die Konvergenzkurve beschreibt die Veränderung des Verlustes des Modells über die Perioden bzw. Epochen des Trainings. In der Regel wird diese für das Training-Set sowie das Validation-Set ermittelt.

### **Optimizer**

Die Aufgabe des Optimierungsalgorithmus ist es, den Verlust des Modells zu verringern und dadurch bessere Prognosen zu garantieren. Dieser wird beim Training des Modells angewendet. Optimierungsalgorithmen sind in der Regel spezifische Implementationen des Gradient Descent Optimizers.

### **Gradient Descent Optimizer**

Der Gradient Descent Optimizer ist ein Optimierer für generell konvexe Probleme.

Als Gradient wird hier ein Vektor von partiellen Ableitungen der (unabhängigen) Gewichte des Modells bezeichnet. Mithilfe dieses Gradienten kann die Richtung der größten bzw. kleinsten Steigung der Funktion ermittelt werden. Der Gradient wird als  $\nabla f$  notiert und ist für eine gegebene Funktion  $f(x, y)$  der Vektor der partiellen Ableitungen:

$$\nabla f(x, y) = \left( \frac{\partial f}{\partial x}(x, y), \frac{\partial f}{\partial y}(x, y) \right)$$

### **Formel 2.6 Gradient, partielle Ableitungen der Gewichte**

Der Gradient Descent Optimizer nutzt die negative Richtung des Vektors (Richtung der geringsten Steigung) sowie dessen Betrag multipliziert mit der Learning Rate (s.u.), um schnellstmöglich den Verlust zu minimieren.

## **Learning Rate**

Bei der Learning Rate handelt es sich um die Schrittgröße mit der ein Optimierer die Verlust-Funktion minimiert. Beim Gradient Descent Optimizer wird z.B. die Learning Rate auf den Betrag des Gradienten multipliziert. Das skalare Ergebnis dient als Faktor für die Optimierung in Richtung des Gradienten.

Generell gilt, dass eine sehr kleine Learning Rate eine längere Laufzeit beim Training veranlasst, während eine zu hohe Learning Rate dazu führen kann, dass das Optimum nicht erreicht wird. Jedes Problem hat eine eigene optimale Learning Rate (für eindimensionale Probleme ist dies die Inverse 2. Ableitung von  $f(x)$ ;  $\frac{1}{f''(x)}$ ).

## **Batch/Batch Size**

Ein Batch beschreibt die Menge an Stichproben, die in einer Iteration des Modelltrainings genutzt werden soll. Bei der Batch Size handelt es sich um die Anzahl der Stichproben in der oben beschriebenen Menge. Generell gilt, dass es effizienter ist, den Verlust mithilfe eines kleinen Batches zu berechnen. Ein Batch besteht in der Regel aus 10 bis 1000 Stichproben.

## **Regularization**

Generell ist ein größeres/komplexeres Modell stärker auf die Trainingsdaten spezialisiert, wodurch es schnell zu Overfitting kommen kann. Um dieser Spezialisierung entgegenzuwirken, wird die Regularisierung (engl. Regularization) genutzt, um je nach Komplexität des Modells das Modelltraining anzupassen, ähnlich wie es die Verlust-Funktion tut. Dafür gibt es verschiedene Arten der Regularisierung:

**L1 Regularization** bestimmt die Komplexität des Modells anhand der Summe der absoluten Gewichte. Dadurch werden Features, die irrelevant oder kaum relevant sind aus dem Modell entfernt.

**L2 Regularization** bestimmt die Komplexität des Modells anhand der Summe der quadrierten Gewichte. Insbesondere bewirkt es bei Ausreißern, dass diese nicht so stark auf das Modell wirken.

**Dropout Regularization** bewirkt, dass z.B. eine bestimmte zufällige Auswahl an Layern eines neuronalen Netzwerkes in einem einzelnen Gradientenschritt entfernt werden.

**Early Stopping** ist eine Methode, bei der das Training beendet wird, bevor der Verlust beim Validation-Datensatz wieder zu steigen beginnt.

Mithilfe der Regularisierungsrate ( $\lambda$ ) kann die Stärke der Regularisierungsfunktion auf das Modelltraining beeinflusst werden, so dass sich die Verlust Gleichung folgendermaßen ändert; dies wird als „Strukturierte Risikominimierung“ bezeichnet:

$$\text{minimize}(\text{loss function} + \lambda(\text{regularization function}))$$

### Formel 2.7 Strukturierte Risiko Minimierung

#### Threshold

Klassifikationsmodelle geben als Prognose einen Prozentwert (durch die Sigmoid-Funktion) zurück. Um diesen zu einer Klassifikation zwischen zwei Kategorien zu konvertieren, wird ein Schwellwert (engl. Threshold Value) genutzt, mithilfe dessen die Prognose einer der beiden Kategorien zugeordnet werden kann.

#### Evaluation

Zur Evaluation des trainierten Modells können verschiedene Metriken verwendet werden. Im Folgenden werden die in dieser Arbeit verwendeten Metriken vorgestellt und kurz erläutert.

#### Accuracy/Precision/Recall/ROC Curve/AUC

Die Genauigkeit (engl. Accuracy) ist definiert als der Prozentsatz, den das Modell richtig prognostiziert.

$$\text{Accuracy} = \frac{\text{Anzahl der korrekten Prognosen}}{\text{Gesamtanzahl der Prognosen}} = \frac{TP + TN}{TP + FP + TN + FN}$$

### Formel 2.8 Accuracy

Bei der Nutzung der Genauigkeit ist zu beachten, dass diese Metrik nicht stabil gegenüber nicht ausbalancierten Datensätzen ist, beispielsweise wenn Klasse 1 zu 2% vertreten ist, während Klasse 2 zu 98% vertreten ist.

Die Präzision (engl. Precision) beschreibt den Anteil der positiv prognostizierten Stichproben die korrekt sind ( $\text{Precision} = TP / (TP+FP)$ ).

Die Treffer-Quote (engl. Recall) beschreibt den Anteil der positiven Stichproben die korrekt prognostiziert wurden. ( $\text{Recall} = TP / (TP+FN)$ ).

Eine ROC-Kurve (Grenzwertoptimierungskurve; engl. Receiver Operating Characteristic Curve) kann genutzt werden, um die Performance eines

Klassifikationsmodells bei allen Klassifikationsschwellwerten zu plotten. Die Ordinate entspricht der Sensitivität (korrekt positiv Klassifizierungen/Recall) und die Abszisse der 1-Spezifität (korrekt negativ Klassifizierung; engl. False Positive Rate;  $FPR = FP / (FP + TN)$ ) des Modelles. Die Fläche unter der geplotteten Kurve wird als AUC (engl. Area under the Roc Curve) bezeichnet. Dieser Wert kann als weitere Metrik genutzt werden, um die Performanz des Modells zu bestimmen. Bei einem Modell, dessen Prognosen immer richtig sind, ist der AUC-Wert 1 (bzw. 0 bei einem Inversen Ergebnis), in allen anderen Fällen liegt der Wert zwischen diesen beiden Werten [0,1], wobei 0.5 der schlechteste Fall ist (zufälliges Klassifizierungsmodell).

### **Confusion Matrix**

Durch die Zuordnung mithilfe des Schwellwertes kann es zu folgenden Zuständen kommen, die in der Folgenden Confusion Matrix gezeigt werden.

**Tabelle 1** Confusion Matrix

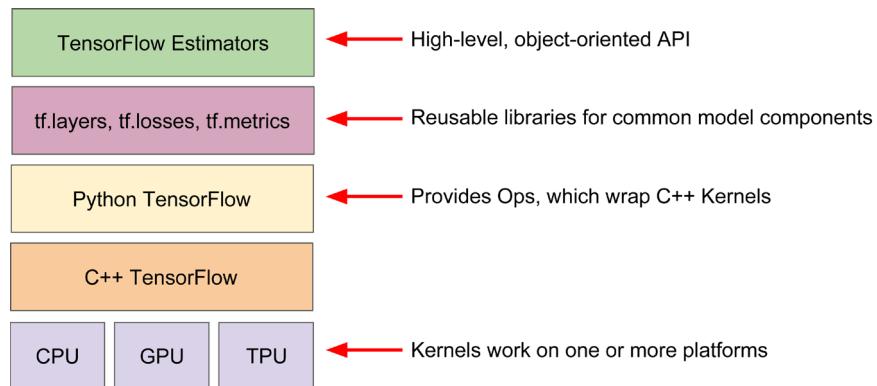
<b>True Positive (TP):</b> Das Modell prognostiziert korrekt ein positives Ergebnis.	<b>False Positive (FP):</b> Das Modell prognostiziert ein positives Ergebnis, welches jedoch nicht ein negatives Ergebnis ist.
<b>False Negative (FN):</b> Das Modell prognostiziert ein negatives Ergebnis, welches jedoch ein positives Ergebnis ist.	<b>True Negative (TN):</b> Das Modell prognostiziert korrekt ein negatives Ergebnis.

## 2.5. Tensorflow

Im Rahmen der Forschungsarbeit dieser Thesis wird das Tensorflow Framework <sup>11</sup> genutzt.

Hierbei handelt es sich um eine Bibliothek zur datenstromorientierten Programmierung, welche eine der meist verwendeten Bibliotheken ist, die im Rahmen des maschinellen Lernens verwendet wird. Die Bibliothek bietet gleichzeitig Low-Level und High-Level Funktionen zum Definieren, Trainieren und Ausführen von Modellen. Außerdem werden Möglichkeiten des Exportes und Importes geboten.

In der folgenden Grafik von der Tensorflow Website wird die Hierarchie des Tensorflow Toolkits gezeigt.



**Abbildung 14** Tensorflow Framework Hierarchie Bildquelle: tensorflow.org

Im Rahmen dieser Arbeit wird hauptsächlich auf der Ebene des TensorFlow Estimators gearbeitet und es werden die vordefinierten Modelle (linear regressor, linear classifier & neural networks) genutzt, da es hauptsächlich auf die Parameter der Modelle ankommt, die in verschiedenen Testreihen verändert werden.

Der folgende Code (Pseudo-Code) Ausschnitt zeigt, wie Tensorflow in Python verwendet wird, in dem ein Neuronales Netzwerk erstellt und trainiert sowie ausgewertet wird. Es wurden extra viele der in Abschnitt 2.4 beschriebenen Elemente in diesem Beispiel integriert.

---

<sup>11</sup> <https://www.tensorflow.org/>

```

1. import tensorflow as tf
2.
3. # Layer und Neuronen
4. layer = [600, 600, 400, 400]
5.
6. # Optimizer
7. opt = tf.train.AdamOptimizer()
8.
9. # erstellen eines Neuronalen Netzwerks als Klassifizierer
10. estimator = tf.estimator.DNNClassifier(feature_columns,optimizer=opt,
    hidden_units=layer)
11.
12. # trainieren des Models mit Beispieldaten
13. estimator.train(input_fn=train_input_fn, steps=2000)
14.
15. # mithilfe des trainierten Models Prognosen erstellen
16. Predictions = estimator.predict(input_fn=predict_input_fn)

```

**Listing 2.1** Tensorflow Erstellen, Trainieren und nutzen eines Neuronalen Netzwerkes

In sämtlichen Testläufen wurde die Tensorflow Version 1.14.0 in der GPU-Variante genutzt.

Als Referenz zu den Hauptimplementierungen wurden einige Testdurchläufe mithilfe des Keras-Frameworks<sup>12</sup> durchgeführt, welches eine höhere Abstraktion für Neuronale Netze liefert und für schnelle Experimente ausgelegt ist.

Als Umgebung wurde Python gewählt. Dies stellte sich anfänglich an einigen Stellen als problematisch dar, da Python Scripts zur Laufzeit ausgeführt werden und nicht kompiliert werden incl. der entsprechenden Compiler-Checks. Dies hatte zur Folge, dass an einigen Stellen semantische Fehler oder fehlende Bibliotheks-Importe dazu führten, dass ein Training nach sehr langer Berechnungszeit nicht komplett evaluiert werden konnte.

Insbesondere aus diesem Grund wurde direkt nach erfolgreichem Modelltraining dieses exportiert, so dass dies ggf. in einem weiteren Schritt manuell evaluiert werden konnte und das trainierte Modell nicht verloren ging.

Weiter wichtige verwendete Python Bibliotheken sind: numpy für die Verarbeitung der Datenstreams, matplot für die Bilderstellung der Verteilungsplots und panda für die Arbeit mit den großen Datenmengen.

---

<sup>12</sup> <https://keras.io/>

# 3.

## Datensatz Erstellung und Analyse

Dieser Abschnitt beschreibt die verwendeten Datensätze die exportiert wurden aus Szenenbeschreibungen, die die darin enthaltene Geometrie sowie die relevanten Szene Parameter beinhalten.

Es wird dargestellt wie die Testdaten in Features und Label konvertiert werden und die Datensätze in Trainings-, Validation- und Testdatensatz aufgeteilt werden. Außerdem wird die Verteilung innerhalb dieser Teilmengen analysiert.

### 3.1. Datensatz für Szenengeometrie Schnitttests

Dieser Datensatz wurde erstellt mit dem Ziel Daten zu generieren, mit denen ein Modell trainiert werden kann, um Schnitte von Strahlen mit der der Szenen Geometrie zu erlernen. Das trainierte Modell soll dann für einen gegebenen Strahl ausgeben, ob dieser die Szenengeometrie schneidet oder nicht. Dabei soll das Modell eine virtuelle 3-dimensionale Silhouette der Szenengeometrie erlernen.

#### 3.1.1. Testszene & Geometrie

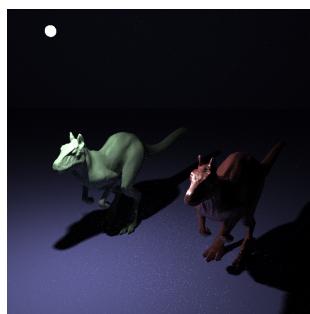


Abbildung 15 Killeroo-Simple Scene

Bei der verwendeten Testszene (siehe Abbildung 15), aus der der Datensatz für diese Arbeit exportiert wurde, handelt es sich um die “Killeroo”-Simple Scene, die eine bekannte Beispieldaten des pbrt-Renderers ist. Es liegt eine einfache Szene mit 2 Mesh-Modellen, einer Grundfläche sowie einer Hintergrundfläche und einer Kugel

vor. Außerdem wurden die Render Parameter so angepasst, dass die Anzahl der zu exportierenden Strahlen nicht zu groß wird. Dies kann insbesondere in Abbildung 15 an den Artefakten auf dem Boden erkannt werden.

Mit dem in Abschnitt 2.2 beschriebenen pbrt-System kann diese Szene auf obengenannter Hardware innerhalb von unter einer Minuten gerendert werden, unter Ausnutzung sämtlicher Kerne.

**Tabelle 2** Übersicht Szenenparameter Testszene Datensatz für Schnitttests

Kamera	Look-At-Matrix	400 20 30	
	(ViewPos, Look-At, Up)	0 63 -110	
		0 0 1	
Rotation		-5° um die Z-Achse	
Field-Of-View		39	
Objekte	Position		
	Killerroo Tier #1 (Mesh)	100 200 -140	[150,100,200]
	Killerroo Tier #2 (Mesh)	-100 200 -140	[150,100,200]
	Grundfläche (Triangle Mesh)	0 0 -140	2000x2000 horizontal
	Hintergrundfläche (Triangle Mesh)	-400 0 -140	2000x2000 vertikal
	Kugel	150 120 20	Radius 3

Tabelle 2 zeigt die Szenenparameter dieser Szene. Eine Kamera ist an der Position (400 20 30) und hauptsächlich in negative X-Richtung ausgerichtet. Der normalisierte (gerundete) Richtungsvektor der Kamera kann aus der Look-At-Matrix abgeleitet werden und lautet (-0.94, 0.1, -0.33). Es ist dabei zu beachten, dass das pbrt-System ein linkshändiges Koordinatensystem nutzt wie in Kapitel 2.1 des Buches beschrieben. Die Kamera ist um -5 Grad um die Z-Achse gedreht. In dieser Szene stellt die Z-Achse den Up-Vektor dar. Bei der Kamera handelt es sich um eine perspektivische Kamera mit einem Sichtfeld von 39°.

Das gerenderte Bild der Szene (Abbildung 15) hat eine quadratische Auflösung von 700x700 Pixeln, eine Dateigröße von 1,92 MB und besitzt das von Industrial Light &

Magic entwickelte high dynamic-range Bildformat „exr“.

Es wird ein Pathracing Algorithmus angewendet und Pixelsample von 128 verwendet.

Die Kugel befindet sich an der Position (150 120 20) und besitzt den Radius 3. Sie stellt eine Flächenlichtquelle dar, die Licht emittiert.

Die Grundfläche liegt in der XY-Ebene und auf beiden Achsen jeweils im Wertebereich von [-1000,1000].

Die beiden Mesh-Objekte sind gegenüber den Definitionen um den Faktor 0.5 skaliert und jeweils um -60° um die Z-Achse rotiert.

Das erste Modell befindet sich an der Position (100, 200, -140), das 2. Modell an der Position (-100, 200, -140).

Da die Material- und Lichteigenschaften keine Relevanz<sup>13</sup> für die Schnitttests haben, wird auf eine genauere Analyse dieser Parameter im Rahmen dieser Arbeit verzichtet.

### 3.1.2. Raw Daten

Dieser Abschnitt handelt davon, wie die exportieren Raw Daten aussehen und wie diese in Feature Vektoren konvertiert werden.

#### Raw Data

```
0 : {   origin: [(float) origin.x, (float) origin.y, (float) origin.z]
        direction: [(float) direction.x, (float) direction.y, (float) direction.z]
        hit: (boolean) hit }
```

Bei den Raw Daten handelt es sich um Strahlentests bzw. deren Ergebnisse. Wie in Abschnitt 2.1. beschrieben, basiert das Raytracing auf der Strahlenverfolgung und ein Strahl kann durch den Ursprung (*origin*) und die Richtung (*direction*) definiert werden. Bei diesen beiden Eigenschaften handelt es sich jeweils um Vektoren mit 3 Komponenten. Des Weiteren befindet sich in den Raw Daten noch das Feld „*hit*“, welches angibt ob der Strahl eine der Szenengeometrien schneidet oder nicht.

Wenn der Wert des Feldes „*hit*“ 0 ist, wird die Szenengeometrie nicht geschnitten. Wenn der Wert 1 ist, trifft der Strahl die Szenengeometrie.

Wie man an der Struktur der Raw Daten erkennen kann, sind sämtliche Parameter, die als Features verwendet werden sollen bereits Gleitkomma-Werte, sodass für diese kein Mapping genutzt werden muss.

---

<sup>13</sup> Displacement liegt in dieser Szene nicht vor.

### 3.1.3. Datenteilung und Datenverteilung

Der komplette Datensatz aller Schnitttest beim Rendern der Szene ist eine CSV-Datei mit insgesamt ~16.8 Mio Schnitttest. Aus den Renderstatistiken konnte abgelesen werden, dass es sich davon bei 6,16 Mio. Strahlen um Schattenstrahlen handelt. Von der Kamera selber sind davon 3.9 Mio. Strahlen ausgegangen.

Die Dateigröße des Datensatzes beträgt ~1.2 Gb.

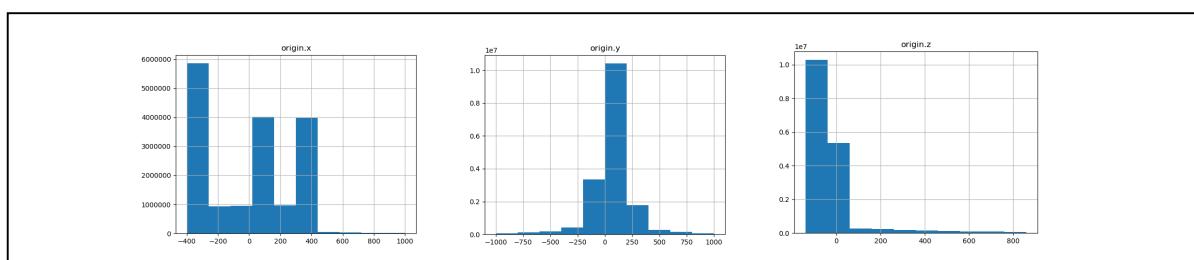
Um die Speichergröße zu reduzieren, wurde ein kleinerer Datensatz, der aus einer Teilmenge des kompletten Datensatz besteht, erstellt. Der kleinere Datensatz umfasst ungefähr eine Million Daten und besitzt eine Größe von etwas mehr als 70 MB. Dieser Datensatz wurde für die in Abschnitt 5 beschriebenen Modell-Trainings genutzt, um die Trainingszeit in einem überschaubaren Rahmen zu halten.

Die Datenplots in den folgenden Unterabschnitten zeigen die Verteilung der Features innerhalb der Datensätze und beschreiben diese.

#### 3.1.3.1. Kompletter Datensatz

Der Komplette Datensatz enthält insgesamt 16.844.643 Daten.

Die Schnitttests, die exportiert wurden, sind die, die beim Rendern der Szene mit den beschriebenen Parametern verwendet wurden. Die folgende Abbildung zeigt die Verteilung der Ursprungs-Koordinaten der Strahlen.

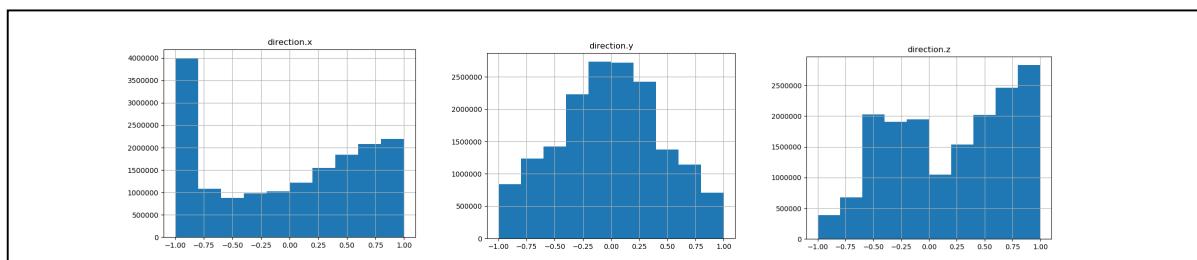


**Abbildung 16** Plot Ursprungskoordinaten der Strahlen; Kompletter Datensatz

Es ist zu erkennen, dass ein großer Teil der x-Koordinaten der Ursprungspunkte ungefähr bei den Werten -400 und 400 liegen. Vergleicht man Tabelle 2 damit, wird ersichtlich, dass diese Werte der Kameraposition (400) und der Hintergrundfläche (-400) entsprechen. Die weiteren Szenengeometrien sind durch den mittleren Peak zwischen 0 und 200 auf die Lichtquelle zurückzuführen, die sich an der Position x=150, y=120 & z= 20 befindet (vgl. Tabelle 2). Die weiteren kleineren Verteilungen repräsentieren die weiteren Objekte in der Szene.

An dem Plot der Z-Koordinate in Abbildung 16 kann erkannt werden, dass sich sämtliche Ursprungspunkte um den 0 Wert befinden. Insbesondere sind keine extremen Ausreißer bei der Z-Richtung in die negative Richtung zu erkennen. An dieser Stelle ist nochmals zu beachten, dass sich es bei der Z-Achse in dieser Szene um den Up-Vektor handelt und diese daher die „Höhen“-Werte beschreibt. Somit beschränkt die Grundfläche der Szene die Ursprungspunkte auf ein Minimum von -140 (vgl. Abbildung 16).

Die Verteilung auf der Y-Achse zeigt eine Art Normalverteilung um den Wertebereich 0-250. Der Wertebereich von -1000 bis 1000 lässt sich durch die Größe der Grundfläche erklären.

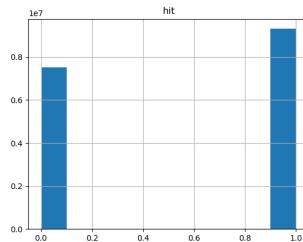


**Abbildung 17** Plot Richtungsvektoren der Strahlen; Kompletter Datensatz

Abbildung 17 zeigt die Verteilung der Richtungsvektoren bzw. deren Komponenten. Es ist zu sehen, dass ein großer Anteil an Richtungsvektoren in die negative X-Richtung zeigt. Dies ist auf die Orientierung der Kamera zurückzuführen, die von dem Punkt  $x=400$  in Richtung der negativen X-Achse gerichtet ist.

Es ist zu erkennen, dass ein geringer Anteil an Strahlen eine negative Z-Richtung von -1 besitzen. Die Strahlen haben tendenziell eine Ausrichtung zur positiven Z-Achse. Die Verteilung im Intervall [-0.5, 0.0] ist hauptsächlich auf die initiale Neigung der Kamera zurückzuführen.

Der Plot der Y-Richtungen weist wieder eine Normalverteilung auf, wie bei den Y-Ursprungs-Koordinaten und zeigt, die hauptsächlich frontale Verteilung der Strahlen.



**Abbildung 18** Plot Verteilung positive & negativer Schnitttests der Strahlen; Kompletter Datensatz

Der Plot in Abbildung 18 zeigt die Verteilung der Schnitttest Ergebnisse. Mit anderen Worten, ob ein Strahl die Szenen Geometrie getroffen hat (1) oder nicht (0).

Es ist zu erkennen, dass die Verteilung nicht ganz ausgewogen ist, da eine Tendenz positiver Schnitttests besteht<sup>14</sup>.

Von den ~16 Mio. Schnitttests besitzen ~7 Mio. ein negatives Label und ~9. Mio. ein positives.

**Tabelle 3** Übersicht Parameter-Verteilung im kompletten Datensatz

	Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	Hit
Count	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0
Mean	-12.8	56.6	-42.3	0.0	0.0	0.2	0.6
Std	315.9	185.5	144.6	0.7	0.5	0.5	0.5
Min	-400.0	-999.9	-140.0	-1.0	-1.0	-1.0	0.0
25%	-400.0	1.2	-150.0	-0.8	-0.3	-0.3	0.0
50%	64.0	54.8	-91.6	0.1	0.0	0.3	1.0
75%	226.9	117.5	30.0	0.6	0.3	0.7	1.0
Max	1000.0	1000.0	860.0	1.0	1.0	1.0	1.0

An der Tabelle 3 und den vorherigen Verteilungsdiagrammen kann erkannt werden, dass sämtliche Features und Labels frei von unerwarteten Werten („Magic Values“) sind<sup>15</sup>, die außerhalb der erwarteten Wertebereiche liegen. Daher muss dies nicht

<sup>14</sup> Ein höherer Anteil an positiven Schnitttests spricht für den verwendeten Raytracing Algorithmus, da negative Schnitttests Zeitkosten und möglichst zu minimieren sind.

<sup>15</sup> Dies ist zurückzuführen darauf, dass es sich bei den Daten um geometrische Berechnungen und nicht zufällige Erhebungen handelt.

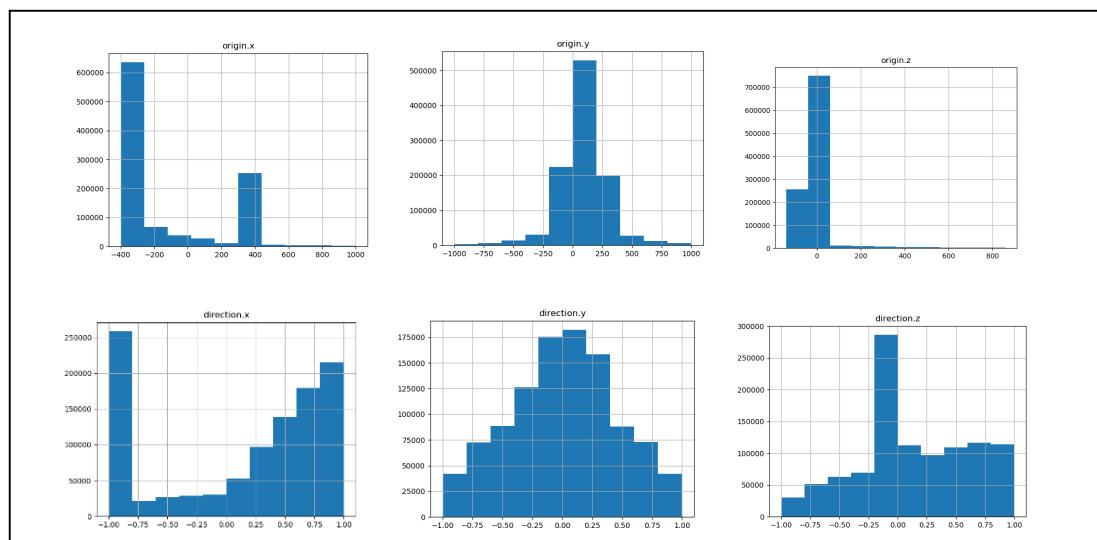
beim Feature Engineering und Training besonders beachtet werden.

In Tabelle 4 sind die beschriebenen Verteilungen numerisch zusammengefasst und es wurden jeweils der Mittelwert (engl. Mean), die Standardabweichung (Std), der Minimum und Maximum-Wert sowie der 25-, 50- und 75-Percentil bestimmt und in der Tabelle eingetragen.

### 3.1.3.2. Kleiner Datensatz

Die kleine Teilmenge des kompletten Datensatzes aus Abschnitt 3.1.3.1 beinhaltet die die ersten 1.048.575 Daten. Insbesondere ist die dadurch neu erstellte CSV-Datei lesbar für Programme wie Microsoft Excel<sup>16</sup> und es wird ermöglicht, einen schnelleren, detaillierten Einblick in den Datensatz zu bekommen. Da sich dies auf die Verteilung auswirken könnte, sind im Folgenden die Verteilungen der Teilmenge geplottet.

#### Features



**Abbildung 19** Oben Plot Ursprungskoordinaten, Unten Plot Richtungsvektoren der Strahlen; Kleiner Datensatz

Abbildung 19 zeigt die Verteilungen der Ursprungskoordinaten und der Richtungsvektoren in dem kleinen Datensatz.

Es fällt auf, dass sich der Plot der Ursprungs-X-Koordinate stark von dem Plot im

<sup>16</sup> In Microsoft Excel ist ein Tabellenblatt auf 1.048.576 Zeilen und 16.384 Spalten beschränkt. Da die erste Zeile in der CSV-Datei die Spalten beschreibt, bleiben 1.048.576 Zeilen für die reinen Daten. <https://support.office.com/en-ie/article/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3>

kompletten Datensatz (vgl. Abbildung 16) unterscheidet. Insbesondere ist festzustellen, dass viele der Daten in dem Bereich zwischen -200 und 200 nicht vertreten sind. Da sich in diesem Bereich die Szenengeometrie der beiden Mesh-Objekte (die Tiere) befinden, beschreibt dieser Teil der Daten die indirekte Reflexion und direkten Lichteinflüsse, die auf die Objekte wirken. Insbesondere die Position der Lichtquelle ist sehr unterrepräsentiert.

Wenn ein System erstellt werden soll, dass diese Maschinen-Lern-Beschleunigung für die Berechnung der Sichtbarkeit zwischen Lichtquellen und Objekten (z.B. Schattenstrahlen) genutzt werden soll, sind genau diese Daten relevant.

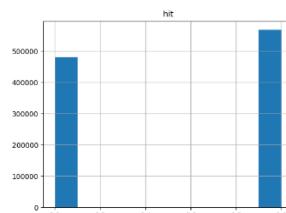
Daraus ist zu erkennen, dass insbesondere die Strahlen der unteren Rekursionslevel repräsentiert sind.

Dieser Umstand verändert jedoch nicht, dass dieser Datensatz auch für Schattenstrahlen genutzt werden kann, wenn die Kamera oder die Wand im Hintergrund als Lichtquelle angenommen wird. In dem Fall sind die ausgehenden Strahlen gut repräsentiert.

Aus der Beobachtung, dass hauptsächlich die Strahlen ausgehend von der Kamera und der Hintergrundfläche vertreten sind und der Verteilung der Richtungsvektorkomponenten, kann erkannt werden, dass die von diesen Objekten ausgehen. Hierdurch ist eine größere Ansammlung von Richtungsvektoren mit der X-Richtung +1 und -1 in Abbildung 19 zu erkennen.

Bei dem Richtungsvektorplot ist ebenfalls festzustellen, dass bei der Z-Koordinate der Wertebereich zwischen -0.25 und 0 sehr viel stärker vertreten ist als die restlichen Wertebereiche. Diese starke Ansammlung in dem Bereich ist im kompletten Datensatz nicht so stark zu erkennen.

Es ist zu vermuten, dass durch die Rotation der Kamera sowie die Look-At Matrix und der sehr stark präsenten Kamerastrahlen in diesem Datensatz diese Verteilung verursacht wird.



**Abbildung 20** Plot Verteilung positive & negativer Schnitttests der Strahlen; Kleiner Datensatz

Die Verteilung der positiven und negativen Schnitttestergebnisse, ähnelt der Verteilung des vollständigen Datensatzes (Abbildung 18). Der Datensatz enthält konkret ca. 470.000 negative Schnitttestergebnisse und ca. 530.000 positive Schnitttestergebnisse.

**Tabelle 4** Übersicht Parameterverteilung im kleinen Datensatz

	Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	Hit
Count	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0
Mean	-141.7	76.5	-4.5	0.1	0.0	0.1	0.5
Std	348.1	207.0	105.6	0.7	0.5	0.5	0.5
Min	-400.0	-999.9	-140.0	-1.0	-1.0	-1.0	0.0
25%	-400.0	-10.5	-11.0	-0.8	-0.1	-0.1	0.0
50%	-400.0	54.8	18.6	0.4	0.0	0.0	1.0
75%	396.7	186.3	30.0	0.8	0.5	0.5	1.0
Max	999.9	999.9	860.0	1.0	1.0	1.0	1.0

Zusammenfassend für diesen Abschnitt zeigt Tabelle 4 die Verteilung der Strahlen des kleinen Datensatzes mit konkreten Werten für den Mittelwert, die Standardabweichung, das Minimum und Maximum sowie die Perzentile.

Diese Teilmenge des vollständigen Datensatzes wird anhand der Verteilung für diese Forschungsarbeit als repräsentativ angesehen und dient als Datensatz für die folgenden Forschungsdurchläufe.

### 3.2. Datensatz für Sichtbarkeitstests

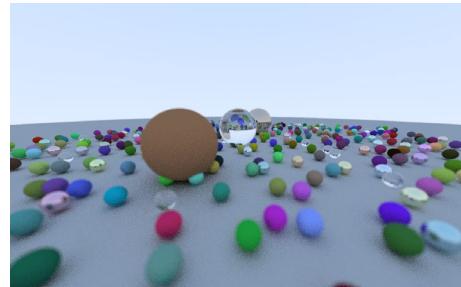
Dieser Datensatz wurde ebenfalls erstellt, mit dem Ziel Daten zu generieren, mit denen ein Modell trainiert werden kann, um Szenengeometrie zu erlernen.

Es wurde jedoch ein anderer, weniger genereller Ansatz gewählt.

Das Modell soll für zwei gegebene Punkte prognostizieren, ob eine Sichtbarkeit zwischen diesen Punkten besteht oder ob sich zwischen diesen Punkten verdeckende Szenengeometrien befinden.

### 3.2.1. Testszene & Geometrie

Für die Erstellung dieses Datensatzes wurde ein komplett eigener Raytracer programmiert. Dieser besitzt nicht die Komplexität des pbrt Renderers, ist jedoch insbesondere auf die Geometrie Ebene des Raytracer-Algorithmus ausgelegt und besitzt daher weniger komplexe Licht- und Shading-Komponenten.

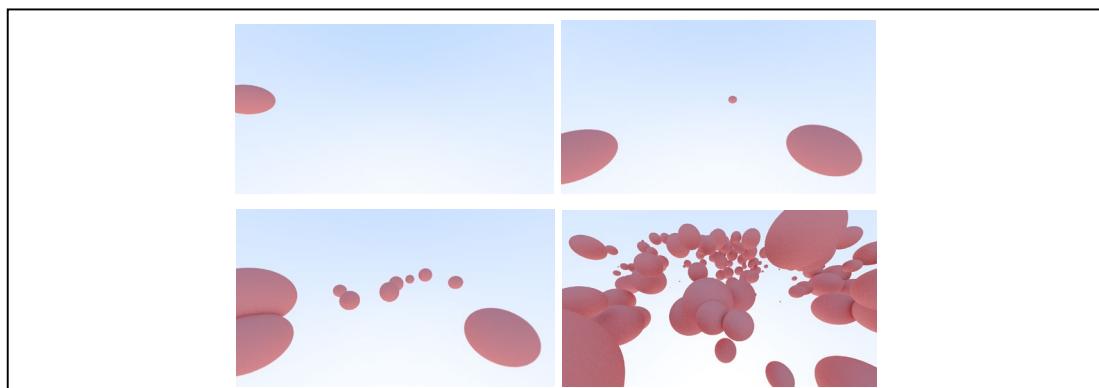


**Abbildung 21** Raytracer Demo Bild

Abbildung 21 zeigt, dass dieser dennoch fähig ist, einige Elemente der GI zu beachten und zu berechnen. Das Bild wurde in einer Auflösung von 1280x800 innerhalb von 40 Sekunden gerendert.

Im Gegensatz zu dem pbrt-Renderer, enthält dieser Raytracer eine eigene Komponente, um die Sichtbarkeit zweier Punkte innerhalb der Szene zu überprüfen. Für die Testszenen wurden lediglich Kugel-Geometrien verwendet, um eine schnelle Erstellung von Schnitttests, Renderergebnissen und Datenexport zu gewährleisten. Es wurde weiterhin mit einer floating point Genauigkeit gearbeitet.

Dieser Raytracer kann ebenfalls nur ppm Bilder exportieren, was für diesen Anwendungszweck ausreichend ist.



**Abbildung 22** Szenengeometrien für Sichtbarkeitstests

Oben Links Szene-ID 1 (1 Kugel), Oben Rechts Szene-ID 2 (3 Kugeln), Unten Links Szene-ID 3 (10 Kugeln), Unten Rechts Szene-ID 4 (300 Kugeln)

Abbildung 22 zeigt verschiedene Szenen, für die Daten exportiert wurden, um mithilfe derer die Sichtbarkeit von Punkten innerhalb dieses Raumes zu untersuchen. Hauptsächlich unterscheiden sich die Szenen darin, dass jeweils eine unterschiedliche Anzahl an Kugeln im Raum erstellt wurde. Die Kugeln sind jeweils mit einem pseudo-zufälligen Radius an unterschiedlichen Punkten im Raum angeordnet. Der Raum wird durch eine Bounding Box begrenzt mit den Ausmaßen, die in Tabelle 5 gezeigt sind.

**Tabelle 5** Szenen Bounding Box Ausmaße für Sichtbarkeitstests

	X	Y	Z
Min	-10	-2	-10
Max	10	2	10

Die Kugel besitzt eine geringere Varianz auf der Y-Achse, die, im Gegenteil zu der Szene aus Abschnitt 3.1, den Up-Vektor der Szene (im View-Space) beschreibt.

Die gesamte Box besitzt dementsprechend ein Volumen von 1600.

### 3.2.2. Raw Daten

Dieser Abschnitt beschreibt die Struktur der exportierten Raw Daten und wie diese in Feature Vektoren konvertiert werden können.

#### Raw Data

```
0 : {   A: [(float) A.x, (float) A.y, (float) A.z]
        B: [(float) B.x, (float) B.y, (float) B.z]
        d: [(float) d.x, (float) d.y, (float) d.z]
        l: (float) l
        v: (boolean) v }
```

Die Raw Daten in diesem Datensatz beinhalten die 2 Punkte (*A* und *B*), deren Sichtbarkeit bestimmt werden soll.

Zusätzlich wurde der Vektor Richtungsvektor von Punkt *A* zu Punkt *B* bestimmt und normalisiert, sodass *d* ein Einheitsvektor ist, der für die Sichtbarkeit der Punkte relevant ist. Der skalare Wert *l* beschreibt den Abstand von Punkt *A* zu Punkt *B*.

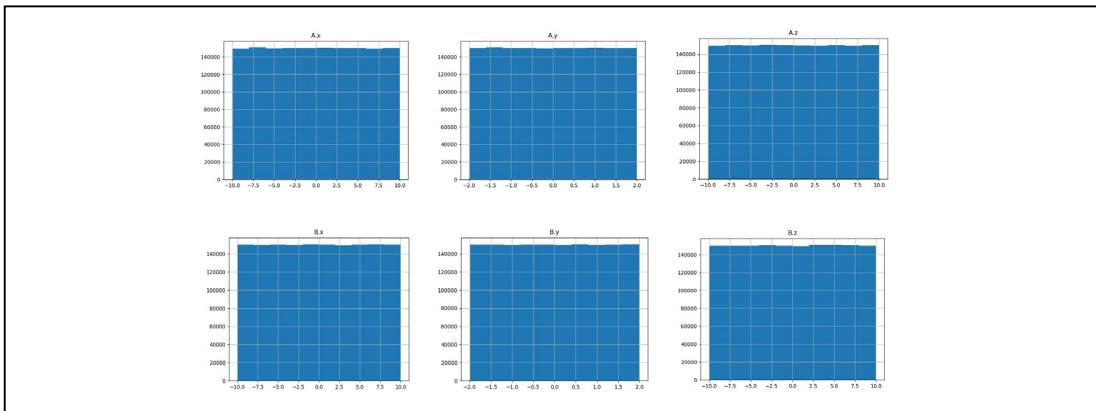
Alle beschriebene Vektoren bestehen aus 3 floating Point Werten und auch der

Skalare Wert  $I$  ist ein floating Point Wert.

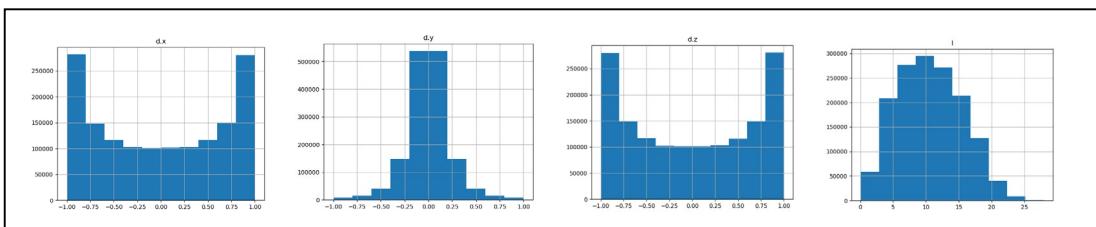
Der binäre Wert  $v$  gibt an, ob eine Sichtbarkeit zwischen den Punkten  $A$  und  $B$  besteht. Ein Wert von 1 beschreibt die Punkte können sich gegenseitig sehen, ein Wert von 0 gibt an, dass keine Sichtbarkeit zwischen den Punkten besteht. In diesem Fall befindet sich dementsprechend eine Szenengeometrie zwischen den Punkten, die die Sichtbarkeit zwischen diesen verhindert.

### 3.2.3. Datenverteilung

Für sämtliche Datensätze wurde die Sichtbarkeit von 2,5 Mio. Punkt-Tupeln getestet. Die Punkte wurden ebenfalls Pseudo-Zufällig innerhalb der Bounding Box aus Tabelle 5 erstellt mit einem Mindestabstand von 0.001 zueinander. Abbildung 23 zeigt die Gleichverteilung der Punkte im Raum.



**Abbildung 23** Sichtbarkeitstest Punktverteilungen; Oben Punkt A; Unten Punkt B

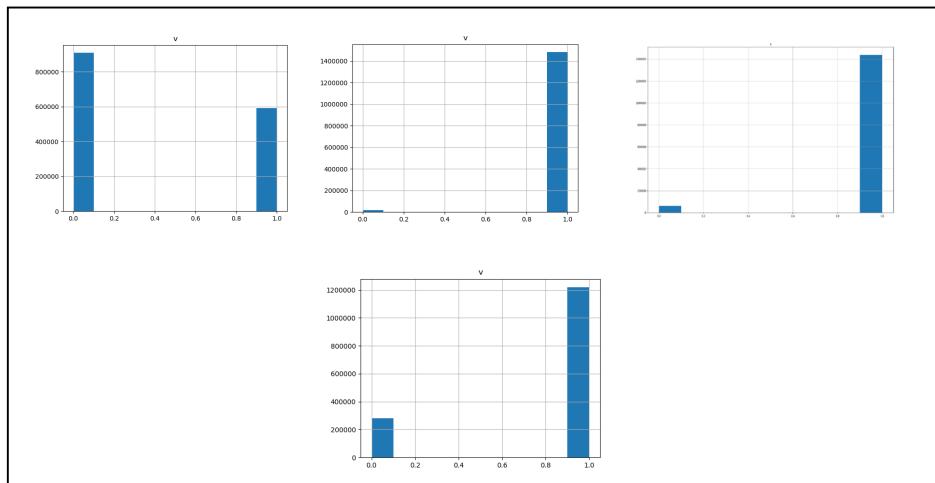


**Abbildung 24** Sichtbarkeitstest; Links Richtungsvektor  $(x,y,z)$  Verteilung, Rechts Punktabstandsverteilung

Abbildung 24 zeigt die Verteilung der Richtungsvektoren von Punkt  $A$  zu Punkt  $B$  sowie den Abstand der Punkte zueinander.

Es ist zu erkennen, dass durch geringen Ausdehnung auf der Y-Achse ein stärkere Verteilung der Y-Komponente um den Wert 0 vorliegt und bei der X und Z-Achse jeweils um die Werte -1 und 1. Die Punkte besitzen hauptsächlich eine

Normalverteilung des Abstands um den Wert 10, das Minimum liegt bei 0.001 und das Maximum bei 30.



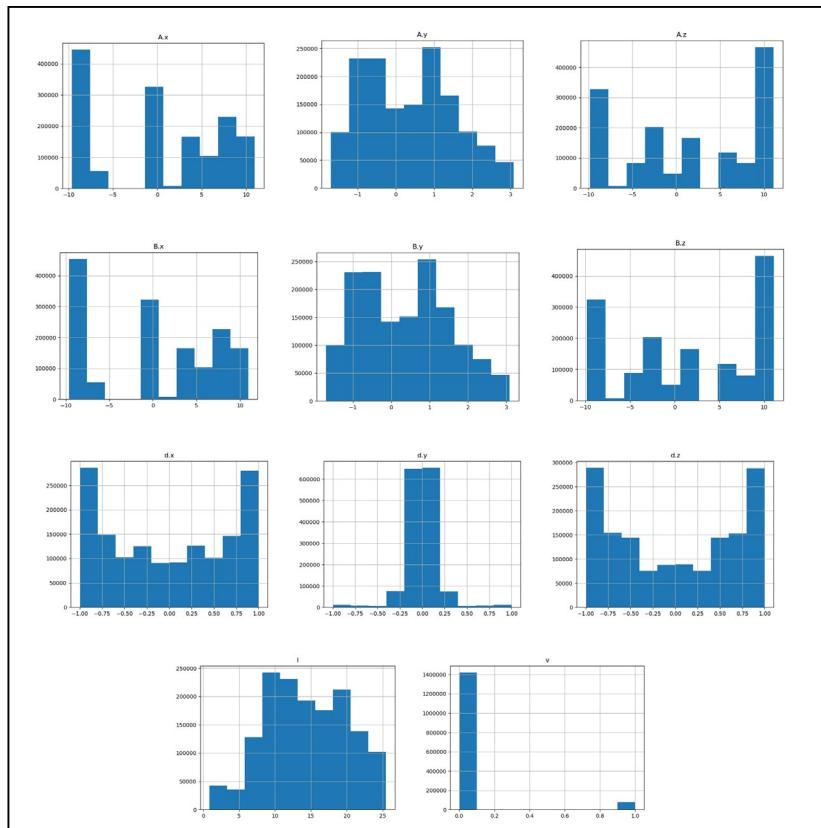
**Abbildung 25** Sichtbarkeitstest Verteilung Sichtbarkeit

Oben Links Szene-ID 4 & Zufällige Verteilung, Oben Mitte Szene-ID 1 & Zufällige Verteilung, Oben Rechts Szene-ID 2 & Zufällige Verteilung, Unten Szene-ID 3 & Zufällige Verteilung

Die Plots in Abbildung 25 zeigen die Verteilungen der Sichtbarkeiten bei den verschiedenen verwendeten Szenen. Bei 300 Kugeln liegt ein größerer Anteil an Verdeckungen vor, was auf die große Anzahl an Kugeln in dem Raum zurückzuführen ist. Ca. 60% der Punkt-Tuples wurde als verdeckt berechnet und 40% als sichtbar. Anhand der weiteren Plots kann erkannt werden, dass ein starker Überhang an positiven Sichttests vorliegt. Der Anteil an negativen Sichttests steigt, wie zu erwarten, mit steigender Szenenkomplexität. In Szene-ID 1 ist eine Kugel vorhanden, mit einem Radius von ~1 was einem Volumen von ca. 4,18 entspricht, die Anzahl an negativen Sichtbarkeitstests liegt unter 6,000. In Szene-ID 2 befinden sich 3 Kugeln und ca. 10,000 Schichtbarkeitstests sind negativ und in Szene-ID 3 befinden sich 10 Kugeln und ca. 28.000 negative Sichtbarkeitstests.

Es kann erkannt werden, dass in den 3 exportierten Datensätzen, mit der geringen Anzahlen an Kugel, die negativen Sichtbarkeitsresultate extrem unterrepräsentiert sind aufgrund der zufälligen Verteilung im Raum zu dem eingeschränkten Volumen im Raum, welches durch die Kugeln belegt ist.

Um auch bei einer geringeren Anzahl an Kugel eine möglichst annähernd ausgeglichene Verteilung zu bekommen, wurden zwei weitere Datensätze erstellt. Bei diesen sind die Punkte auf den Geometrieoberflächen jeweils verschiedener Geometrien (hier Kugeln) verteilt.



**Abbildung 26** Sichtbarkeitstest Verteilung Sichtbarkeit; Szene-ID 3 & Zufällige Verteilung auf Kugeloberflächen

1. Reihe Verteilung Punkt A, 2. Reihe Verteilung Punkt B, 3. Reihe Verteilung Richtungsvektoren, 4. Reihe Links Abstand Punkt A und Punkt B, 4. Reihe Rechts Sichtbarkeitsverteilung

Abbildung 26 zeigt eine komplett zufällige Verteilung der Punkte A und B auf den Kugeloberfläche der 10 Kugeln in Szene-ID 3. Es ist zu beachten, dass ein geringer Offset ergänzt wurde, um eigene Verdeckungen<sup>17</sup> durch floating point Ungenauigkeiten an den Punkten zu vermeiden.

An den Punktverteilungen kann erkannt werden, dass die Punkte nicht mehr gleichverteilt im Raum sind, sondern nur auf den Kugeloberflächen liegen.

Die Verteilung der Richtungsvektoren weist keine große Veränderung gegenüber der zufälligen Punktverteilung auf und auch die Längen haben weiterhin eine annähernde Normalverteilung um den Wert 10.

Es ist jedoch zu beobachten, dass die Sichtbarkeit, das Komplement gegenüber der vorherigen Verteilung aufweist. Dies kommt dadurch zustande, dass ein Großteil der Punkte auch auf der, der zweiten Kugel abgewandten Seite erzeugt werden und

---

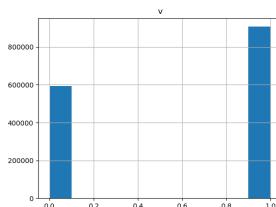
<sup>17</sup> Engl. Self-Intersections; Ähnlich wie Z-Fighting bei Rasterisierung oder Shadow-Mapping Artefakte.

somit die Kugel selber die Sichtbarkeit verhindern.

Um diesem Problem entgegen zu wirken, wurde eine weitere Punktverteilung erzeugt, bei der die Punkte auf den zueinander zugewandten Hemisphären der Kugeln erzeugt wurden, sodass eine Sichtbarkeit nicht von vornherein ausschließbar ist. Für die Verteilung wurden 2 Verfahren untersucht. Zum einen die Hammersley-Methode<sup>18</sup> zur Erzeugung von zufälligen Vektoren in einer Hemisphäre und eine iterative Methode, bei der das Skalar Produkt des Zufallsvektor mit dem Richtungsvektor als Referenz genommen wurde.

Da bei der Generierung der Testdaten die Geschwindigkeit zweitrangig war, wurde die zweite, eigene Methode für die Erstellung der Punkte genutzt.

An Abbildung 27 kann erkannt werden, dass eine stärker ausgeglichene Sichtbarkeitsverteilung das Resultat dieser Punkteverteilung ist.<sup>19</sup>



**Abbildung 27** Sichtbarkeitstest Verteilung Sichtbarkeit; Szene-ID 3 & Zufällige Verteilung auf zueinander zu gewendeten Kugeloberflächen

Tabelle 6 zeigt die Verteilung der Kugel im Raum in Szene-ID 3 und den Radius der jeweiligen Kugel; bei Szene-ID 1 & 2 wurden jeweils die erste bzw. ersten 3 Kugel verwendet.

**Tabelle 6** Kugel Verteilung Szene-ID 3 (Szene-ID 1 & 2 sind äquivalent)

Center (x,y,z)			Radius
-8.51253	0.592608	-5.28855	1.00116
9.7058	-0.978851	-2.79946	1.27024
7.1514	-1.2855	8.68465	1.31892
-0.282296	0.507035	-9.82849	1.03485
-1.03977	-1.63414	6.04358	1.27299
5.81225	0.643513	9.46531	1.64016
-8.63277	1.52562	-9.19248	1.56328
-9.63988	-1.70141	9.70153	1.29679
2.96243	0.0639057	1.34495	1.02564
-9.47081	-0.479446	-2.29896	1.72057

<sup>18</sup> Sampling with Hammersley and Halton Points  
<http://www.cse.cuhk.edu.hk/~ttwong/papers/udpoint/udpoint.pdf>

<sup>19</sup> Da sich die weiteren Verteilungen (A.xyz, B.xyz, d.xyz, I) kaum von denen in Abbildung 26 unterschieden, wurde auf eine zusätzliche Abbildung dieser verzichtet.

### 3.3. Feature Engineering

Bei der Feature Konstruktion und Optimierung (engl. Feature Engineering) werden beim maschinellen Lernen oft Gleitkommawerte in bestimmten Wertebereichen zusammengefasst und somit diskretisiert. Dies kann den Lernaufwand des Modells minimieren und den Einfluss von Ausreisern minimieren. Dies wird auch als Binning bezeichnet. (Kapitel 8 Representation; Cleaning Data [9])

Zum Beispiel könnte man (in anderen Datensätzen) geografische Koordinaten zu Gebieten anhand des Längen- und Breitengrades zusammenfassen.

Im Rahmen des Raytracings ist jedoch die geometrische Genauigkeit insbesondere bei Schnittests von ausschlaggebender Bedeutung. Eine kleine Abweichung bezüglich des Floating Point Wertes kann entscheidende Auswirkungen haben. Insbesondere aus diesem Grund nutzen viele Raytracer *double* Präzision anstelle von *floating point* Präzision<sup>20</sup> und nehmen dadurch einen höheren Speicherverbrauch und eine längere Renderdauer in Kauf.

Auch weitere Methoden, wie das Skalieren der Felder des Feature Vektors oder das Clamping (Abschneiden) wurden aus oben genannten Bedenken vermieden. Beim Skalieren von Features wird in der Regel der Bereich kleiner gemacht, was zu einer geringeren Präzision führen würde. Im Gegensatz dazu würde beim Skalieren mit einem Faktor  $> 1$  zwar keine Präzision verloren gehen, solange die Werte im Wertebereich des *float* Wertebereichs bleiben, jedoch können die Modelle des maschinellen Lernens besser mit kleineren Wertebereichen ( $>0$ ) arbeiten. (vgl. [9]) Dadurch würde diese Veränderung der Daten sich voraussichtlich auch kontraproduktiv auswirken. Beim Clamping geht ebenfalls, wie der Begriff bereits erahnen lässt, die Präzision der Daten verloren.

Insbesondere werden die vorgestellten Methoden des Feature Engineerings genutzt um Daten, die eine größere Streuung besitzen, oder Abweichungen von den generellen Werten zu behandelnd. Bei diesem Verfahren sollen insbesondere Ausreißer oder fehlerhafte Daten einen kleineren (bis keinen) Einfluss auf das Training des Modells besitzen. Dadurch, dass die hier genutzten Daten auf geometrisch und numerisch exakten Berechnungen basieren, wird nicht von fehlerhaften Daten und großen Ausreisern ausgegangen (bis auf die beschriebenen kleinen Sampling Artefakte). Die beschriebenen Methoden könnten jedoch bei einem

---

<sup>20</sup> Nach dem IEEE Standard [14] umfasst ein float Wert 32 Bit und ein double 64 Bit, somit die doppelte Größe und daher eine höhere Präzision.

zufälligen Sampling von Strahlen im Raum die Verteilung dieser verbessern.

Neben den hier vorgestellten Feature Vektoren gibt es noch viele andere Möglichkeiten die Daten zu verarbeiten und Varianten für die Feature Vektoren zu erstellen.

Letztendlich ist beim maschinellen Lernen das Feature Engineering (z.B. auch die Erstellung synthetischer Features) und die Auswertung der Trainingsdaten einer der wichtigsten Schritte und notwendig, um Modelle für komplexere Probleme zu erstellen. Dieser Schritt nimmt in der Regel auch am meisten Zeit in Anspruch.

Für das Training mit den ermittelten Daten wurden folgende Feature Vektoren entworfen. Es ist zu bedenken, dass die Erstellung synthetischer Features auch bei späterer Evaluierung und der Anwendung des Modells durchgeführt werden müssen.

### **3.3.1. Datensatz für Szenengeometrie Schnitttests**

Als Label für das Training und die Validierung wurde jeweils der „hit“-Wert aus den Raw-Daten verwendet. Der binäre Wert gibt an, ob der Strahl die Szenen Geometrie schneidet. Bei den Klassifizierungsmodellen, wird der hit-Wert verwendet, um daraus kategorische Label zu erstellen. Wie in der Einleitung dieses Abschnitts beschrieben steht der Wert 0 dafür, dass der Strahl die Szenengeometrie nicht trifft und wird als negative Kategorie (0) gewertet. Die Kategorie 1 wird als positiv gewertet und beschreibt, dass der Strahl die Szenengeometrie trifft. In den Raw-Daten ist dieser mit dem hit-Wert 1 gekennzeichnet.

#### **Feature Vektor #1**

[*origin.x, origin.y, origin.z, direction.x, direction.y, direction.z*]

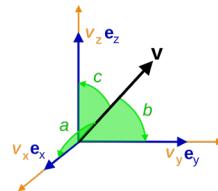
Der Feature Vektor besteht aus den identischen Daten, die aus den exportieren Raw-Daten entnommen wurden. Er besitzt eine Größe von 6 Floating-Werten. Die ersten drei Werte sind der Ursprung des Strahls die folgenden drei Parameter beziehen sich auf die Richtung des Strahls.

#### **Feature Vektor #2 (mit synthetischen Features)**

[*origin.x, origin.y, origin.z, direction.x, direction.y, direction.z, angle.x, angle.y, angle.z*]

Eine zweite Variante beinhaltet, zusätzlich zu den Felder des Feature Vektors #1, 3 weitere Felder. Bei diesen Feldern handelt es sich um synthetische generierte Richtungskosinus [11] des Richtungsvektors. In Abbildung 28 stellt der Vektor  $\vec{v}$  den

ermittelten (euklidischen) Richtungsvektor in  $\mathbb{R}^3$  dar und  $\vec{e}_x, \vec{e}_y, \vec{e}_z$  die Standardbasisvektoren.



**Abbildung 28** Vektor  $v$  im  $\mathbb{R}^3$   
Bildquelle: Direction cosine, en.Wikipedia.org

$$\alpha = \cos a ; a = \frac{\vec{v} \cdot \vec{e}_x}{|\vec{v}| |\vec{e}_x|} = \frac{v_x}{|\vec{v}|} = \frac{v_x}{\sqrt{v_x^2 + v_y^2 + v_z^2}}$$

**Formel 3.1** Berechnung des Richtungskosinus zwischen dem Richtungsvektor ( $\vec{v}$ ) und Standardbasisvektor  $\vec{e}_x$

Formel 3.1 zeigt die Berechnung des Richtungskosinus basierend auf dem Kosinussatz zwischen einem Vektor  $\vec{v}$  und dem Standardbasisvektor  $\vec{e}_x = (1,0,0)$ , der die Welt X-Achse der Szene beschreibt. Die Berechnungen der Richtungskosinus  $\beta$  und  $\gamma$  ist äquivalent mit den entsprechenden Basisvektoren und wird hier nicht explizit notiert. Da es sich bei den Richtungsvektoren um Einheitsvektoren handelt, ist der Divisor bei der Formel 3.1 immer 1, so dass eine rechenintensive Berechnung der Quadratwurzel entfällt. In Folge dessen muss lediglich auf die einzelnen Komponenten des Richtungsvektors die Arkuskosinus Funktion (Umkehrfunktion zum Kosinus) angewendet werden, um den Winkel zu erhalten.

Die Erstellung dieser synthetischen Features kann vor dem Modelltraining für jeden Strahl in dem Datensatz mit der oben genannten Formel für alle drei Standardbasisvektoren durchgeführt werden.

Die berechneten Richtungswinkel sind in Radiant nach der Anwendung der Arkuskosinus Funktion. Eine Konvertierung in Grad wurde für die Modelltrainingsversuche als nicht notwendig erachtet.

### Feature Vektor #3 (mit Feature Kreuzungen)

[*origin.x, origin.y, origin.z, crossOrigin* [= *cross(origin.x, origin.y, origin.z)*],  
*direction.x, direction.y, direction.z, crossDirection* [= *cross(direction.x, direction.y, direction.z)*],  
*crossOrigDir* [= *cross(crossOrigin, crossDirection)*] ]

Ein alternativer Ansatz ist es, die semantischen zusammengehörenden Daten mittels Feature Kreuzung (engl. Feature Crossing) zu kombinieren. Insbesondere findet Feature Kreuzung dann statt, wenn es sich um nicht lineare Probleme handelt, die mit dem Modell gelöst werden sollen. Dabei werden zwei oder mehr Features miteinander multipliziert und es entsteht ein neues Feature, welches beim Training wie die anderen Features ein Gewicht besitzt und erlernt wird.

Um den Zusammenhang der 3 Werte des Ursprungs im  $\mathbb{R}^3$  zu verdeutlichen, wird eine Feature Kreuzung (*crossOrigin*) zwischen den Werten *origin.x*, *origin.y* und *origin.z* erstellt. Dasselbe gilt für den Richtungsvektor, für den eine Feature Kreuzung (*crossDirection*) zwischen den Werten *direction.x*, *direction.y* und *direction.z* erstellt wird. Um den geometrischen Zusammenhang zwischen dem Ursprung und dem ausgehenden Richtungsvektor zu verdeutlichen, wird eine weiterer Feature Kreuzung (*crossOrigDir*) zwischen den neu erstellen Feature Kreuzungen (*crossOrigin* & *crossDirection*) erstellt.

Eine solche Kreuzung der Vektoren wird bei Neuronalen Netzen, die „Fully-connected“ sind, automatisch vorgenommen durch die Verbindung der Neuronen.

### 3.3.2. Datensatz für Sichtbarkeitstests

Bei diesem Datensatz wurde das Feature Engineering implizit durchgeführt. Für die entsprechend zu untersuchenden Faktoren wurden explizit Datensätze erstellt, die z.B. Bestimmte Verteilungen besitzen (vgl. Absatz 3.2.3). Zusätzlich kann die Erstellung des Richtungsvektor zwischen den Punkten und die Berechnung des Abstandes als Feature Engineering angesehen werden.

Auf die Erstellung von Vektoren, die die Richtungscosinus beschreiben, wurde für diesen Datensatz, aufgrund der Ergebnisse, die bei dem Datensatz für die Szenenschnitttests ermittelt wurden, verzichtet.

#### Feature Vektor #1

[*A.x, A.y, A.z, B.x, B.y, B.z, direction.x, direction.y, direction.z, length*]

Der Feature Vektor besteht aus den identischen Daten, die aus den exportierten Raw-Daten entnommen wurden. Er besitzt eine Größe von 10 Floating-Werten, die ersten drei Werte sind der Ursprung Punkt, die weiteren drei Werte beschreiben den Endpunkt des Sichtstrahls. Die folgenden drei Parameter beschreiben den Richtungsvektor von Punkt A zu Punkt B. Der letzte Parameter beschreibt den Abstand von Punkt A und Punkt B, somit die Länge des Sichtstrahls.

# 4.

## Hardware

Sämtliche Training- und Testläufe wurden auf einem Computer durchgeführt, welcher folgende Spezifikationen aufweist.

**Tabelle 7** Verwendete Computer Hardware

CPU	AMD Ryzen Threadripper 2950X 16-Core Processor 3,50Ghz
GPU	NVIDIA GeForce RTX 2080 Ti Dedizierter GPU-Speicher: 11GB
Arbeitsspeicher	64 GB
Sekundär Speicher	Samsung SSD 970 Pro 512GB Samsung SSD 860 EVO 1TB
Betriebssystem	Windows 10 – Pro 64 Bit

Wie im 2. Abschnitt beschrieben, wurde die GPU Variante von dem Tensorflow Framework verwendet. Während sämtlicher Versuche konnten folgende Beobachtungen gemacht werden.

Bei jedem Training wurden lediglich 12% der GPU Kapazität genutzt. Die meiste Zeit befand sich die GPU im Copy-Modul. Bei genauerer Analyse kann erkannt werden, dass der dedizierte GPU-Speicher eine maximale Auslastung hat und sich hier scheinbar das Bootleneck befindet. Dies scheint laut Internet Recherchen ein öfter auftretendes Phänomen zu sein, wenn auf der GPU diese Art von Trainingsdaten (nicht Bilddaten) verarbeitet werden.

Die folgende Abbildung zeigt die einzelnen Auslastungen der Grafikkartenkomponenten. Es ist klar die starke Auslastung des dedizierten GPU

Speichers zu erkennen. Im direkten Vergleich konnte jedoch festgestellt werden, dass das Training auf der GPU in der obengenannten Systemkonfiguration trotz dieses Bottlenecks schneller ist als auf der CPU.



**Abbildung 29** GPU Auslastung beim Modell-Training

Bei der Durchführung des Modelltrainings konnte erkannt werden, dass sich parallele Arbeiten an dem Computer, wie zu erwarten, negativ auf das Modelltraining auswirkten, sodass nach Möglichkeit während der Trainingsphase keine parallelen Berechnungen auf dem Computersystem durchgeführt wurden.

# 5.

## AI Modelle, Training & Evaluierung

Dieses Kapitel beschreibt die verschiedenen Modelle, die trainiert wurden, in diversen Testdurchläufen. Insbesondere ist auf die verschieden genutzten Parameter zu achten, die verwendet wurden.

In allen Trainings-/Testdurchläufen wurden die in Abschnitt 3 genannten Parameter (Ursprungspunkt und Richtung) für jeden Strahl als Features für das Modell verwendet. Der binäre Wert Hit wurde als Label für das Training und die nachfolgende Evaluierung verwendet. Als Erinnerung für Abschnitt 5.1: ein mit 0 gelabelter Datensatz bedeutet, dass der Strahl die Geometrie nicht trifft; ein mit 1 gelabelter Datensatz bedeutet, dass der Strahl die Szenen-Geometrie schneidet. Ein Großteil der Modelle wurde nach dem Training exportiert, um für mögliche weitere Test zur Verfügung zu stehen. Die Modell-ID hilft, die Modelle den Testdurchläufen zuzuordnen und ist in den kommenden Unterabschnitt bei den einzelnen Tests notiert.

Die Beschreibungen der einzelnen Durchläufe der Modelle ist hier nach Kategorie der Modelle sortiert. In der Forschungsarbeit selber wurde jedoch nach Reihenfolge der Modell-IDs vorgegangen. Insbesondere wurde zwischen linearen Klassifiziermodell und neuralem Netzwerk öfters gewechselt, um diese beiden Modelltypen besser vergleichen zu können.

Es ist ebenfalls zu beachten, dass das gesamte Training in Perioden unterteilt wurde, so dass nach jeder Periode der Fortschritt des Modelltrainings analysiert werden kann. Das aktuelle Modell wird mit dem Trainings- und Test-Set evaluiert und die Ergebnisse werden für das Plotten der Konvergenzkurven zwischengespeichert. Durch diesen Monitoring-Schritt verlängerte sich das Modelltraining um ca. den doppelten bis dreifachen Zeitfaktor.

## 5.1. Datensatz für Szenengeometrie-Schnitttests

### 5.1.1. Linearer Regressor

Die ersten Testversuche wurden mit einem linearen Regressor durchgeführt.

Ein linearer Regressor liefert einen floating Point-Wert zurück, der jedoch keine direkte Aussage über mögliche Schnitte mit der Geometrie aussagt. Durch manuelles Testen ergaben sich Werte außerhalb des erwarteten Wertebereiches [0,1].

```
1. import tensorflow as tf
2.
3. # erstellen eines linearen Regressors
4. estimator = tf.estimator.LinearRegressor(feature_columns,
optimizer=opt)
```

**Listing 5.1** Tensorflow Linearer Regressors Pseudocode

Es wurden verschiedene Tests mit dem linearen Regressionsmodell durchgeführt.

Diese Testreihe wurde hauptsächlich durchgeführt, um ein Setup bzw. eine Pipeline für die Tensorflow API zu erstellen und die Datenverarbeitung für komplexere Modelle vorzubereiten. Unter anderem lag ein Hauptaugenmerk in diesen Durchläufen darin, die Daten so zu formatieren, dass diese in einem Tensorflow Modell Training genutzt werden können.

Es wurden verschiedene Größen von Trainings-Sets und Validation-Sets gewählt, um unter anderem die Auswirkungen der Größen auf das Training zu beobachten. Gewählt wurde der für lineare Regression typische Gradient Descent Optimizer, um den Verlust über das Training zu verringern und der RMS-Error für die Auswertung herangezogen. Als Output des Modells werden, wie beschrieben, ungeclampte floating Point-Werte erwartet.

Als Feature Vektor wurde Vektor #1 (vgl. Abschnitt 0) gewählt, der kein Feature Crossing oder synthetische Features aufweist.

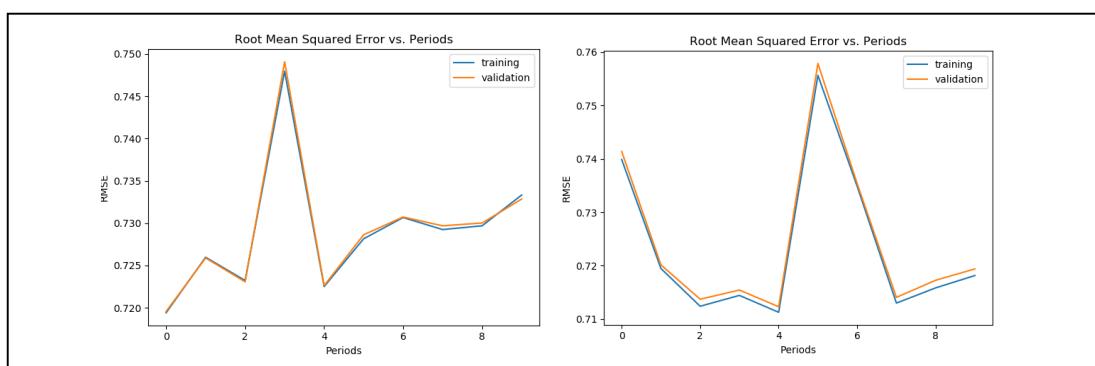
In Abbildung 30 sind die Konvergenzkurven von zwei ausgewählten Trainingsdurchläufen mit einem zugrundeliegenden Regressions-Modell geplottet. Der Hauptunterschied zwischen den Modellen liegt zum einen an der Größe der verwendeten Datensätze und zum anderen an einer Permutation dieser, die bei Modell-ID 2 ergänzt wurde.

Für Modell-ID 1 wurde eine Anzahl von 400.000 Trainings-Daten und 100.000 Validations-Daten gewählt. Bei Modell-ID 2 wurde eine Trainings-Set aus 671.088 Daten (Vergrößerung um den Faktor 1.6), ein Validation-Set mit 167.772 Daten sowie zusätzlich ein Test-Set mit 209.715 Daten erstellt. Vor der Teilung des Trainings-Set

und dem Validations-Set wurde bei Modell-ID 2 eine Permutation der Datensamples durchgeführt, um eine bessere Verteilung innerhalb dieser Teilmengen zu gewährleisten.

**Tabelle 8** Modell-Parameter Linearer Regressor Modell ID 1 & 2

Periods	Learning Rate	Steps	Batch Size
10	0,0001	500	10



**Abbildung 30** Linearer Regressor Konvergenz-Kurve, Links ID 1, Rechts ID 2

Die Tabelle 8 zeigt die weiteren verwendeten Modellparameter der beiden Durchläufe.

Das Training (incl. Monitoring) des Modells-ID 1 dauerte 1 Stunde und 10 min, bei Modell-ID 2 wurde eine Dauer von 3 Stunden gemessen, was auf die Vergrößerung des Datensatzes zurückzuführen ist.

Bei dem Modell-ID 1 kann eine recht zufällige Trainingsentwicklung beobachtet werden. Von einem anfänglichen RMS-Error von 0.72 kann nach 10 Trainings-Perioden eine Verschlechterung auf 0.73 erkannt werden und ein maximaler Verlust in Periode 3 mit einem RMSE-Wert von 0.75.

Insgesamt kann fast über die gesamte Trainingsdauer eine steigende Konvergenzkurve erkannt werden und es sind Schwankungen zwischen der Trainings- und Validations-Kurve zu beobachten.

Aufgrund der negativen Entwicklung wurde auf eine zusätzliche Evaluierung des Modells bei diesem Durchlauf verzichtet. Es konnte bei dem Durchlauf jedoch die Infrastruktur für die weiteren Modelltrainings erstellt werden und insbesondere die

definierte Input-Funktion für Tensorflow kann in weiteren Durchläufen in ähnlicher Form wiederverwendet werden.

Da es sich bei dem Modell um einen ersten Prototyp handelte, wurde dieses Modell in diesem Durchlauf nicht exportiert.

Bei Modell-ID 2 ist eine Verbesserung des Modells innerhalb der ersten 4 Perioden des Trainings zu erkennen. Der Peak zwischen der 4. und 7. Periode ist wieder ein Indiz dafür, dass ein lineares Regressionsmodell an dieser Stelle scheinbar willkürlich die Gewichte ändert und dies zu einem Fall um knapp 0.05 beim RMSE führt. Es kann auch bedeuten, dass an diesem Punkt das Modell konvergiert ist, da die Kurve zwischen Periode 7 und 9 wieder zu steigen beginnt.

Jedoch bleibt im Gegensatz zu den Fluktuationen bei Modell-ID 1 die Kurve des Validations-Sets wie zu erwarten kontinuierlich etwas über der Trainings-Set-Kurve. Das Modell wurde nach dem Training mit dem Testdatensatz evaluiert und ein finaler Root Mean Squared Error von 0.73 konnte ermittelt werden. Dieser ist geringfügig schlechter als der RMSE, der in der letzten Periode des Trainings an dem Trainings-Set und Validations-Set ermittelt werden konnte.

Anhand der beschriebenen Testfälle konnte erkannt werden, dass sich ein lineares Regressionsmodell nicht optimal für die Lösung dieses Problems eignet. Die Ausgabe des Modells kann bei diesem nicht eindeutig gedeutet werden, da der Wertebereich, wie in der Einleitung beschrieben, an einigen Stellen stark von der binären Einstufung, ob ein Strahl die Geometrie schneidet oder nicht abweicht. Weitere Versuche, das Problem mit einem linearen Regressor (z.B. mit anderen Parametern) zu lösen, wurden nicht unternommen.

### 5.1.2. Lineares Klassifizierungsmodell

```
1. import tensorflow as tf
2.
3. # erstellen eines linearen Klassifizierers
4. estimator = tf.estimator.LinearClassifier(feature_columns,
optimizer=opt)
```

**Listing 5.2** Tensorflow Lineares Klassifizier-Modell Pseudocode

Da es sich bei dem vorliegenden Problem generell um ein Klassifizierungsproblem handelt, ist ein lineares Klassifizierungsmodell der zweite Ansatz, der für den Lösungsansatz gewählt wurde. Hierbei wird das Label als Kategorie betrachtet. Es gibt demnach die Kategorie 0, die für ein Nichttreffen der Szenengeometrie steht und die Kategorie 1, die dafür steht, dass die Szenengeometrie von dem Strahl getroffen wird. Das Modell liefert eine Wahrscheinlichkeitsprognose der Zugehörigkeit der Stichprobe (des Strahls) zu jeder der beiden Kategorien. Die Summe beider Prozentwerte summiert sich auf 1 auf. Wie in Absatz 2.4 beschrieben, wird für lineare Klassifizierungsmodelle als Verlustfunktion Log Loss verwendet. Dies bedeutet auch, dass die Plots nicht direkt mit den Konvergenzkurven aus den Testdurchläufen des linearen Regressors verglichen werden können. Im Folgenden sind die Trainingsdurchläufe abhängig von den veränderten Parametern sortiert und es wird dessen Auswirkung analysiert.

In sämtlichen Trainingsdurchläufen wurde der Datensatz zu 80 % in Trainings- und Validations-Daten geteilt und zu 20% in Test-Daten.

Innerhalb der ersten Teilung wurde eine Permutation durchgeführt, bevor die Teilmenge erneut zu 80% in Trainingsdaten und 20% Validationsdaten aufgeteilt wurde. Daraus resultieren die in Tabelle 9 dargestellten absoluten Mengen-Größen.

**Tabelle 9** Datensatzteilung; Szenengeometrieschnitttests, Absolute Mengen Größen

Training Size	Validation Size	Test Size	Datensatz-Größe
671,088	167,772	209,715	1,048,575

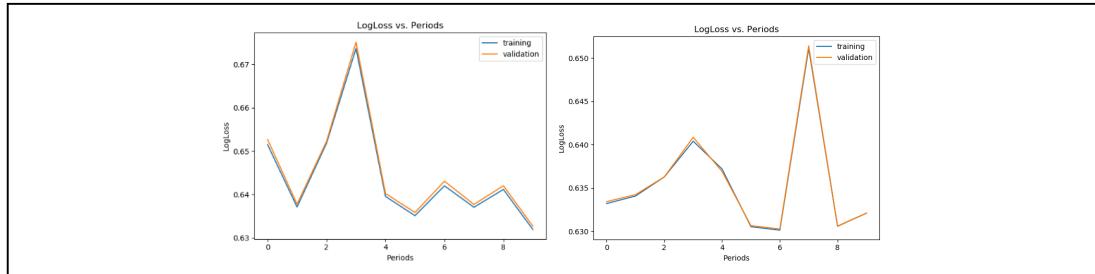
## Gradient Descent Optimizer

Aufbauend auf den Trainingsdurchläufen in Absatz 5.1.1 wurde der Gradient Descent Optimierer zur Minimierung des Verlustes beim Training evaluiert. Tabelle 10 liefert eine Übersicht über die weiteren genutzten Modellparameter.

**Tabelle 10** Modell-Parameter Linear Klassifiziermodell ID 3, 4 & 6

Periods	Learning Rate	Steps	Batch Size
10	0,0001	500	10

Die Trainingsdauer bei dieser verwendeten Parameter- und Inputdatenkombination betrug zwischen 2 Stunden und 30 Minuten bei Testdurchlauf 3 und max. 3 Stunden bei Test 5 und 6.



**Abbildung 31** Lineares Klassifiziermodell Konvergenz-Kurve; Links ID 3, Rechts ID 6

Abbildung 31 zeigt, wie sich das Modell bei Test 3 mit seinen Prognosen über das Training entwickelt. Es ist zu sehen, dass das Modell etwas schlechter beim Validations-Set ist als beim Trainings-Set, was jedoch erwartet wurde. Für die Evaluierung wurde (wie bei der Verlust Funktion des Modells) der Log Loss verwendet. Daher ist hier, wie in der Einleitung erwähnt, ein neuer Wertebereich auf der Y-Achse des Plots, entstanden, was einen Vergleich mit dem Linearen Regressionsmodell erschwert. Zwar kann sich das Modell über die 10 Perioden von einem Wert von 0.65 auf knapp 0.63 verbessern, jedoch sind die Schwankungen und ebenfalls der Peak in Periode 3 zu beachten. Eine kontinuierliche Verbesserung des Modells, welche an einer gleichmäßig sinkenden Konvergenzkurve zu erkennen wäre, ist aus diesem Testlauf nicht ersichtbar.

Zusätzlich wurde nach abgeschlossenem Training die Genauigkeit (engl. accuracy) sowie die Fläche unter der Grenzwertoptimierungskurve (AUC) berechnet. Für die folgenden Testfälle kann die Genauigkeit als Qualitätsmaß für das Modell angesehen werden. Die Genauigkeit nach dem Modell Training konnte als 0.64 bestimmt werden. Dementsprechend liegt das Modell mit 64% der Prognosen richtig. Dies ist jedoch nur marginal besser als ein Algorithmus, der zufällig die Strahlen klassifiziert und im Schnitt 50% richtige Ergebnisse liefert. Für die Fläche unter der ROC Kurve (AUC) konnte der Wert 0.68 ermittelt werden.

Sowohl das Modell-ID 3 sowie Modell-ID 6 weisen jeweils einen negativen Peak in verschiedenen Phase des Trainings auf, was wahrscheinlich an der unterschiedlichen Permutation der Trainingsdaten liegt. Es konnte jedoch festgestellt werden, dass sich die Modelle, abgesehen von dem Peak, nur bedingt unterscheiden.

Hier ist eine Stärke des linearen Klassifizierungsmodells festzustellen. Bei ähnlichen Parametern erlernt sich lineare Modell in einer ähnlichen Weise bei individuellen Trainings und sind nicht so stark abhängig von den zufälligen Initialwerten. Insbesondere im Vergleich zu neuronalen Netzwerken ergibt sich hier ein Vorteil dieser Modelle. Eine genauere Analyse und Diskussion ist in Abschnitt 6 zu finden.

### **AdamOptimizer**

Ein weiterer Ansatz ist die Anwendung des AdamOptimizer als Optimierungsalgorithmus.

Beim Adam (adaptive moment estimation) Algorithmus handelt es sich um eine Abwandlung des stochastischen Gradient Descent Algorithmus, der laut der Autoren effizient ist, wenig Speicherverbrauch hat und insbesondere für Probleme mit großen Datenmengen vorgesehen ist [12].

Während des Trainings wird die Learning Rate für jedes Gewicht des Modells individuell angepasst.

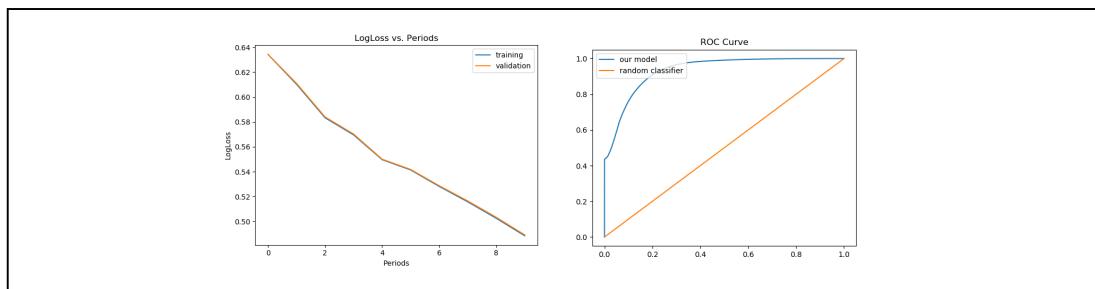
Des Weiteren wurden die Batch Size in diesem Durchlauf auf 40 erhöht, so dass mehr Werte in den Optimierungsschritt einbezogen werden. Die Learning Rate wurde initial auf 0,001 gesetzt, was einer Erhöhung um den Faktor 10 gegenüber den bisherigen Durchläufen ist. Dadurch soll das Modell schneller konvergieren.

Die Schrittzahl wurde auf 800 erhöht, was generell die Trainingszeit verlängert. (Siehe Tabelle 11)

**Tabelle 11** Modell-Parameter lineares Klassifiziermodell ID 8

Periods	Learning Rate	Steps	Batch Size
10	0,001	800	40

Durch die veränderten Parameter verlängerte sich die Trainings Zeit des Modells auf 3 Stunden und 30 Minuten.



**Abbildung 32** Lineares Klassifiziermodell ID 8; Links Konvergenz-Kurve, Rechts ROC Kurve

Das Modell lieferte nach abgeschlossenem Training einen AUC-Wert von 0.93 und einen Accuracy-Wert von 0.80 beim Validations-Set. Die ROC des Validations-Set ist in Abbildung 32 (rechts) geplottet. Beim Test-Set konnte ebenfalls ein AUC-Wert von 0.93 ermittelt werden, der Accuracy-Wert ist minimal schlechter und beträgt 0.79, was zu erwarten war.

Dieser Durchlauf mit dem Adam Algorithmus zur Optimierung konnte sehr positive Werte liefern. An der Konvergenzkurve auf der linken Seite in Abbildung 32 kann ein erwartetes Trainingsverhalten des Modells erkannt werden. Es konnte eine kontinuierlich sinkende Konvergenzkurve über die gesamten Trainingsperioden ermittelt werden. Über 10 Perioden konnte der Log Loss von 0.64 auf 0.50 gesenkt werden. Eine Korrektheit der Prognosen von ca. 80% wurde bei dem Validations-Set sowie dem Test-Set ermittelt.

Der Ursprung der starken Verbesserung der Modellergebnisse zu den vorherigen Durchläufen wird durch die Änderung des Optimierungs-Algorithmus angenommen.

### Learning Rate & Stepssize

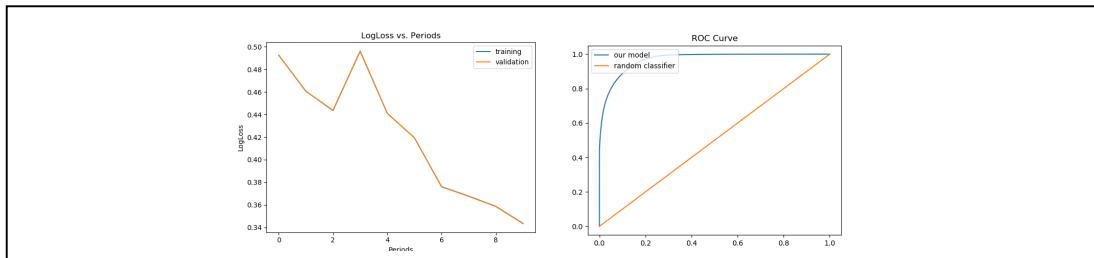
Aufbauend auf die sehr positiven Ergebnisse bei Modell-ID 8 wurde in diesem Durchlauf weiterhin der AdamOptimizier genutzt der im Hinblick auf die Validierung

der Thesis eine starke Verbesserung gegenüber dem generellen Gradient Descent Optimizer gebracht hat. Hauptsächlich wurden in diesem Durchlauf minimal die Learning Rate und die Schrittanzahl des Optimier Algorithmus angepasst. Alle weiteren Parameter wurden wie in Tabelle 12 ersichtlich gegenüber dem vorherigen Durchlauf beibehalten.

**Tabelle 12** Modell-Parameter lineares Klassifiziermodell ID 9

Periods	Learning Rate	Steps	Batch Size
10	0,002	900	40

Es wurde eine Verlängerung der Trainingsdauer gemessen, die sich auf 4 Stunden belief. Dies ist hauptsächlich auf die Erhöhung der Schrittzahl um 100 zurückzuführen.



**Abbildung 33** Lineares Klassifiziermodell ID 9; Links Konvergenz-Kurve, Rechts ROC Kurve

Beim Validations-Set konnte nach abgeschlossenem Training des Modells ein AUC-Wert von 0.97 und eine Accuracy von 0.87 ermittelt werden. Die selben Werten wurden bei dem Test-Set ermittelt werden.

Im Vergleich zum vorhergehenden Test verbessert sich Genauigkeit um weitere 7% zum vorhergehenden Modell. In Abbildung 33 sieht man, dass die ROC Kurve stärker steigt und sich der AUC-Wert weiter dem Wert 1.0 annähert. Insbesondere kann dieser Kurve entnommen werden, dass knapp 70% der negativen Klassifizierungen korrekt sind. Die Konvergenzkurve sinkt stetig im Verlauf des Trainings und abgesehen von dem Peak in Periode 3 ist festzustellen, dass sich die Kurve des Trainings-Sets kaum von der des Validations-Sets unterscheidet. Das ist dadurch zu erklären, dass eine generell starke Ähnlichkeit zwischen allen Strahlen besteht, von denen 64% Prozent für das Training verwendet wurden (siehe Tabelle 12).

Es ist zu bemerken, dass das Training dieses Modells bereits bei einem Log Loss-

Wert von knapp 0.50 startet. Dieser Wert ist der selbe, den das Model des vorhergehenden Durchlaufs in der letzten Periode erreicht. Hier wird angenommen das dieses Modell auf dem vorherigen Modell aufbaut und dessen Werte initial übernimmt, da sich die beiden Modelle in demselben temporären Speicherbereich befanden. Dafür spricht auch, dass das Modell mit ID 8 bis zur letzten Periode eine stetig sinkende Konvergenzkurve aufweist. Diese Hypothese erklärt auch den Peak in Periode 3 des Modells ID 9, der dadurch entsteht, dass die Historie des Modells 8 nicht übernommen wurde. Mit knapp 88% Trefferquote konnte dieses Modell als bestes Modell der Testreihe bei linearen Klassifizierungsmodellen bewertet werden.

### Feature Vektor – Verwendung der Raumwinkel

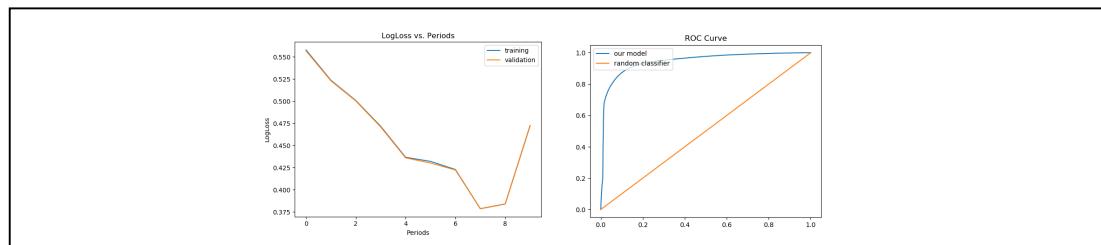
Für das Modell-ID 13 wurde der Feature Vektor #2 (vgl. Abschnitt 3.3.1) verwendet. Dieser besitzt als synthetisches Feature die Winkel zu den Standardbasisvektoren in Radian.

Um einen Vergleich mit dem vorherigen Modellen zu bekommen, wurden die weiteren Modellparameter (vgl. Tabelle 13) der vorherigen Durchläufe beibehalten.

**Tabelle 13** Modell-Parameter lineares Klassifiziermodell ID 13

Periods	Learning Rate	Steps	Batch Size
10	0,002	900	40

Das Modelltraining dauerte wie bei den Vorgängern um die 4 Stunden und 30 Minuten, es ist kein sehr großer Anstieg der Trainingsdauer durch die Vergrößerung der Input-Feature-Komplexität zu erkennen.



**Abbildung 34** Linear Klassifiziermodell ID 13; Links Konvergenz-Kurve, Rechts ROC Kurve

An Abbildung 34 ist zu erkennen, dass das Modell beim Training eine kontinuierliche Verbesserung über die 10 Trainingsperioden erfahren kann.

Das Optimum ist bei Periode 7, von welchem Punkt die Konvergenzkurven wieder zu steigen beginnen. Im Vergleich zu den Trainingsdurchläufen mit dem Feature-Vektor #1 erkennt man, dass sich bei den Konvergenzkurven eine minimale Verbesserung innerhalb der ersten Perioden des Trainings bis zu dem Optimum (logloss ~0.38) in Periode 7 des Modells ergibt.

Die finale Evaluierung mit dem Test-Set und dem Validierungs-Set ergab folgende Werte: Der AUC-Wert bei beiden Sets betrug 0.95, die Genauigkeit dieser betrug beim Validations-Set ein Wert von 0.73 und beim Test-Set einen etwas besseren Wert von 0.75. An der ROC-Kurve in Abbildung 34 ist zusätzlich zu erkennen, dass die Kategorie „Zuweisungsgenauigkeit“ des Modells (insbesondere bei der positiven Kategorie) nicht so abrupt ist wie bei Modell-ID 8 (vgl. Abbildung 32).

Es ist zu beachten, dass die Evaluierung auf dem Zustand des Modells nach der letzten Trainingsperiode beruht. Daher lieferte das Modell voraussichtlich ein schlechteres Ergebnis, als wäre es mit dem Zustand beim Optimum in Periode 7 validiert worden.

Es kann anhand dieser Beobachtung allgemein festgestellt werden, dass das synthetische Feature, welches auf den Richtungscosinus basiert, zu keiner gravierenden Verbesserung des Modells beim Training führte.

### 5.1.3. Neuronal Network

In diesem Abschnitt wird die Testreihe mit dem Deep-Neural-Network Klassifizierungsmodell an dem Datensatz (Abschnitt 3.1) zur Lösung des vorliegenden Problems beschrieben. Der folgende Pseudo-Code Ausschnitt zeigt, wie mithilfe von der Tensorflow-API ein Deep-Neural Network Klassifizierungsmodell erstellt werden kann. Im Vergleich zu den bisherigen linearen Modellen wird an diesen ein Array (hidden\_units) übergeben. Die Größe des Arrays gibt die Anzahl der Layer im neuronalen Netz an, die jeweiligen Werte innerhalb des Arrays geben jeweils die Anzahl der Neuronen in jedem Layer an.

```
1. import tensorflow as tf
2.
3. # erstellen eines DNNClassifier regressor Objektes
4. # hidden_units gibt die Anzahl der Layer und die jeweils
5. # enthaltenen Neuronen an
6. estimator = tf.estimator.
    DNNClassifier(feature_columns,hidden_units=[512,512,24])
```

**Listing 5.3** Tensorflow Deep Neural Network Klassifizier-Modell Pseudocode

Bei der Verwendung von neuronalen Netzen ist generell davon auszugehen, dass das Training länger dauert als bei linearen Modellen. Das ist auf die Komplexität dieser Modelle zurückzuführen.

Für das vorliegende Problem wurde ein „Fully-Connected“-Ansatz gewählt, da alternative Ansätze laut Literatur eher für Bildanalysen genutzt werden. Außerdem wird bei diesen neuronalen Netzen aufgrund der Verbindung implizit Feature Crossing erstellt. Zusätzlich kann sich die zufällige Verteilung der initialen Gewichte stark auf das Netzwerk auswirken, so dass trotz gleicher Parameter und Input-Daten je nach Durchlauf verschiedene Ergebnisse herauskommen können. Aus diesem Grund wurden die Testreihen mehrfach mit gleichen Parametern ausgeführt. Es wird jedoch nur eine Auswahl an Resultaten vorgestellt.

## **Optimierungsalgorithmus & Netzwerkstruktur**

Wie in den ersten Tests der linearen Modelle wurde auch bei dem neuronalen Netze zuerst die Optimierungsalgorithmen verglichen.

Es wurde wieder der Gradient Descent Optimizer (Modell-ID 5) verwendet außerdem wurde der FTRL Optimizer (Modell-ID 7) und der Adam Optimizer (Modell-ID 9) verwendet.

Bei den Durchgänge kamen folgende Netzwerkstrukturen zur Anwendung:

### **Gradient Descent Optimizer:**

Es wurden 4 Layer erstellt. Das 1. Layer besitzt 1024 Neuronen, im 2. & 3. Layer wurden jeweils 512 und im 4. Layer 256 definiert. Das hidden\_units Array sieht dementsprechend folgendermaßen aus: [1024, 512, 512, 256].

### **FTRL Optimizer:**

Auch beim FTRL Optimizer wurden äquivalent zum Gradient Descent Optimizer 4 Layer erstellt. Das hidden\_units Array sieht dementsprechend folgendermaßen aus: [1024, 512, 512, 256].

Aufgrund der Komplexität wurde für dieses Netzwerk zusätzlich L1 Regulierung mit einer Regulierungsstärke von 0.1 gewählt

### **Adam Optimizer:**

Es wurden 5 Layer erstellt; die ersten 2 Layer besitzen jeweils 30 Neuronen. Das 3. & 4. Layer besitzt jeweils 20 Neuronen und das letzte Layer 10 Neuronen. Das hidden\_units Array sieht dementsprechend folgendermaßen aus: [30, 30, 20, 20, 10].

Es wurde eine in allen Durchgängen sehr hohe Schrittzahlen von 1000 und eine große Batch Size von 40 gewählt.

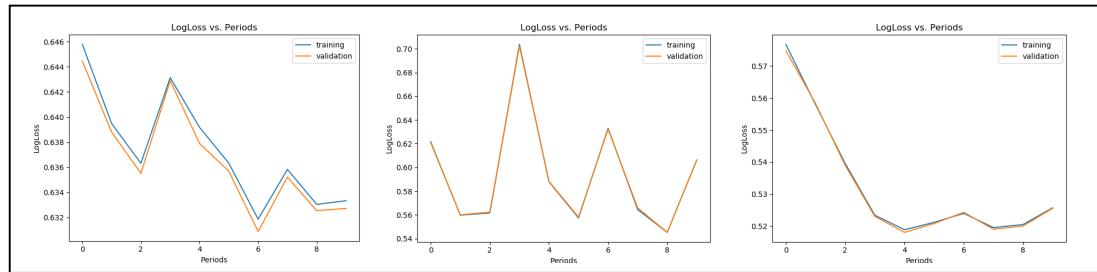
Tabelle 14 fasst die verwendeten Parameter zusammen.

**Tabelle 14** Parameter Deep Neural Network Klassifizier-Modell ID 5, 7 &10

Periods	Learning Rate	Steps	Batch Size
10	0,001 (Bei Modell-ID 10: 0,002)	1000 (Bei Modell-ID 10: 900)	40

Die Trainingsdauern betragen in den jeweiligen Modellen: Gradient Descent Optimizer: 5 Stunden und 30 Minuten, FTRL Optimizer: 8 Stunden und beim Adam Optimizer: 5 Stunden

Wie in der Einleitung dieses Absatzes beschrieben wurde, ist eine längere Trainingszeit bei neuronalen Netzen zu erwarten. Zusätzlich kann aus Tabelle 14 entnommen werden, dass eine sehr hohe Optimierungsschrittzahl für das Training gewählt wurde.



**Abbildung 35** Konvergenzkurven Deep Neural Network Klassifizierungs-Modell ID 5, 7 &10

Links Modell-ID 5 (Gradient Descent Optimizer), Mitte Modell-ID 7 (FTRL Optimizer), Rechts Modell-ID 10 (Adam Optimizer)

Nach den Modell-Trainings konnten die in Tabelle 15 gezeigten Werte ermittelt werden.

**Tabelle 15** Auswertung Deep Nerual Network Klassifizierungs-Modell ID 5, 7 &10

	Validation-Set		Test-Set	
	Accuracy	AUC-Wert	Accuracy	AUC-Wert
Gradient Descent Optimizer (Modell-ID 5)	0,59	0,75	0,58	0,75
FTRL Optimizer (Modell-ID 7)	0,54	0,5	-	-
Adam Optimizer (Modell-ID 9)	0,7	0,74	0,69	0,74

Im direkten Vergleich konnte der Adam Optimizer die besten Ergebnisse erreichen. Es ist doch zu beachten, dass dieses Modell eine kleinere Netzstruktur als die anderen beiden und kleinere Abweichungen bei den weiteren Trainingsparametern besitzt (vgl. Tabelle 14). Die Parameter für den Gradient Descent Optimizer und den Adam Optimizer wurden abhängig von den vorherigen Ergebnissen der linearen Modelle gewählt.

Bei allen Modellen ist zu erkennen, dass lediglich ein minimaler Unterschied zwischen den evaluierten Validations- und Testsets vorliegt. Dies lässt sich dadurch erklären, dass insbesondere bei den Modellen mit der ID 5 und 7, ein sehr geringer finaler Genauigkeitswert vorliegt und dieser nur minimal besser ist, als der eines zufälligen Klassifizierungsmodells, welches im Schnitt einen Genauigkeitswert von 0.5 aufweist. Bei steigender Genauigkeit kann an dem Modell 9 erkannt werden, dass sich diese Werte leicht unterscheiden.

Die Konvergenzkurve des Modells mit der ID 5 in Abbildung 35 zeigt trotz Fluktuationen einen Lernfortschritt des Modells über die 10 Perioden, bei dem eine geringe Verbesserung des Log Loss von 0.645 auf 0.635 beobachtet werden kann.

Beim Trainingsverlauf des Modells mit der ID 7 erkennt man einen sehr unruhigen Verlauf. Es sind Unterschiede des Log Loss-Wertes von knapp 0.15 zu sehen. Insbesondere zwischen der 2. und 3. Trainingsperiode steigt der Log Loss extrem an. Diese extremen Schwankungen sind auf den verwendeten Optimierungsalgorithmus des Modells zurückzuführen. Der FTRL-Algorithmus ist speziell für, sich ändernde Daten, wie z.B. User-Meinungen/Befragungen vorgesehen.

Über das Training passt dieser das Modell neuen Daten an und es werden z.T. erlernte Werte eher verworfen [13]. Dieser Optimierungsalgorithmus ist hauptsächlich

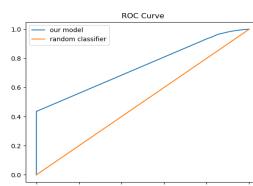
für online Modelle (Modelle die kontinuierlich lernen) vorgesehen. Da es sich bei dem vorliegenden Problem um ein geometrisches Problem handelt und sich die Gesetzmäßigkeiten für dieses nicht (wie z.B. Ansichten/Meinungen) über verschiedene Perioden verändern, ist der hierfür angewendete Optimierungsalgorithmus nicht optimal. In Hinblick auf den schlecht möglichen AUC-Wert von 0,5, der erreicht werden kann, erweist sich das Modell als weniger geeignet.

Aufgrund der geringen Erfolgsquote des Modells wurde auf eine Evaluierung dieses anhand eines Test-Sets verzichtet.

Bei der Nutzung des Adam Optimiziers kann eine klare Verbesserungen gegenüber den anderen Optimierungsalgorithmen beobachtet werden.

Die ermittelten Genauigkeits-Werte des Modells liegen weit über einem Random Klassifizieren und den vorhergehenden Neuronalen Netzwerke.

An der Konvergenzkurve des Trainings kann bis zur 4. Periode eine kontinuierliche Verbesserung erkannt werden. Ab Periode 4 wird keine weitere Verbesserung erzielt. Für dieses Modell wurde eine ROC Kurve geplottet, die in Abbildung 36 zu sehen ist. Die Abbildung zeigt, dass ca. 40% der positiven Klassifizierungen korrekt identifiziert werden und lediglich knapp 10% der negativen Klassifizierungen identifiziert das Modell korrekt.



**Abbildung 36** Deep Neural Network Klassifizier-Modell ID 10 ROC Kurve

In Anbetracht der Trainingsdauer und der ernüchternden erreichten Ergebnisse der Modelle ID 5 und 7 wurde auf eine Wiederholung der Trainings mit diesen Optimieralgorithmen verzichtet. Unter Umständen kann das Ergebnis auf suboptimale Initialparameter zurückzuführen sein. Die Optimieralgorithmen wurden zusätzlich mit einer Netzwerkgröße gleich der des Modells ID 10 evaluiert, aber konnten keine starke Verbesserung vorweisen. Es konnte bereits beim der linearen Klassifizierung erkannt werden, dass der Adam Optimizier eine größere Verbesserung der Trainingsergebnisse erreichen konnte. Dies wird durch diese Versuchsreihe auch bei den neuronalen Netzwerken bestätigt.

## Neuronale Netzwerk Komplexität

Da sich die Komplexität des Modells mit der ID 10 im vorhergehenden Versuch wesentlich von den beiden anderen Modellen unterscheidet, wurden Modelle mit dem Adam Optimizer mit verschiedenen Netzwerkgrößen in diesem Abschnitt evaluiert. Die weiteren Parameter wurden beibehalten und sind zur besseren Übersicht in Tabelle 16 erneut gelistet.

**Tabelle 16** Parameter Deep Neural Network Klassifizier-Modell ID 11, 12, 14 und 15

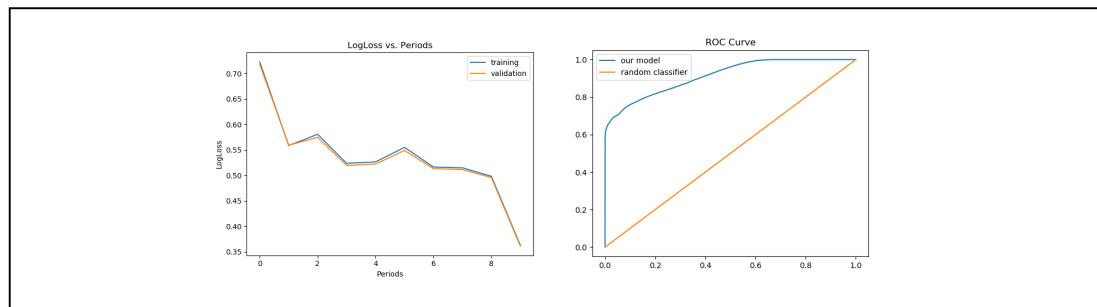
Periods	Learning Rate	Steps	Batch Size
10	0,002	900	40

Bei Modell-ID 11 wurde, im Vergleich zum vorherigen Modell, die Anzahl der Neuronen in jedem der 5 Layer jeweils um den Faktor 10 erhöht. Daher wird folgendes Array für die `hidden_units` verwendet: [300, 300, 200, 200, 100].

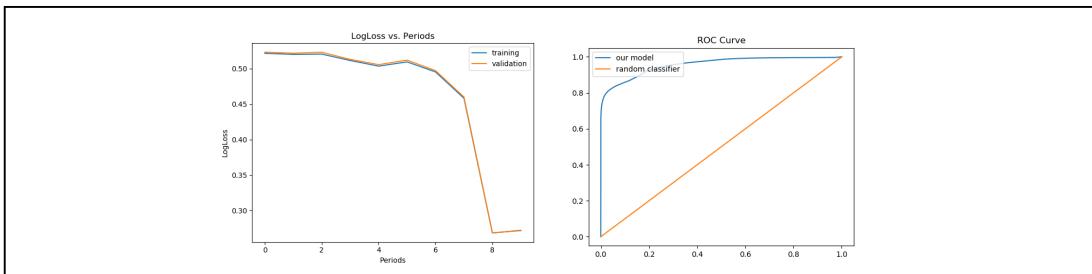
Bei Modell-ID 12 wurde die Anzahl in den 5 Layern nochmal jeweils um den Faktor 2 erhöht. Dies führt zu folgendem `hidden_units` Array: [600, 600, 400, 400, 200].

Durch die Vergrößerung der Netzwerkstruktur erhöht sich die Dauer des Trainings und eine Gesamtdauer von 6 Stunden wurde für das Modell-ID 11 gemessen.

Das Training des Modells mit der ID 12 verlängerte sich nochmals und führte zu einer Gesamtdauer von ca. 6 Stunden und 30 Minuten. In beiden Fällen ist diese Verlängerung auf die gestiegene Komplexität der Netzwerke zurückzuführen sein.



**Abbildung 37** Deep Neural Network Classifier Modell-ID 11; Links Konvergenz-Kurve, Rechts ROC Kurve



**Abbildung 38** Deep Neural Network Klassifizier-Modell ID 12; Links Konvergenz-Kurve, Rechts ROC Kurve

Bei Modell-ID 11 wird nach abgeschlossenem Training mit dem Validations-Set ein AUC-Wert von 0.91 und eine Genauigkeit von 0.82 erreicht. Beim Test-Set kann ein AUC-Wert von 0.94 und eine Genauigkeit von 0.85 ermittelt werden.

Modell-ID 12 liefert eine weitere Verbesserung und es kann für das Validations-Set ein AUC-Wert von 0.96 und eine Genauigkeit von 0.87 ermittelt werden. Für das Test-Set konnte ein AUC-Wert von 0.95 und ein Genauigkeits-Wert von 0.86 verzeichnet werden.

Die Konvergenzkurve in Abbildung 37 zeigt, dass das Modell-ID 11, abgesehen von einigen kleinen Schwankungen, eine kontinuierliche Verbesserung erreicht und sich der Log Loss über die 10 Perioden von 0.70 auf knapp 0.35 verbessern kann. Insbesondere können 2 größere Sprünge in diesem Training beobachtet werden, die sich zwischen der 0. und 1. Periode sowie von der 8. zur 9. Periode befinden.

An der ROC Kurve in derselben Abbildung stellt man fest, dass knapp 60% der positiven Klassifizierungen und knapp 40% der negativen Klassifizierungen korrekt sind.

Abbildung 38 zeigt die Konvergenzkurve des Modells mit der ID 12 über die 10 Trainingsperioden. Insbesondere kann ein großer Sprung des Log Loss von der 7. zur 8. Periode beobachtet werden, innerhalb der der Log Loss von ca. 0.45 zu knapp 0.29 sinkt.

Dies könnte auf die Zufälligkeit von neuronalen Netzen zurück zu führen sein, es ist jedoch auffällig, dass diese Verbesserung erst in der 7. Periode Auftritt. Auf der anderen Seite kann jedoch zwischen der 5. und 7. Periode bereits ein geringes Absinken des Log Loss beobachten werden, welches auf diese starke Verbesserung hindeuten kann.

An der ROC-Curve kann erkannt werden, dass das Modell wieder die positiven Kategorien besser klassifiziert als die negativen. Knapp 80% korrekte positive

Klassifizierung und ca. 65-70% korrekte negative Klassifizierung, können aus dem ROC-Plot entnommen werden.

Aus diesen Versuchen kann geschlossen werden, dass sich eine Vergrößerung der Netzwerk-Struktur positiv auf das Training auswirkt.

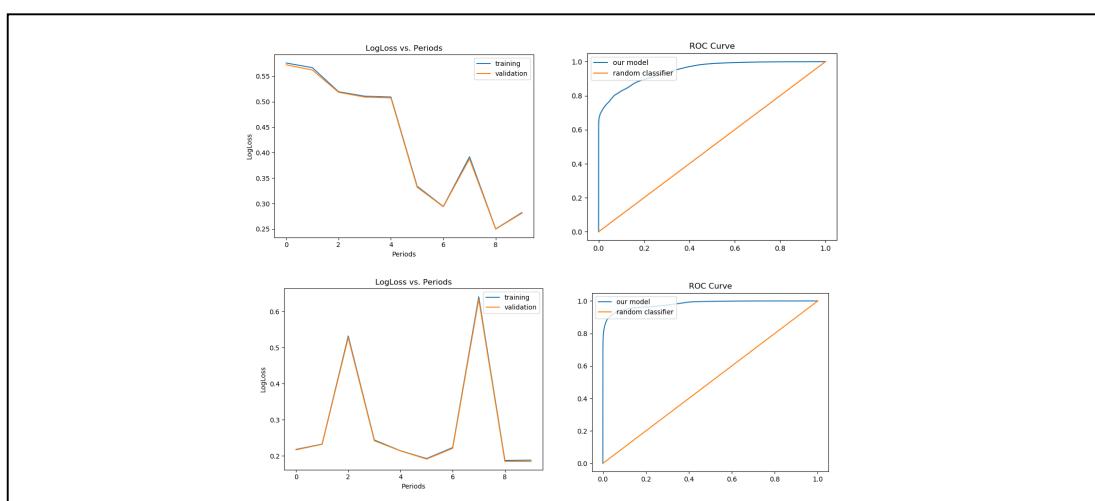
### Verlängerung Trainingsdauer & Feature Vektor

Die Modelle ID 14 und 15 sind aufeinander aufbauend und unterscheiden sich von den Vorgängern daran, dass der Modell ID 15 mit erlernten Werten vom Modell ID 14 weiter lernt. Anders gesagt wurde das Training mit doppelter Länge durchgeführt.

Es wurde weiterhin der AdamOptimizer gewählt und die Parameter wie in Tabelle 16 beibehalten. (hidden\_units: [600, 600, 400, 400, 200])

Als weitere Variation wurde für diesen Durchlauf der Feature Vektor #2 (vgl. 3.2.1) gewählt und dadurch der Richtungscosinus mit beachtet.

Diese Testreihe sollte insbesondere durch die zwei aufeinander aufbauenden Trainings eine maximale Genauigkeit unter Vernachlässigung der Zeitkomponente erforschen.



**Abbildung 39** Deep Neural Network Klassifizier-Modell ID 14 und 15 Konvergenzkurven & ROC-Kurve

Konvergenzkurven (Oben Links ID 14 oben links, ID 15 unten links), ROC-Kurve (ID 14 oben rechts, ID 15 unten rechts)

Bei den neuronalen Netzen wirkte sich das Feature Engineering stärker auf die Dauer des Modelltrainings aus, so dass für das Training des Modells ID 14 ca. 8 Stunden

Trainingszeit benötigt wurden und für Modell-ID 15 weitere 9 Stunden. Es wurde eine Gesamtdauer von 17 Stunden Trainingszeit der beiden Tests zusammen gemessen.

Für Modell-ID 14 kann ein finaler Genauigkeitswert von 0.86 ermittelt werden und ein AUC-Wert von 0.95 bei dem Training- sowie dem Validations-Set.

Die Evaluierung des Modells mit der ID 15 liefert eine Verbesserung des Genauigkeits-Wertes und es kann ein absoluter Wert 0.92 ermittelt werden. Der AUC-Wert ist auf 0.98 gestiegen.

An der Abbildung 39 lässt sich erkennen, dass sich während des Trainings, das Modell-ID 14 fast kontinuierlich verbessern kann. Die einzige Ausnahme ist die Periode 7, in der sich das Modell im Vergleich zur vorherigen Periode verschlechtert. An der ROC-Kurve dieses Trainings erkennt man, dass das trainierte Modell tendenziell wieder eine stärkere Sicherheit bezüglich der positiven Kategorie besitzt.

Im aufbauenden Training mit Modell-ID 15 kann kaum eine größere weitere Verbesserung an den Konvergenzkurven erkannt werden. Insbesondere deuten die enormen Schwankungen darauf hin, dass das Training stärker durch Zufallsfaktoren beeinflusst wurde und es eher auf Try-And-Error aufbaut als einer strukturellen Verbesserung folgt. Dies entspricht stark dem Charakter von neuronalen Netzen im Vergleich zu linearen Klassifizierungsmodellen.

Im direkten Vergleich zu vorherigen Trainingsdurchläufen mit neuronalen Netzwerk Modellen (mit identischen Trainingsparametern) konnte keine erhebliche durch die Nutzung der synthetisch generierten Richtungskosinus Verbesserung erkannt werden.

Viel entscheidender war in dieser Versuchsreihe die doppelte Trainingsdauer, die zu dem in der Forschungsarbeit besten Wert von 0.92 führte. Zu beachten ist, dass insgesamt knapp 17 Stunden Training erforderlich waren, dieses Resultat zu erreichen.

Da die 3 weiteren Felder im Feature Vektor anhand der Ergebnisse keinen weiteren positiven Einfluss auf das Modelltraining hatten, ist anzunehmen, dass das Modell mit der ID 12 ein ähnliches finales Ergebnis bei doppeltem Durchlauf liefern würde bei einer geschätzten Trainingszeit von ca. 13 Stunden. Die Zeitdifferenz ist an dieser Stelle auf die unterschiedliche Feature-Vektor-Komplexität und damit verbundene Speichergröße zurückzuführen.

Es wurden weitere Test durchgeführt, in denen weitere Parameter der Netzwerke verändert wurden, wie die Learning Rate, Schrittgröße und Batch Size. Weitere erwähnenswerten Beobachtungen konnten jedoch nicht gemacht werden.

## 5.2. Datensatz für Sichtbarkeitstests

Die Testreihe baut auf den Ergebnissen aus Abschnitt 5.1 auf. Um die Modelle von denen der vorherigen Versuchsreihe zu unterscheiden, wurde als Präfix „2\_Nr“ ergänzt.

Da die Daten ähnliche geometrische Probleme enthalten, wie in den vorherigen Trainings, wird vermutet, dass sich das Training in ähnlicher Weise wie in dem vorherigen Abschnitt entwickelt. Anzumerken ist, dass der Input komplexer ist. Es wurde bei diesen beschriebenen Durchläufen weniger Wert auf die Vorstellung verschiedener Modell-Parameter Wert gelegt, vielmehr wird die Auswirkung unterschiedlicher Inhalte der Inputdaten, wie in Abschnitt 3.2 beschrieben, analysiert. In der nachfolgenden Tabelle sind die absoluten Größen der in Abschnitt 3.2.3 beschriebenen Datenteilungen nochmals aufgezeigt, die bei den folgenden Trainingsdurchläufen verwendet wurden.

**Tabelle 17** Datensatzteilung; Sichtbarkeitstests, Absolute Mengen Größen

Training Size	Validation Size	Test Size	Datensatz-Größe
960000	240000	300000	15,000,000

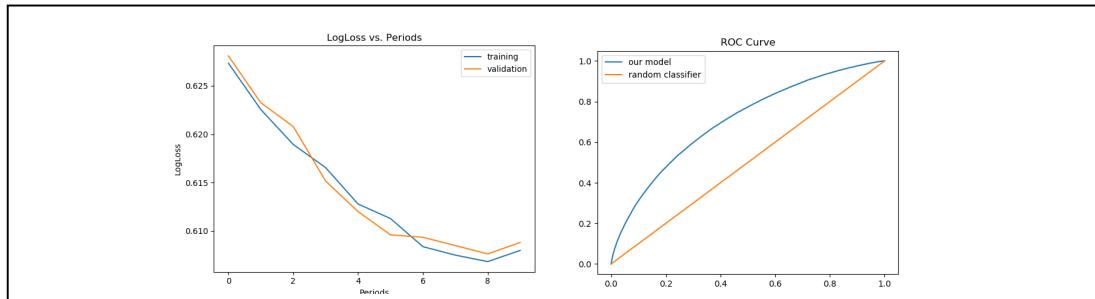
### 5.2.1. Lineares Klassifizierungsmodell

Das Modell ID 02\_01 wurde anhand der exportierten Sichtbarkeitstestdaten aus Szene-ID 4 trainiert. In dieser Szene befanden sich 300 zufällig verteilte Kugeln im Raum mit unterschiedlichem Radius um den Wert 1. Abbildung 24 und Abbildung 25 in Abschnitt 3.2.3 zeigen die Verteilung der Inputparameter in diesem Datensatz.

**Tabelle 18** Parameter Modell-Training für Modell ID 02\_01 & 02\_02

Periods	Learning Rate	Steps	Batch Size
10	0,001	900	40

Das Training mit den genannten Parametern dauerte 4 Stunden. Abbildung 40 zeigt die Trainingsentwicklung des Modells über die 10 Trainingsperioden.



**Abbildung 40** Lineares Klassifizierungsmodell ID 02\_01; Links Konvergenz-Kurve, Rechts ROC-Kurve

Es kann eine kontinuierliche Verbesserung des Modells beobachtet werden. Das Training fängt jedoch ab Periode 6 zu konvergieren und es wird wenig weitere Verbesserung erreicht.

Ein finaler Genauigkeitswert von 0.65 und ein AUC-Wert von 0.68 wird ermittelt. Festzustellen ist, dass bei den ähnlichen Trainings-Parametern dieses Problem komplexer zu sein scheint, als das der Schnitttests in vorherigen Versuchen.

Insbesondere ist die Schwankung der Training-Set-Kurve im Vergleich zur Validation Set-Kurve verwunderlich. Es wird vermutet, dass dies daran liegt, dass die hohe Genauigkeit Auswirkungen hat, wie die Verteilungen in den einzelnen Datenteilmengen ausgewertet werden.

An der AUC-Kurve kann erkannt werden, dass eine hohe Fehlerquote bei der Klassifizierung besteht, die jedoch gleichverteilt auf beide Label ist.

Da die Kurve bereits ab Periode 6 anfängt, zu konvergieren und weniger negative Steigung besitzt, ist davon auszugehen, dass auch eine Verlängerung des Trainings keine weiteren positiven Wirkungen haben wird.

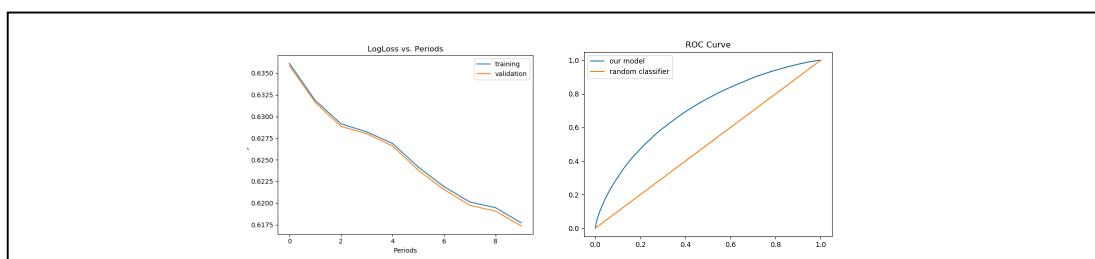
### 5.2.2. Neuronales Netzwerk

Im Vergleich zu dem linearen Klassifizierungsmodell in Abschnitt 0 wurde mit gleichen Daten ein Neuronales Netz trainiert.

Es wurde ausgehend von Abschnitt 5.1.3 eine Netzwerkstruktur gewählt, die zwei Layer mit jeweils 600, zwei Layer mit jeweils 400 und ein Layer mit 200 Neuronen besitzt (`hidden_units=[600, 600, 400, 400, 200]`).

Das Training dieses Netzes dauerte ca. 8 Stunden, was auf die Komplexität zurückzuführen ist. Es handelt sich bei diesem wie auch bei diesen neuronalen

Netzwerken um „Fully-Connected“ Netzwerke und entsprechend viele Gewichte. Abbildung 41 zeigt die Entwicklung des Modells über die Trainingsdauer.



**Abbildung 41** Lineares Klassifizierungsmodell ID 02\_03; Links Konvergenz-Kurve, Rechts ROC-Kurve

Bei der Evaluierung des Modells nach abgeschlossenem Training kann für das Validations-Set ein Genauigkeitswert von 0.66 und ein AUC-Wert von 0.7 ermittelt werden. Dasselbe Ergebnis konnte auch beim Test-Set festgestellt werden.

Es ist über die gesamte Trainingsdauer eine kontinuierliche Verbesserung an der stetig sinkenden Kurve zu erkennen, die auch zu Trainingsende noch nicht konvergiert ist und in folgenden Trainingsperioden voraussichtlich weiter sinken würde.

Anhand der ROC-Kurve ist zu erkennen, dass die Klassifizierung jedoch kaum besser ist als bei Modell ID 02\_01 mit dem linearen Klassifizierungsmodell nach 10 Trainingsperioden.

### Auswirkungen Inputdaten auf Modelltraining - Szenenkomplexität

In den folgenden Trainingsdurchläufen soll bewiesen werden, dass die Modelle eine bessere Genauigkeit bei geringerer Szenenkomplexität erreichen können und eine Abhängigkeit zwischen Szenenkomplexität und Trainingserfolg besteht. Es wurden die Szenen-IDs 1 bis 3 evaluiert, die eine ansteigende Anzahl an Kugeln besitzen (vgl. Abschnitt 3.2.3).

Um das Training zu beschleunigen, wurde zum einen die Learning Rate und die Batch Size vergrößert wie in Tabelle 21 erkennbar ist.

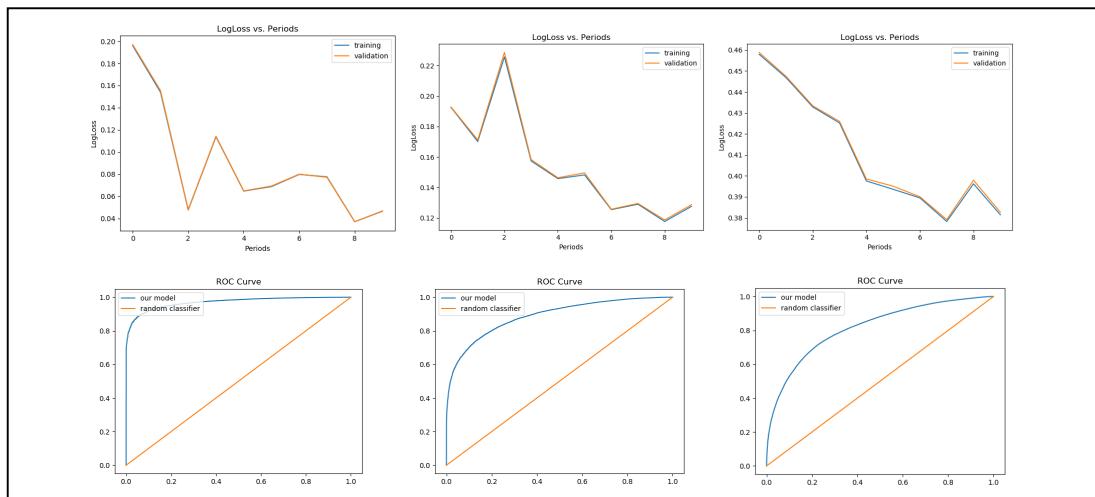
**Tabelle 19** Parameter Modell-Training für Modelle ID 02\_03 - 02\_07

Periods	Learning Rate	Steps	Batch Size
10	0,01	900	60

Zusätzlich wurde eine geringere Netzwerkkomplexität gewählt, die durch den folgenden Vektor beschrieben ist: hidden\_units=[600, 600, 400, 400, 200].

Aufgrund der veränderten Parameter konnten die Modelle jeweils innerhalb von 3 bis 4 Stunden trainiert und ausgewertet werden.

Abbildung 42 zeigt den Trainingsverlauf der 3 Modelle und die finale Auswertung dieser.



**Abbildung 42** Deep Neural Network Klassifizierungs-Modell ID 02\_03 - 02\_05; Oben Konvergenz-Kurven, Unten ROC-Kurven

Links Modell-ID 02\_03 (Szene ID 1; 1ne Kugel), Mitte Modell-ID 02\_04 (Szene ID 2; 3 Kugeln), Rechts Modell-ID 02\_05 (Szene ID 3; 10 Kugeln)

Es konnten folgende (in Tabelle 20 abgebildete) Werte nach dem Training ermittelt werden.

**Tabelle 20** Auswertung Deep Nerual Network Klassifizierungs-Modell für ID 02\_03 - 02\_05

	Validation-Set		Test-Set	
	Accuracy	AUC-Wert	Accuracy	AUC-Wert
1ne Kugel in der Szene ID 1 (Modell-ID 02_03)	0,99	0,88	0,99	0,89
3 Kugeln in der Szene ID 2 (Modell-ID 02_04)	0,96	0,88	0,96	0,88
10 Kugeln in der Szene ID 3 (Modell-ID 02_05)	0,83	0,82	0,83	0,82

Klar erkennbar ist, dass mit steigender Anzahl an Szenengeometrie die Genauigkeit

des Modells sinkt. In Anbetracht der Verteilungen, die in Abschnitt 3.2.3 ermittelt wurden, sind auch die vorliegenden Modelle mit Genauigkeitswerten  $> 0.9$  nicht sonderlich erfolgreich. Beachtet man nämlich, dass bei der Label-Verteilung eine ähnlich hoher Prozentwert bzgl. einer Kategorie vorliegt, erreicht ein Modell, welche z.B. immer eine Kategorie prognostiziert ein ähnlich hohes Ergebnis.

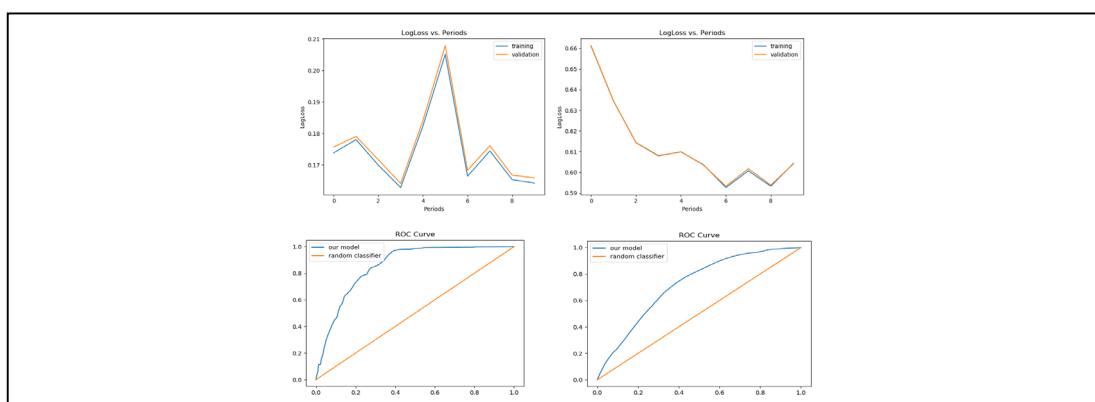
Bei einer Anzahl von 10 Kugeln nähert sich das Resultat insbesondere die ROC-Kurve stärker dem Ergebnis des Modells mit der ID 02\_02 an, welches mit 300 Kugeln trainiert wurde.

Aus den Ergebnissen kann erkannt werden, dass die Modelle zwar bessere Genauigkeits- und AUC-Werte liefern aber in Anbetracht der beschriebenen Problematik kein entscheidend besseres Endresultat liefern können.

### **Auswirkungen Inputdaten auf Modelltraining – Oberflächenpunktverteilungen**

Wie in Abschnitt 3.2.3 analysiert, sind die Punkte der vorherigen Inputdaten in den vorhergehenden Trainingsdurchläufen zufällig im Raum verteilt und repräsentieren nicht die Strahlen, die in einem normalen Raytracing Prozess untersucht werden.

Aus diesem Grund wurden die zwei weiteren Datensätze erstellt, die diesen Faktor berücksichtigen sollen. Sämtliche weiteren Parameter wurden zum besseren Vergleich wie in den vorherigen Versuchen belassen.



**Abbildung 43** Deep Neural Network Klassifizierungs-Modell ID 02\_06 & 02\_07; Oben Konvergenz-Kurven, Unten ROC-Kurven

Links Modell-ID 02\_06 (Zufällige Verteilung auf Kugeloberflächen), Rechts Modell-ID 02\_07 (Szene ID 3; Zufällige Verteilung auf Kugeloberfläche unter Beachtung der potentiellen Sichtbarkeit)

Für den Datensatz mit der zufälligen Punktverteilung auf den Kugeloberflächen (Model-ID 02\_06) konnte ein finaler Genauigkeitswert von 0.95 und ein AUC-Wert von 0.85 festgestellt werden sowohl bei dem Validations- als auch bei dem Test-Set.

Für den Datensatz mit der zufälligen Punktverteilung auf den Kugeloberflächen unter Berücksichtigung der potentiellen Sichtbarkeit (Model-ID 02\_07) konnte ein finaler Genauigkeitswert von 0.7 und ein AUC-Wert von 0.72 festgestellt werden beiden Sets.

An dem Verteilungsplot des Datensatzes bei Modell ID 02\_06 konnte erkannt werden, dass die positiven Sichtbarkeitslabel stark unterrepräsentiert sind was sich scheinbar wie in den vorherigen Modellen, auf das Modelltraining auswirkt. Resultierend daraus ist ein hoher Genauigkeitswert aber eine geringerer AUC-Wert. Insbesondere an der ROC-Kurve kann in Abbildung 43 eine starke Unsicherheit dieser Klassifizierungen erkannt werden.

Bei dem Datensatz, der bei Modell ID 02\_07 verwendet wurde, ist eine weitgehend ausgeglichene Label-Verteilung gegeben, wie bei dem Datensatz der Szene ID 4 in den ersten Testreihen. Es kann auch hier trotz der geringeren Anzahl der Kugeln keine stark positive Tendenz im Training erkannt werden und das Modell kann ab Periode 5 kaum weitere Verbesserungen erreichen.

Dass sich das Modell ID 02\_07 kaum von den ersten Versuchen (Modell ID 02\_02) unterscheidet, liegt daran, dass zwar eine geringere Anzahl an Kugel vorliegt aber die Verteilung der Punkte im Raum stärker durch die Kugeloberfläche eingeschränkt ist, wodurch sich eine ähnliche relative Punkt-Geometrie Verteilung ergibt.

### **Keras Framework Vergleichsnetzwerk**

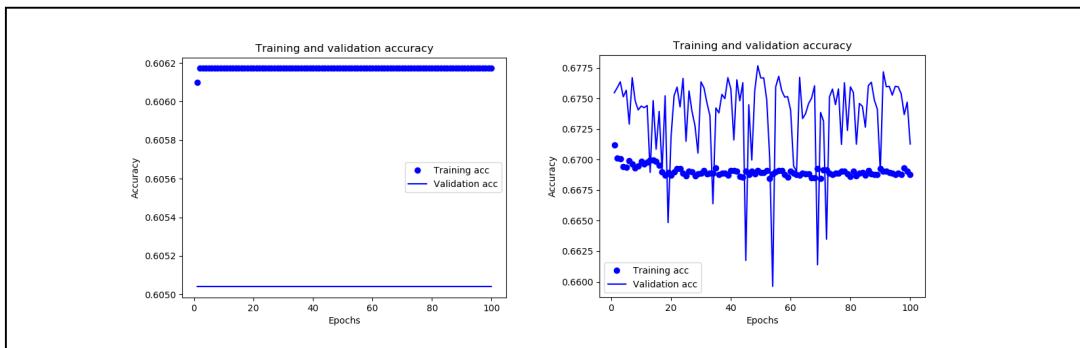
Alle Modelle und Trainings wurden auf einer Struktur durchgeführt, die im Rahmen dieser Arbeit programmiert wurde. Um mögliche Fehler bei der Programmierung und dem Training auszuschließen, wurde eine Referenzimplementierung mit dem Keras Framework erstellt und verschiedene Durchläufe mit variierenden Parametern durchgeführt.

Es wurde als Grundlage die folgende Keras Netzwerk-Struktur genutzt (Listing 5.4). Zum Vergleich wurde ebenfalls ein „Fully-Connected“ Netzwerk ohne Dropout genutzt.

Es wurden ebenfalls 3-4 Layer getestet, was jedoch zu einem enormen Zeitaufwand führte.

Layer (type)	Output Shape	Param #
<hr/>		
dense_1 (Dense, RELU)	(None, 200)	2200
dropout_1 (Dropout)	(None, 200)	0
dense_2 (Dense, RELU)	(None, 125)	25125
dropout_2 (Dropout)	(None, 125)	0
dense_3 (Dense, Sigmoid)	(None, 1)	126
<hr/>		

**Listing 5.4 Keras Modell Struktur**



**Abbildung 44** Auswertung Keras Referenzimplementierung; Links 2 Layer, Rechts 4 Layer

Durch die in Abbildung 44 gezeigten Ergebnisse, die stark schwankende Werten ohne eine erkennbare Verbesserung über das Training zeigen, kann darauf geschlossen werden, dass die für die Arbeit implementierten Modelle bessere bzw. äquivalente Ergebnisse liefern könnten.

Das Training wurde mit dem Datensatz, der aus Szene ID 4 exportiert wurde, durchgeführt. Es können maximale Werte von 0.67 bzw. 0.68 erreicht werden, die ebenfalls von Modell 02\_02 erreicht werden konnten und nur minimal besser sind als die des linearen Klassifizierungsmodells mit der ID 02\_01.

# 6.

## Geschwindigkeit, Genauigkeit, Auswertung

Dieser Abschnitt beschreibt und vergleicht die Genauigkeit und Geschwindigkeiten der Modelle beziehungsweise der Testreihen in Abschnitt 5.

Von den vorgestellten Testreihen<sup>21</sup> der Modelle konnte bei der Schnitttest Untersuchung das neuronale Netzwerk Modell ID 15 in Abschnitt 5.1.3 am besten abschneiden. Ein finales Genauigkeits-Ergebnis von 0.92 bezüglich der richtig zugeordneten positiven Kategorien konnte jedoch lediglich durch einen Zeitaufwand von 20 Stunden Trainingszeit erreicht werden.

Bei den Sichtbarkeitstests konnte eine maximale 99%ige Genauigkeit erreicht werden, wobei angemerkt werden muss, dass die Trainingsszene nur eine Kugel umfasste und die Verteilung im Input-Set >99% positive Labels und <1% negative Labels besaß.

Es ist jedoch festzustellen, dass lineare Klassifizierungsmodelle bereits eine gute vergleichbare Genauigkeit bei einer geringeren Trainingszeit erreichen (vgl. Abschnitt 5.1.2). Z.B. konnten die Modelle mit der ID 8 und 9 Genauigkeitswerte zwischen 0.8 und 0.9 in unter 5 Stunden erreichen.

Während der Testreihen konnte erkannt werden, dass die Nutzung des Adam Optimizers als Optimierungsalgorithmus zu einer erheblichen Verbesserung des Trainings führt.

Viele Optimierungsalgorithmen sind darauf ausgelegt User-Inputs, Interessen oder auch Texte zu analysieren. Dies sind alles nicht deterministische Werte und daher nicht immer eindeutig bestimmbarer Wert. Bei diesen Daten kommt es aufgrund dieser Eigenschaft zu einer gehäuften Unregelmäßigkeit (auch bezeichnet als engl. Noise). Das kann unter Umständen zu Problemen, bei genauen (deterministischen) Daten

---

<sup>21</sup> Es wurden im Rahmen dieser Arbeit weitere Trainings (insgesamt fast 30) durchgeführt, jedoch wurden lediglich 15 Modelle vorgestellt, die ein repräsentatives und interpretierbares Ergebnis liefern.

führen, wie denen, die in dieser Arbeit vorliegen. Zum Beispiel konnte der generelle Gradient Descent Optimizer bei vorherigen Versuchen mit anderen Datensätzen, die nicht solche deterministischen Werte enthalten, gute Ergebnisse liefern. Dieser Optimierungsalgorithmus erwies sich jedoch für diese Forschungsreihe als nicht optimal. Insbesondere für neuronale Netzwerke ist der Adam Optimizer ein oft verwendeteter Algorithmus, der sich aktuell durchgesetzt<sup>22</sup> hat. Neben dem Optimierungsalgorithmus ist zu beachten, dass das Training mit einer großen Anzahl an Schnitttests als Input durchgeführt wurde. In fast allen Durchläufen wurden mehr als 800,000 Feature-Vektoren für das Training generiert und genutzt.

Es wird vermutet, dass das Modell durch das Training mit Hilfe der Samples die Geometrie der Szene erlernt. Mit diesem Wissen kann das Modell das Auftreten eines Schnittes mit der Szenengeometrie und einem gegebenen Strahl prognostizieren bzw. in der zweiten Testreihe die Sichtbarkeit zwischen zwei Punkten in der Szene vorhersagen.

Bei den verwendeten Strahlen stellt man an der Verteilung der Strahlen (vgl. Abschnitt 3.1.3.1f. und 3.2.33.1.3.2) fest, dass viele Strahlen sich sehr ähneln. Bei einer genaueren Analyse einzelner Samples aus dem Datensatz kann erkannt werden, dass sich die Daten an vielen Stellen lediglich am Nachkommaanteil unterscheiden und selbst dort erst an der x-ten Stelle (vgl. Tabelle 21).

**Tabelle 21** Ausschnitt Schnitttest Datensatz (Exportiert aus der pbrt Killeroo-Szene [1])

Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	hit
396.734711	54.7862091	30	-0.942072153	0.33537069	0.00517117977	1
-399.999817	338.417938	34.3734436	0.958314896	1.24889672e-08	-0.285714269	1
-399.999817	338.417938	34.3734436	0.745356023	-0.302660376	0.594004273	0
396.734711	54.7862091	30	-0.94228375	0.334784895	0.00452622771	1
-399.999817	337.858826	33.827179	0.218455017	0.922786713	0.317398936	0
-399.999817	337.858826	33.827179	0.885163188	-0.375686049	0.274492413	0
396.734711	54.7862091	30	-0.94217515	0.33508727	0.00475978851	1

---

<sup>22</sup> <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>

Allgemein wird in solchen Fällen beim maschinellen Lernen zur Behandlung solch eines marginalen Unterschiedes eine Skalierung vorgenommen. Diese wird aber in diesem Fall als nicht sinnvoll erachtet, wie in Abschnitt 3.3ff erläutert. Zusammengefasst wird beim Raytracing eine enorme Genauigkeit für die Schnitttests benötigt, so dass eine Skalierung diese Genauigkeit zerstören würde. In der Regel wird erwartet, dass ein Modell besser auf die Trainingsdaten angepasst ist, als auf Validierungs- oder Testdaten. Die Trainingsergebnisse dieser Versuchsreihen zeigten jedoch, dass der Unterschied zwischen diesen Daten-Teilmengen kaum erkennbar ist.

Aufgrund der beschriebenen Menge an Strahlen mit starker Ähnlichkeit und der Verwendung einer sehr großen Menge an Trainingssamples, führte dies in den Testversuchen zu dem Phänom, dass die Trainings-, Validierungs- und sogar Test-Evaluierungen sehr ähnlich und zum Teil identisch waren. Dies kann aus den Konvergenzkurven und Genauigkeits-Werten in Abschnitt 5 geschlossen werden. Es ist anzunehmen, dass bei komplexeren Szenen und einer geringeren Anzahl an Trainingsschnitttests dieser Anteil stärker divergiert.

Anhand der analysierten ROC-Kurven, insb. in Abschnitt 5.1, kann erkannt werden, dass fast alle im Rahmen dieser Arbeit erstellten Modelle eine stärkere Tendenz besitzen, die positive Kategorie richtig bzw. mit einer höheren Sicherheit zu prognostizieren. Bei der negativen Kategorie besitzen die Modelle eine höhere Varianz und dementsprechend eine größere Unsicherheit bei der Zuordnung. Dies kann daran erkannt werden, dass die Kurven sehr kontinuierlich auf der X-Achse (es wird für die negative Kategorie das Komplement betrachtet) abfallen. Im Gegensatz dazu besitzen die meisten Plots auf der Y-Achse eine konstante Steigung (von 1) auf bis zu einem bestimmten Wert, ab dem die Kurve sehr schnell abfällt (in Betrachtung der Y-Achse).

Dies führt dazu, dass das Modell eine gute Arbeit leistet, die positiven Ergebnisse zu finden und zu kategorisieren. Das Modell kann jedoch etwas schlechter die negativen Ergebnisse klassifizieren.

Dies muss für möglichen Anwendungsszenarien beachtet werden.

# 7.

## Praktische Anwendbarkeit & Ergebnisbewertung

Dieser Abschnitt diskutiert und bewertet die praktische Anwendbarkeit der aus dieser Arbeit resultierenden Erkenntnisse.

Es wurden in dieser Forschungsarbeit verschiedene Modelle darauf trainiert, für einen gegebenen Strahl<sup>23</sup> zu prognostizieren, ob dieser die Szenengeometrie trifft oder nicht, bzw. ob zwischen 2 Punkten im Raum eine Sichtbarkeit untereinander besteht.

Beide Problemstellungen beschreiben typische geometrische Probleme, die in der Computer Grafik, genauer beim Raytracing, angetroffen werden.

Es wird von allen Modellen ein Genauigkeitswert zurückgeliefert. Dieser wird dann mit einem Schwell-Wert (engl. Threshold) in einen bool'schen Wert konvertiert, der die Fragestellung beantwortet.

Im Grunde stellt sich die Frage, ob es möglich ist, ein Modell auf eine Szenengeometrie zu trainieren. Die beiden gewählten Ansätze in dieser Arbeit sind strahlenbasiert, da eine Repräsentation der Szene gefunden werden muss, die diese akkurat beschreibt und gleichzeitig eine Form bietet, mit der das Modell neue Wert prognostizieren kann. So ist es z.B. nicht möglich, das Netzwerk mit reinen Geometriedaten (Vertices & Primitiven) zu trainieren und bei der späteren Nutzung für Punkte die Sichtbarkeit abzufragen. Es gilt Trainingsdatenformat = Abfrage-/Testdatenformat.

Beim Feature Engineering wurden verschiedene Ansätze diskutiert, synthetische Daten aus den ursprünglichen Daten zu generieren. Es konnte festgestellt werden, dass sich durch diese Ergänzung bei den Modellen die Trainingsdauer verlängerte und die Größe entsprechend wuchs. Es resultierten keine nennenswerten

---

<sup>23</sup> Wie in Abschnitt 2.1 beschrieben besteht ein Strahl aus Ursprung und Richtung.

Verbesserungen beim Modelltraining durch die Nutzung der getesteten synthetischen Inputs.

Es wurden verschiedene Modellansätze (u.a. Formen von linearen Klassifizierungsmodellen, DNN) evaluiert die im Bereich des maschinellen Lernens Anwendung finden. Es konnte erkannt werden, dass neuronale Netzwerke für dieses Problem die besten Resultate liefern konnten, diese jedoch mit einem großen Zeitaufwand beim Training verbunden waren. Der große Zeitaufwand konnte auch beim Abfragen des Modells bei manuellen Tests beobachtet werden, wodurch die konstante Zeitkomponente des Systems stieg und dieser Faktor bei der Nutzung mit beachtet werden muss.

In Bezug auf den zeitlichen Aufwand konnte bei allen Trainingsdurchläufen eine durchschnittliche Trainingszeit von 6 Stunden gemessen werden. Es wurde eine hohe Anzahl an Schnitttests bzw. Punktsichtbarkeiten als Input verwendet, um die simplen verwendeten Szenen möglichst akkurat zu beschreiben. Für die verwendeten Szenen ist es vorstellbar, dass eine geringere Anzahl ähnliche Ergebnisse liefern würde. Diese Trainings sollen auch als Anhaltspunkt für komplexe Szenen dienen, die in der Regel beim Raytracing vorliegen, und bei denen eine solche Anzahl an Schnitttests benötigt wird, um diese akkurat zu beschreiben.

Aus den Versuchsreihen konnte erkannt werden, dass das Problem der Sichtbarkeit von Punkten nicht mit den untersuchten Methoden in effizienter Weise lösbar ist.

Sämtliche trainierte Modelle konnten keine zuverlässige Kategorisierung der Ergebnisse liefern. Bei Versuchen mit einer hohen Genauigkeit konnte dies auf eine Unterrepräsentation von Werten im Datensatz zurückgeführt werden. Eine Korrektur führte wieder zu einem schlechteren durchschnittlichen Genauigkeitswert. In den erstellten Datensätzen waren lediglich einfache Geometrien repräsentiert, mit denen keine erfolgreichen Ergebnisse erreicht werden konnten. Der für diese Arbeit implementierte Raytracer (wie auch der pbrt Renderer) ist jedoch fähig, auch komplexere und nicht konvexe Geometrien zu rendern und entsprechende Schnitttest-Datensätze zu erstellen.

Beim zweiten Ansatz ging es generell darum, zu untersuchen, ob ein Modell die Geometrie einer Szene „erlernen“ kann. Dafür wurde das Modell mit den Schnitttestergebnissen eines Raytrace Durchlaufs trainiert und sollte für einen gegebenen Strahl mit Ursprung aussagen, ob dieser Strahl die Szenengeometrie trifft

oder ins „Leere“ geht. Es konnte bei diesen Versuchen erkannt werden, dass die Modelle je nach Trainingsdauer und Parametern signifikante Verbesserungen und Genauigkeitswerte von ca. 0.9 erreichen konnten. Dies gelang jedoch auf Kosten der Trainingsdauer von ca. 8 Stunden. Die hohen Genauigkeitswerte zeigten jedoch an einigen Stellen die Probleme auf, dass die Modelle die Tendenz aufwiesen, hauptsächlich die positiven Schnitttests richtig zu kategorisieren. Es trat eine höhere Fehlerquote bei den negativen Klassifizierungen auf. Insgesamt wird angenommen, dass die Fehler hauptsächlich an den Kanten der Geometrie stattfinden, bei denen auch beim normalen Raytrace-Algorithmus floating-point Ungenauigkeiten auftreten könnten.

Ein weiterer Punkt, der bei den Ergebnissen zu beachten ist, ist, dass ein hoher Genauigkeitswert generell ein positives Zeichen ist, jedoch lediglich eine Aussage über die absolut richtig zugeordneten Kategorien macht. Wie Tabelle 22 (Confusion Matrix) zeigt, kann eine Prognose eines Modells im Rahmen des Raytracings zu 4 verschiedenen möglichen Resultaten führen.

**Tabelle 22** Raytracing Modell Confusion Matrix

<b>True Positive (TP):</b> Ein Strahl, der die Szenengeometrie trifft, wird als ein solcher prognostiziert.	<b>False Positive (FP):</b> Ein Strahl, der die Szenengeometrie nicht trifft, wird fälschlicherweise als ein Strahl prognostiziert, der die Szene schneidet.
<b>False Negative (FN):</b> Ein Strahl, der die Szenengeometrie trifft, wird fälschlicherweise als ein Strahl prognostiziert, der die Szene nicht schneidet.	<b>True Negative (TN):</b> Ein Strahl, der die Szenengeometrie nicht trifft, wird als ein solcher prognostiziert.

Die beiden in grün gekennzeichneten Möglichkeiten sind das optimale Ergebnis.

Bei einem Genauigkeitswert von 1.0 liefert das Modell immer die richtigen Zuordnungen und dementsprechend eines der beiden grün gekennzeichneten Ereignisse. In der Regel besitzen Modelle, die mittels maschinellem Lernen trainiert wurden, meist keinen genauen Wert von 1.0 und auch die, in dieser Forschungsarbeit trainierten Modelle haben eine gewisse Fehlerquote.

Um die beste Zuordnung zu erreichen, müssen in diesem Fall auch die falschen Zuordnungen beachtet werden. Je nach Anwendungszweck muss der Threshold so

gewählt werden, dass die falschen Zuordnungen entweder ausgewogen sind oder so, dass die Kategorie mit einem gravierenderem negativ Effekt minimiert wird.

In einem Szenario, in dem das Modell als Proxy-Modell zur Überprüfung verschiedener Strahlen genutzt wird und abhängig von dem Ergebnis einen detaillierten Schnitttest durchführt, um weitere Parameter zu bestimmen (z.B. Schnittpunkt), haben die beiden False Klassifizierungen eine unterschiedliche Wirkung. Da bei der genauen Überprüfung die geometrisch exakte Lösung bestimmt wird, würden False Positives implizit korrigiert werden. An dieser Stelle entsteht lediglich ein Mehraufwand an Zeit. Bei False Positives wird jedoch der Strahl nicht genauer analysiert, sondern verworfen, was zu geometrischen Fehlern und dementsprechend Fehlern im Ausgabebild (z.B. Rauschen) führen würde.

Im direkten Vergleich mit den Beschleunigungsstrukturen, die in modernen Raytracer angewendet werden und in Abschnitt 2.3 vorgestellt wurden, können die Modelle an Leistung und Geschwindigkeit nicht mit diesen mithalten. Seit der Nutzung des Raytrace-Algorithmus ist eine „[...]“ große Menge an Forschung in Beschleunigungsstrukturen geflossen [...]“ (Kapitel 4.2 [1]) und diese sind hochoptimiert. Z.B. konnte die verwendete pbrt “Killeroo”-Simple Scene innerhalb 30 bis 40 Sekunden komplett gerendert werden mit sämtlichen knapp 15 Mio Schnitttests. Das Modelltraining für einen Genauigkeitswert von mehr als 0.9 benötigte knapp 17 Stunden. Es ist jedoch die Idee entstanden, für die Beschleunigung der Traversierung dieser Strukturen einen AI-basierten Ansatz zu wählen, der einen Einstiegspunkt in diese Strukturen liefern könnte.

Ein weiteres Problem, dass eine praktische Anwendbarkeit erschwert, ist, dass sich keine Generalisierbarkeit aus den Trainingsdurchläufen ableiten lies und die Modelle für eine statische Szene trainiert sind, die z.B. keine Animationen beinhaltet.

All diese Beobachtungen führen zu dem finalen Schluss, dass Modelle des maschinellen Lernens nicht gut für den vorgestellten geometrischen Teil des Raytrace-Algorithmus geeignet sind. Das hauptsächliche Problem besteht darin, dass gerade dieser Teil eine bestmögliche Genauigkeit benötigt. Bei Modellen im Bereich des maschinellen Lernens basiert das Training auf Lernen aus Erfahrung und gleiche oder sehr ähnliche Beispiel können mithilfe dessen auf neue Beispiele angewendet werden. Bei den Geometriedaten, bei denen selbst eine hohe Nachkommastelle einen Unterschied macht, konnte dieser approximative Ansatz nicht zum Erfolg führen.



# Literaturverzeichnis

- [1] M. Pharr, W. Jakob und G. Humphreys, Physically Based Rendering: From Theory To Implementation, Morgan Kaufmann, 2016.
- [2] Frauenhofer-Institut, „MASCHINELLES LERNEN - EINE ANALYSE ZU KOMPETENZEN, FORSCHUNG UND ANWENDUNG,“ 2018.
- [3] A. S. Kaplanyan, C. R. A. Chaitanya, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai und T. Aila, „Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder,“ [Online]. Available: [https://research.nvidia.com/sites/default/files/publications/dnn\\_denoise\\_author.pdf](https://research.nvidia.com/sites/default/files/publications/dnn_denoise_author.pdf). [Zugriff am April 2019].
- [4] M. v. Übel, „All3DP 25 Best 3D Rendering Software Tools in 2019,“ April 2019. [Online]. Available: <https://all3dp.com/1/best-3d-rendering-software/>.
- [5] E. Haines, P. Hanrahan, R. L. Cook, J. Arvo, D. Kirk and P. S. Heckbert, An Introduction to Ray Tracing (The Morgan Kaufmann Series in Computer Graphics), Academic Press; 1st edition, (February 11, 1989).
- [6] Wikipedia, „Raytracing,“ [Online]. Available: <https://de.wikipedia.org/wiki/Raytracing>. [Zugriff am Juli 2019].
- [7] H. Pritchett und R. Tamstorf, „Disney Moana Island Scene Read-Me,“ Disney, <http://datasets.disneyanimation.com/moanaislandscene/island-README-v1.1.pdf>, 2016.
- [8] I. G. a. Y. B. a. A. Courville, Deep Learning, MIT Press, 2016.
- [9] Google Developers, „Machine Learning Crash Course,“ Google, 2019. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/>. [Zugriff am July 2019].

- [10 Wackerly, Mendenhall und Scheaffer, Mathematical Statistics with Applications  
] 7th Edition, Thomson Brooks/Cole, 2008.
- [11 E. W. Weisstein, „Direction Cosine.“ From MathWorld,“ A Wolfram Web  
] Resource., [Online]. Available:  
<http://mathworld.wolfram.com/DirectionCosine.html>. [Zugriff am 07 2019].
- [12 D. P. Kingma and J. Ba, "Adam: A Method for stochastic Optimization," ICLR,  
] 2015.
- [13 H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T.  
] Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M.  
Hrafnelsson, T. Boulos und J. Kubica, „Ad Click Prediction: a View from the  
Trenches,“ Google.
- [14 A. S. Glassner, An Introduction to Ray Tracing, Morgan Kaufmann, 1989.  
]
- [15 M. De Berg, O. Cheong, M. van Kreveld und M. Overmars, Computational  
] Geometry - Algorithms and Applications, 3rd Edition, Springer-Verlag Berlin  
Heidelberg, 2008.
- [16 „IEEE Standard for Binary Floating-Point Arithmetic," in ANSI/IEEE Std 754-  
] 1985,“ IEEE, 12 Oct. 1985.
- [17 P. Shirley, Ray Tracing in One Weekend, 2016.  
]
- [18 Intel, „Intel KI-Workshop für Entwickler,“ Hamburg, 21. Mai 2019.  
]

## Eidesstaatliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

---

Ort, Datum

---

Unterschrift (Vor- und Nachname)