

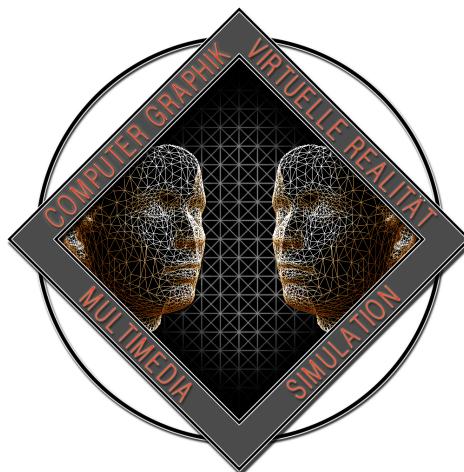
THESIS

---

Real-time Neural Radiance Caching in Heterogeneous  
Participating Media

---

Jan Spindler



INSTITUT FÜR INFORMATIK II - COMPUTER GRAPHIK  
UNIVERSITÄT BONN

Erstgutachter: Prof. Dr. Reinhard Klein  
Zweitgutachter: Prof. Dr. Matthias B. Hullin

31. Dezember 2022



---

## Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

---

Ort, Datum

---

Unterschrift



# Zusammenfassung

---

Das realistische Rendern von Heterogeneous Participating Media in Echtzeit ist ein komplexes Problem. Auch auf moderner Grafikhardware benötigen Monte Carlo Path Tracer viel Rechenzeit, um Bilder mit ausreichender Qualität zu generieren. Mit dem Neural Radiance Caching wurde kürzlich ein Ansatz vorgestellt, der mittels neuronalen Netzen vielversprechende globale Beleuchtung in Echtzeit vorweist. In dieser Arbeit wird diese neu entwickelte Methode auf das Rendern von Heterogeneous Participating Media angewendet und die Ergebnisse hinsichtlich Qualität und Performanz untersucht. Weiterhin wird der Einfluss von verschiedenen Parametern, wie Netzwerkstruktur und Input Encoding, näher betrachtet. Es zeigt sich, dass das Neural Radiance Caching in der Lage ist, das Rauschen eines Path Tracers zu minimieren, während die benötigte Zeit vergleichbar mit der eines klassischen Monte Carlo Path Tracers bleibt.



# **Danksagungen**

---

Ich möchte mich herzlich bei all denjenigen bedanken, die mich in der gesamten Zeit meines Studiums unterstützt und motiviert haben.

Als aller erstes möchte ich mich bei Herrn Prof. Dr. Reinhard Klein und Herrn Prof. Dr. Matthias B. Hullin und dem Lehrkörper des Instituts für Informatik 2 dafür bedanken, dass ich mich im Rahmen meiner Bachelorarbeit mit einem spannenden und praxisorientierten Thema beschäftigen durfte.

Mein besonderer Dank gilt Herrn Tom Kneiphof, der mich während meiner Bachelor Projektgruppe Computergraphik und bei meiner Bachelorarbeit betreut hat.

Ich bedanke mich bei meinem ehemaligen Klassen- und Mathematiklehrer Herrn Klaus Rabus unter anderem für die Beaufsichtigung meiner gymnasialen Facharbeit, in der ich mich zum ersten Mal mit den mathematischen Grundlagen der Computergraphik beschäftigt habe.

Abschließend danke ich meinen Eltern, die mir mein Studium durch ihre Unterstützung ermöglicht haben.

Jan Spindler

Bonn, 31.12.2022



# Inhaltsverzeichnis

---

<b>1 Einleitung</b>	<b>1</b>
<b>2 Verwandte Arbeiten</b>	<b>3</b>
2.1 Volumetrisches Echtzeitrendering . . . . .	3
2.2 Radiance Caching . . . . .	4
2.3 Neural Rendering . . . . .	4
<b>3 Grundlagen</b>	<b>7</b>
3.1 Beleuchtungsmodell . . . . .	7
3.2 Volumetrisches Path Tracing . . . . .	10
3.3 Multilayer Perzeptron . . . . .	15
<b>4 Methodik</b>	<b>23</b>
4.1 Neural Radiance Cache . . . . .	23
4.2 Rendering Pipeline . . . . .	25
4.3 Implementierung . . . . .	26
<b>5 Evaluation</b>	<b>29</b>
5.1 Versuchsaufbau . . . . .	29
5.2 Laufzeitkosten . . . . .	29
5.3 Qualität . . . . .	31
<b>6 Fazit</b>	<b>35</b>
<b>Literatur</b>	<b>37</b>



# 1

## Einleitung

---

Ziel der Computergraphik ist die realistische Bildsynthese von verschiedenen Szenen, auch Rendering genannt. Viele Szenen enthalten visuelle Effekte wie Wolken und Rauch, die als Participating Media bezeichnet werden. Participating Media sind teildurchlässig und erhalten ihre Optik durch die Interaktion des Lichts mit vielen im Raum verteilten Partikeln. Das Aussehen des Mediums wird maßgeblich durch die Dichte und Verteilung der Partikel beeinflusst. Insbesondere Wolken gehören zu der Klasse der Heterogeneous Participating Media (HPM), deren Dichteverteilung nicht gleichmäßig (heterogen) ist.

Wie in Kapitel 3 erläutert wird, erfordert realistisches Rendern von Heterogeneous Participating Media das Simulieren von Lichtpfaden. Dies wird unter anderem bei Wolken problematisch, da dort Licht kaum absorbiert wird und die Lichtpfade viele Streuereignisse enthalten. Diese Lichtpfade können durch volumetrische Path Tracer erzeugt und beleuchtet werden, wodurch eine realitätsgtreue Abbildung entsteht. Einzelne Pfade variieren stark in ihrer Beleuchtung, was sich in einem starken Rauschen äußert. Um eine Ground-Truth zu erzeugen, benötigen Path Tracer deshalb tausende oder Millionen Pfade. Die dafür benötigte Rechenleistung kann auch von moderner Grafikhardware nicht in Echtzeit bereitgestellt werden.

Wie Bittner (2020) zeigt, nutzen viele Echtzeit-Anwendungen einen Ray Marcher, der mehrere einzelne Streuereignisse entlang eines Kamerastrahls akkumuliert. Dies hat den Vorteil, dass das Bild in Echtzeit und ohne das Rauschen eines Path Tracers generiert wird. Da so ein großer Teil des transportierten Lichts nicht simuliert wird, kommt es zu einem sichtbaren Verlust von Qualität und Realismus. Eine Möglichkeit, das Rendern zu erleichtern, ist das Speichern von bereits berechneter Radianz. Jarosz, Donner u. a. (2008) stellen eine Methode vor, die Radianzen an mehreren Stellen speichert und durch Extrapolation wiederverwendet. Diese Methode erzeugt bessere Ergebnisse als ein Path Tracer, aber auch zusätzliche Kosten durch die Extrapolation und durch teure Updates der radianzspeichernden Datenstruktur. Jarosz, Donner u. a. (2008) zeigen das Potential des Radiance Caching in Heterogeneous Participating Media.

Müller, Rousselle u. a. (2021) nutzen für das Speichern und Wiederverwenden von berechneten Radianzen ein neuronales Netz. Beim Rendern wird ein Path Tracer frühzeitig in den sogenannten Neural Radiance Cache (NRC) terminiert, der während der Laufzeit die Radianzverteilung

---

der Szene lernt. Diese Kombination aus Path Tracer und neuronalem Netz implementiert echtzeitfähige globale Beleuchtung. Durch das ständige Lernen kann sich der NRC schnell an dynamische Szenen anpassen. Müller, Rousselle u. a. (2021) zeigen auch, dass der NRC bei Heterogeneous Participating Media die Qualität im Vergleich zu einem Path Tracer wesentlich verbessert. Weitere Untersuchungen des volumetrischen Renderns mithilfe eines NRC lassen Müller, Rousselle u. a. (2021) offen für zukünftige Arbeiten.

Diese Arbeit knüpft hier an und untersucht in Kapitel 5 die Qualität und Performanz, die ein NRC beim Rendern von Heterogeneous Participating Media vorweist. Dabei wird der NRC insbesondere mit einem volumetrischen Path Tracer verglichen. Zusätzlich werden die Auswirkungen von verschiedenen Parametern wie Netzwerkstruktur und Input Encoding untersucht. Wie die Evaluation zeigt, erzielt das Neural Radiance Caching in den meisten Szenarien rauschfreiere Ergebnisse als ein volumetrischer Path Tracer bei teils höheren Bildraten von mehr als 30 Bildern pro Sekunde.

Die weitere Arbeit ist wie folgt strukturiert. Kapitel 2 geht auf relevante Arbeiten aus den Bereichen volumetrisches Echtzeitrendering, Radiance Caching und Neural Rendering ein. In Kapitel 3 werden die theoretischen Grundlagen, auf denen diese Arbeit basiert, erläutert. In Kapitel 4 wird der Aufbau und die Umsetzung des volumetrischen Renderers dargelegt. Kapitel 5 betrachtet den Einfluss verschiedener Parameter auf die Qualität und Performanz des Renderers. Kapitel 6 fasst die Ergebnisse dieser Arbeit zusammen und präsentiert Erweiterungsmöglichkeiten für zukünftige Arbeiten.

# 2

# Verwandte Arbeiten

---

In diesem Kapitel werden verwandte Arbeiten vorgestellt und deren Relevanz im Kontext dieser Arbeit kurz erläutert. Diese Arbeiten behandeln alternative Renderingmethoden von Heterogeneous Participating Media, das Beschleunigen von Renderern mittels Radiance Caching und Forschungsergebnisse im Bereich Neural Rendering.

## 2.1 Volumetrisches Echtzeitrendering

**The Current State of the Art in Real-Time Cloud Rendering With Raymarching.** Echtzeitanwendungen sind oft darauf angewiesen, vereinfachende Annahmen zu treffen, um die nötige Performanz und Stabilität einer Simulation sicherzustellen. Bittner (2020) präsentiert aktuelle Methoden, die das Rendern von Wolken in Echtzeit mit ausreichender Qualität ermöglichen. Dazu werden Ray Marcher eingesetzt. Diese erzeugen im Vergleich zu einem Monte Carlo Path Tracer keine zufälligen Pfade mit mehreren Streuereignissen. Für jedes Pixel werden entlang eines Strahls durch das Volumen mehrere Punkte in gleichmäßigen Abständen gesampelt. An diesen Punkten wird ein einzelnes Streuereignis in Richtung der Lichtquelle simuliert. Die Radianz dieser Streuereignisse wird entlang des Strahls akkumuliert. Da die Punkte deterministisch erzeugt werden und das Streuereignis immer in Richtung der Lichtquelle ausgewertet wird, generiert ein Ray Marches rauschfreie Ergebnisse. Wie in der Einleitung erläutert, wird durch die Simulation eines einzelnen Streuereignisses ein Teil der Beleuchtung nicht berücksichtigt.

**Fast Volume Rendering with Spatiotemporal Reservoir Resampling.** Bitterli u. a. (2020) stellen den ReSTIR Algorithmus vor, der die effiziente Berechnung direkter Beleuchtung von Millionen Lichtquellen umsetzt. Dazu wird aus einer Menge von möglichen Lichtquellen gesampelt. Diese Menge setzt sich aus temporär und räumlich wiederverwerteten Lichtquellen zusammen. Durch diese Wiederverwertung von Lichtquellen sinkt die Varianz des Renderers, so dass sich der Fehler um bis zu zwei Größenordnungen verringert. Lin u. a. (2021) erweitern den ReSTIR Algorithmus auf das volumetrische Path Tracing und erzielen durch die verringerte Varianz eine erhebliche Verbesserung der Qualität des Path Tracers. Statt Lichtquellen verwerten Lin u. a. (2021) Pfade wieder, die der Path Tracer bereits generiert hat.

## 2.2 Radiance Caching

**Radiance caching for participating media.** Jarosz, Donner u. a. (2008) wenden das Radiance Caching auf Participating Media an und erzielen so bessere Ergebnisse als ein volumetrischer Path Tracer. Radianzen werden in einem Octree gespeichert und beim Rendern wiederverwendet. Das Rendering erfolgt mittels Ray Marching. Die gespeicherten Radianzen werden mittels Gradienten extrapoliert und können so in den Ray Marcher inkorporiert werden. Diese Methode ist darauf angewiesen, die Fehler durch Extrapolation möglichst genau zu schätzen. Marco u. a. (2018) bauen auf der Arbeit von Jarosz, Donner u. a. (2008) auf und führen zusätzlich Radianzableitungen zweiter Ordnung ein, wodurch der Fehler durch Extrapolation noch präziser geschätzt werden kann. Um die Komplexität und den Speicherverbrauch zu senken, gehen Marco u. a. (2018) in ihrer Arbeit von homogenen Medien aus. Auch mit der zweiten Ableitung hat der Ansatz von Marco u. a. (2018) das Problem, dass Radianzverteilungen mit hohen Frequenzen, zum Beispiel durch kleine Lichtquellen, nicht richtig approximiert werden können.

**Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields.** Majercik u. a. (2019) stellen eine probenbasierte Methode vor, die indirekte Beleuchtung in Echtzeit ermöglicht. Diese Proben werden in der Szene verteilt und speichern Beleuchtungs- und Geometriedaten. Die Proben werden in Echtzeit aktualisiert, was das Rendern von dynamischen Szenen erlaubt. Pfade werden nach ihrer ersten diffusen Interaktion in die Irradianzproben terminiert. Durch diese kurzen Pfade ist DDGI sehr performant und erzielt Ergebnisse mit wenig Rauschen. Müller, Rousselle u. a. (2021) zeigen, dass ein NRC bei vergleichbaren Laufzeitkosten zwar einen geringeren Bias, aber dafür eine erhöhte Varianz aufweist als DDGI.

## 2.3 Neural Rendering

**Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks.** Kallweit u. a. (2017) präsentieren eine Methode, die neuronale Netze zur Berechnung von volumetrischer Beleuchtung nutzt. Für Punkte in einer Wolke wird ein drei-dimensionaler hierarchischer Descriptor extrahiert. Dieser wird als Eingabe für ein neuronales Netz verwendet, welches die Radianz berechnet. Das neuronale Netz wird vorher an einem Datensatz aus verschiedenen Wolken trainiert. Mittels dieser Methode können komplexe Wolken realistisch in wenigen Minuten gerendert werden. Kallweit u. a. (2017) zeigen erfolgreich das Potential von neuronalen Netzen, volumetrisches Rendern zu beschleunigen. Jedoch eignen sich die Radiance-Predicting Neural Networks aufgrund von langen Trainingszeiten nicht für das Echtzeitrendering von Heterogeneous Participating Media in dynamischen Szenen.

**Neural Importance Sampling.** Die Qualität eines Monte Carlo Path Tracers ist stark von der Qualität des zugrunde liegenden Importance Samplings abhängig. Müller, Mcwilliams u. a. (2019) nutzen neuronale Netze, um das Importance Sampling für Monte Carlo Integration zu verbessern.

## KAPITEL 2. VERWANDTE ARBEITEN

---

Dieses verbesserte Sampling ermöglicht unter anderem ein neuronales Path Guiding, das die Varianz eines Path Tracers reduziert. Zusätzlich wird das One-Blob Encoding vorgeschlagen, welches von Müller, Rousselle u. a. (2021) bei der Implementierung des Neural Radiance Caches für das Encodieren der Richtung eines Strahls verwendet wird.

**Instant Neural Graphics Primitives with a Multiresolution Hash Encoding.** Das von Müller, Evans u. a. (2022) vorgestellte Multiresolution Hash Encoding (MRHE) ist in der Lage, das Generieren verschiedener neuronaler grafischer Primitive wesentlich zu beschleunigen. Zu diesen Primitiven gehören auch Neural Radiance Fields (NeRF) und Neural Radiance Caches. Dieses Encoding ermöglicht die Verkürzung der Trainingszeit bei einem NeRF von mehreren Stunden auf wenige Sekunden. In dieser Arbeit wird neben anderen Input Encodings auch das Multiresolution Hash Encoding verwendet und dessen Einfluss auf die Qualität der Ergebnisse und die Laufzeit der Renderers untersucht.



# 3

# Grundlagen

---

In diesem Kapitel werden die mathematischen und physikalischen Grundlagen erläutert, auf die sich diese Arbeit stützt. Das Beleuchtungsmodell von HPM wird erklärt und gezeigt, wie auf dieser Basis ein Monte Carlo Path Tracer konstruiert werden kann. Die Erklärung des Beleuchtungsmodells und der Rendering Methoden folgen im Wesentlichen der Arbeit von Novák, Georgiev u. a. (2018). Darauf folgen die Grundlagen zu neuronalen Netzen auf denen der NRC aufbaut.

## 3.1 Beleuchtungsmodell

Im Folgenden werden die möglichen Interaktionen von Licht und HPM dargestellt. Abbildung 3.1 zeigt, wie sich Absorption, Out-Scattering, In-Scattering und Emission auf die Radianz entlang eines Strahl durch das Medium auswirken. Anschließend wird eine Gleichung formuliert, die unter Bezugnahme auf die genannten Licht-Medien Interaktionen den Lichttransport entlang eines Strahls angibt.

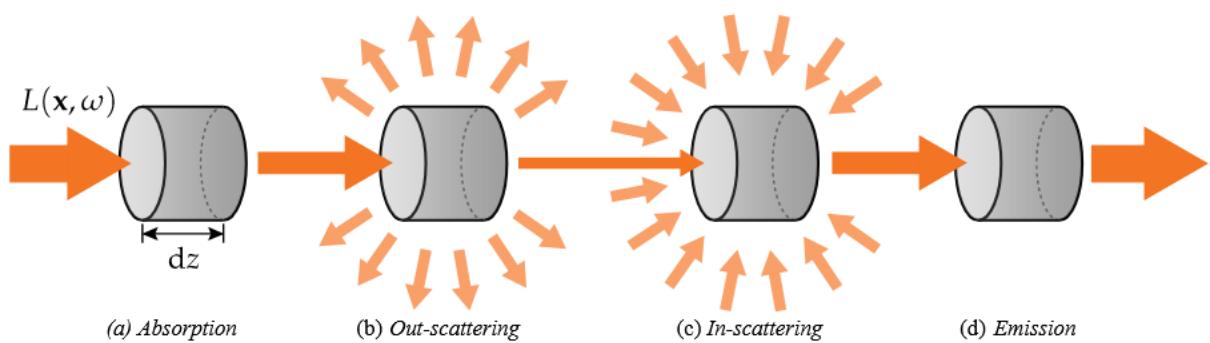


Abbildung 3.1: Arten der Licht-Medien Interaktion (Novák, Georgiev u. a., 2018)

HPM bestehen aus einer Menge von vielen Partikeln, die mit Photonen interagieren. Aufgrund der großen Anzahl an Partikeln werden diese nicht einzeln simuliert, sondern als eine Dichteverteilung  $\rho(x)$  [ $m^{-3}$ ] betrachtet. Die Dichte ist im Allgemeinen positionsabhängig und wird deshalb auch heterogen genannt. Gehen wir nun davon aus, dass ein Lichtstrahl an der Position  $x$  in Richtung

### 3.1. BELEUCHTUNGSMODELL

---

$\omega$  entlang eines differentiellen Streckenelements  $dz$  [m] das Medium durchquert.  $\sigma_a$  [ $m^2$ ] und  $\sigma_s$  [ $m^2$ ] geben die Querschnittsfläche von absorbierenden und streuenden Partikeln an. Werden diese Querschnittsflächen mit der Dichte  $\rho(x)$  multipliziert, ergeben sich die Absorptions- und Streuungskoeffizienten  $\mu_a(x)$  [ $m^{-1}$ ] und  $\mu_s(x)$  [ $m^{-1}$ ].

**Absorbtion.** Beim Traversieren des Mediums kann Licht von den Partikeln absorbiert und in eine andere Energieform wie Wärme umgewandelt werden. Gleichung 3.1 zeigt die dadurch entstehende Reduktion in der Radianz in Richtung  $\omega$ . Der Absorptionskoeffizient  $\mu_a(x)$  gibt an, wie viel Licht durch das Medium absorbiert wird.

$$\frac{dL(x, \omega)}{dz} = -\mu_a(x)L(x, \omega) \quad (3.1)$$

**Out-Scattering.** Das Medium kann das Licht streuen. Dadurch strahlt ein Anteil des Lichts  $L(x, \omega)$  nach dieser Interaktion in andere Richtungen. Somit verringert sich die Radianz in Richtung  $\omega$ . Gleichung 3.2 zeigt, wie das Out-Scattering die Radianz in Richtung  $\omega$  beeinflusst. Der Streuungskoeffizient  $\mu_s(x)$  gibt an, wie viel Licht in andere Richtungen gestreut wird.

$$\frac{dL(x, \omega)}{dz} = -\mu_s(x)L(x, \omega) \quad (3.2)$$

**In-Scattering.** Wird Licht aus anderen Richtungen in Richtung  $\omega$  gestreut, erhöht sich die Radianz  $L(x, \omega)$ . Gleichung 3.3 zeigt, wie das In-Scattering die Radianz beeinflusst.

$$\frac{dL(x, \omega)}{dz} = \mu_s(x)L_s(x, \omega) \quad (3.3)$$

$$L_s(x, \omega_o) = \int_{S^2} p(\omega_o, \omega_i)L(x, \omega_i)d\omega_i \quad (3.4)$$

In Abhängigkeit von dem Winkel zwischen Einfalls- und Ausfallsrichtung des Lichts wird dieses unterschiedlich stark gestreut. Dieser Zusammenhang ist durch die Phasenfunktion  $p(\omega_o, \omega_i)$  gegeben. In dieser Arbeit werden Medien betrachtet, deren Partikel mindestens so groß sind wie die Wellenlänge des Lichts. Die dadurch entstehende Streuung wird auch Lorenz-Mie Streuung genannt (Mie, 1908). In dieser Arbeit wird die Henyey-Greenstein Phasenfunktion (Henyey und Greenstein, 1940) aufgrund der geringen Berechnungskosten verwendet. Gleichung 3.5 zeigt die Henyey-Greenstein Phasenfunktion. Diese ist hier nach dem Cosinus des Winkels  $\theta$  zwischen Einfalls- und Ausfallsrichtung parametrisiert.

$$h*g(\cos\theta) = \frac{1}{4\pi} \frac{1-g^2}{[1+g^2-2g \cdot \cos\theta]^{3/2}} \quad (3.5)$$


---

### KAPITEL 3. GRUNDLAGEN

---

Die Variable  $g \in [-1, 1]$  steuert, wie stark das Licht nach vorne gestreut wird. Bei  $g = 0$  entspricht die Henyey-Greenstein Phasenfunktion der Isotropen Phasenfunktion. Für  $g$  nahe 1 modelliert die Henyey-Greenstein Phasenfunktion ein starkes Streuen nach vorne, was unter Anderem für Wolken charakteristisch ist (Högfeldt, 2016). Abbildung 3.2 zeigt die Henyey-Greenstein Phasenfunktion mit verschiedenen Werten für  $g$ .

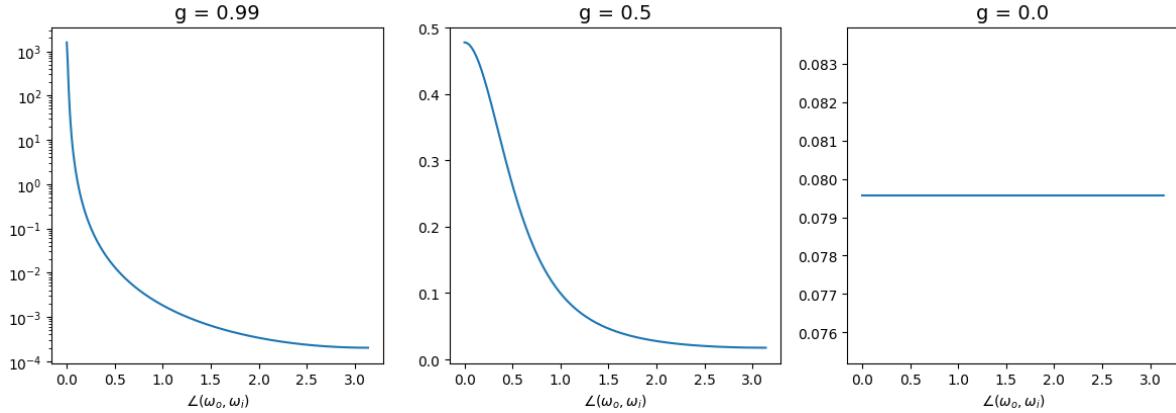


Abbildung 3.2: Henyey-Greenstein Phasenfunktion in Abhängigkeit von  $\theta$  mit  $g \in \{0.99, 0.5, 0.0\}$

**Emission.** Partikel in einem Medium können andere Energieformen in Licht konvertieren. Durch diese Emission erhöht sich die Radianz, wie Gleichung 3.6 zeigt.

$$\frac{dL(x, \omega)}{dz} = \mu_a(x)L_e(x, \omega) \quad (3.6)$$

**Strahlungstransportgleichung.** Durch Kombination von Absorbtion, In-Scattering, Out-Scattering und Emission ergibt sich die Stahlungstransportgleichung (Novák, Georgiev u. a., 2018). Diese gibt die Radianzänderung in einem Medium an der Position  $x$  in Richtung  $\omega$  entlang eines differentiellen Streckenelements  $dz$  an.

$$\frac{dL(x, \omega)}{dz} = -\mu_a(x)L(x, \omega) - \mu_s(x)L(x, \omega) + \mu_a(x)L_e(x, \omega) + \mu_s(x)L_s(x, \omega) \quad (3.7)$$

**Auslöschung.** Die Reduktion der Radianz durch Absorbtion und Out-Scattering lässt sich als Auslöschung zusammenfassen, wie Gleichung 3.8 zeigt. Die Stärke der Auslöschung wird durch den Auslöschungskoeffizienten  $\mu_t(x) [m^{-1}]$  bestimmt.

$$\frac{dL(x, \omega)}{dz} = -\mu_t(x)L(x, \omega) \quad \text{mit } \mu_t(x) = \mu_a(x) + \mu_s(x) \quad (3.8)$$

**Transmittance.** Für das Rendern von HPM ist nicht nur die Änderung der Radianz an einem Punkt im Raum wichtig, sondern auch entlang eines Strahls. Geht von einem Punkt  $y$  in Richtung  $-\omega$

## 3.2. VOLUMETRISCHES PATH TRACING

---

eine Radianz  $L(y, -\omega)$  aus, die auf einen Punkt  $x$  trifft, geht ein Teil dieser Radianz auf dem Weg durch Auslöschung verloren. Der Anteil der Radianz, der verloren geht, ist durch die Transmittance gegeben. Gleichung 3.9 zeigt, wie sich die Transmittance nach dem Lambert-Beerschen Gesetz berechnen lässt (Novák, Georgiev u. a., 2018).

$$T(x, y) = e^{-\int_0^z \mu_t(x + z\omega) dz} \quad \text{mit } z = \|y - x\|_2 \quad (3.9)$$

**Renderinggleichung.** Mit der Strahlungstransportgleichung und der Transmittance lässt sich eine Gleichung formulieren, aus der ein Renderer konstruiert werden kann. Betracht wird ein Kamerastrahl, der von der Position  $x$  aus in Richtung  $\omega$  zeigt. Mit Gleichung 3.10 kann die Radianz berechnet werden, die  $x$  aus der Richtung  $-\omega$  erreicht.

$$L(x, -\omega) = \int_0^\infty T(x, y) [\mu_a(x)L_e(y, -\omega) + \mu_s(x)L_s(y, -\omega)] dz \quad (3.10)$$

mit  $y = x + z\omega$

Hier wird die Problematik des Renders in Heterogeneous Participating Media deutlich. Der Integrand aus Gleichung 3.10 enthält weitere Integrale durch die Transmittance und das In-Scattering. Weiterhin enthält der Integrand für das In-Scattering aus Gleichung 3.4 wieder die Renderinggleichung selbst. Für diese Integrale existiert im Allgemeinen keine geschlossene Lösung.

Im weiteren Kontext dieser Arbeit werden Radianzänderungen durch Absorbtion und Emission vernachlässigt ( $\mu_a(x) = 0$ ). Die besprochene Problematik bleibt weiterhin bestehen.

## 3.2 Volumetrisches Path Tracing

Im Folgenden Abschnitt wird gezeigt, wie die Integrale der Renderinggleichung geschätzt werden können und wie sich daraus ein volumetrischer Path Tracer ableiten lässt.

### 3.2.1 Schätzung der Distanz

Das äußere Integral der Renderinggleichung lässt sich schätzen, indem eine Distanz  $z$  zufällig gezogen wird und der Integrand an dieser Stelle ausgewertet wird. Dazu muss eine sinnvolle Wahrscheinlichkeitsdichtefunktion bestimmt werden. Wir betrachten zuerst das Schätzen der Distanz in einem homogenen Medium und motivieren dadurch anschließend einen Algorithmus zur Distanzschatzung in heterogenen Medien.

Zum Schätzen der Distanz in einem Medium kann die Transmittance verwendet werden. Dazu wird die Transmittance aus Gleichung 3.9 wie folgt umformuliert.

$$T(z) = e^{-\int_0^z \mu_t(x+\tilde{z}\omega) d\tilde{z}} = e^{-\tau(z)} \quad (3.11)$$

Wie bereits erwähnt, gibt die Transmittance den Anteil der Radianz an, der nach der Distanz  $z$  nicht ausgelöscht wurde. Somit entspricht die Transmittance  $T(z)$  der Wahrscheinlichkeit, dass ein Photon eine größere Distanz als  $z$  zurücklegt (Novák, Georgiev u. a., 2018). Folglich entspricht die Wahrscheinlichkeit, dass ein Photon spätestens nach der Distanz  $z$  ausgelöscht wird, dem Term  $1 - T(z)$ . Diese Wahrscheinlichkeit kann auch als CDF  $F(z)$  der Distanz, die ein Photon ohne Auslöschung zurücklegt, betrachtet werden. Gleichung 3.12 zeigt, wie sich die PDF  $p(z)$  zum Schätzen der Distanz aus der CDF ableiten lässt.

$$p(z) = \frac{dF(z)}{dz} = \frac{d}{dz} \left( 1 - e^{-\tau(z)} \right) = \frac{d\tau(z)}{dz} e^{-\tau(z)} = \mu_t(x + z\omega) e^{-\tau(z)} \quad (3.12)$$

Wie Gleichung 3.13 zeigt, kann das Integral der Transmittance in homogenen Medien analytisch berechnet werden, da der Auslöschungskoeffizient konstant ist ( $\mu_t(x) = \mu_t$ ). Mit dieser geschlossenen Form kann die CDF invertiert und für das Importance Sampling der Distanz verwendet werden (siehe Gleichung 3.14).

$$F(z) = 1 - e^{-\int_0^z \mu_t d\tilde{z}} = 1 - e^{-\mu_t z} \quad (3.13)$$

$$z(\xi) = F^{-1}(\xi) = -\frac{\ln(1-\xi)}{\mu_t} \quad (3.14)$$

Da die Transmittance im Allgemeinen in heterogenen Medien nicht analytisch bestimmt werden kann, muss eine andere Methode verwendet werden. Das Delta Tracking (Woodcock, 1965) erlaubt das effiziente Sampeln von Distanzen in heterogenen Medien. Dazu simuliert das Delta Tracking ein homogenes Medium, indem das heterogene Medium mit fiktiven Partikeln aufgefüllt wird. Interagiert ein Photon mit dem fiktiven Medium, wird das Photon nicht dadurch beeinflusst und fliegt ungehindert weiter. Das kombinierte Medium aus realen und fiktiven Partikeln hat einen homogenen Auslöschungskoeffizienten  $\tilde{\mu}_t$ .

$$\tilde{\mu}_t = \mu_t^{\text{real}}(x) + \mu_t^{\text{fiktiv}}(x) \quad (3.15)$$

Algorithmus 1 zeigt das Delta Tracking Verfahren. Dieses sampelt Distanzen in dem kombinierten Medium unter Verwendung von Gleichung 3.14. Für jedes Sample wird eine wahrscheinlichkeitsbasierte Fallunterscheidung durchgeführt. Mit der Wahrscheinlichkeit  $q$  (siehe Gleichung 3.16) handelt es sich um eine reale Interaktion zwischen Licht und Medium. In diesem Fall wird das Delta Tracking abgebrochen und die Distanz  $z$  als Ergebnis betrachtet. Mit der Gegenwahr-

### 3.2. VOLUMETRISCHES PATH TRACING

---

scheinlichkeit handelt es sich um eine Interaktion mit dem fiktiven Medium und das Photon fliegt weiter durch das Medium. Dieses Verfahren wird so lange wiederholt, bis es durch eine reale Interaktion abgebrochen wird.

$$q(x + z\omega) = \frac{\mu_t^{\text{real}}(x + z\omega)}{\tilde{\mu}_t} \quad (3.16)$$

---

**Algorithmus 1 :** Delta Tracking zum Schätzen der Distanz (Novák, Selle u. a., 2014)

---

```

1  $z = 0$ 
2 do
3    $\zeta = \text{uniform}(0, 1)$ 
4    $z = z - \frac{\ln(1-\zeta)}{\tilde{\mu}_t}$ 
5   if  $z \geq d$  then
6     | break
7   end
8    $\epsilon = \text{uniform}(0, 1)$ 
9   while  $\epsilon > \frac{\mu_t(o+z\omega)}{\tilde{\mu}_t};$ 
10  return  $z$ 
```

---

#### 3.2.2 Schätzung der Transmittance

---

**Algorithmus 2 :** Ratio Tracking zum Schätzen der Transmittance (Novák, Selle u. a., 2014)

---

```

1  $z = 0$ 
2  $T = 0$ 
3 do
4    $\zeta = \text{uniform}(0, 1)$ 
5    $z = z - \frac{\ln(1-\zeta)}{\tilde{\mu}_t}$ 
6   if  $z \geq d$  then
7     | break
8   end
9    $T = T(1 - \frac{\mu_t(o+z\omega)}{\tilde{\mu}_t})$ 
10  while true;
11  return  $T$ 
```

---

Sogenannte Track-length Schätzer nutzen die Interpretation der Transmittance  $T(z)$  als Wahrscheinlichkeit, dass ein Photon mindestens die Distanz  $z$  zurücklegt (Novák, Georgiev u. a., 2018). Dies erlaubt das Konstruieren eines Transmittanceschätzers basierend auf Distanzschätzern (Jarosz, Nowrouzezahrai u. a., 2011). Um so einen Schätzer zu motivieren, betrachten wir zunächst

### KAPITEL 3. GRUNDLAGEN

---

den einfachen Schätzer aus Gleichung 3.17 (Novák, Georgiev u. a., 2018). Dieser wertet die Transmittance nur mit einer Wahrscheinlichkeit  $P(„accept“)$  aus. Mit der Gegenwahrscheinlichkeit wird die Transmittance durch 0 geschätzt.

$$\langle T(z) \rangle_{RR} = \begin{cases} \frac{T(z)}{P(„accept“)}, & \text{if } „accept“ \\ 0, & \text{else} \end{cases} \quad (3.17)$$

Wie Gleichung 3.18 zeigt, ist der Erwartungswert dieses Schätzers gleich der Transmittance, wenn gilt:  $\forall z : T(z) > 0 \Rightarrow P(„accept“) > 0$  (Novák, Georgiev u. a., 2018).

$$E[\langle T(z) \rangle_{RR}] = P(„accept“) \cdot \frac{T(z)}{P(„accept“)} + P(\neg „accept“) \cdot 0 = T(z) \quad (3.18)$$

Sei die zurückgelegte Strecke  $z_i$  eines Photons als Zufallsvariable gegeben. Wie bereits erläutert, entspricht die Transmittance  $T(z)$  der Wahrscheinlichkeit  $P(z_i > z)$ . Wird „accept“ durch die Bedingung  $z_i > z$  ersetzt, entspricht die Wahrscheinlichkeit  $P(„accept“)$  der Transmittance, wodurch sich der Transmittanceterm in Gleichung 3.17 wegbürtzt (Novák, Georgiev u. a., 2018). Gleichung 3.19 zeigt den so entstehenden Track-length Schätzer  $\langle T(z) \rangle_{TL}$ . Die Bedingung  $z_i > z$  kann durch einen Distanzschätzer wie das Delta Tracking überprüft werden.

$$\langle T(z) \rangle_{TL} = \begin{cases} 1, & \text{if } z_i > z \\ 0, & \text{else} \end{cases} \quad (3.19)$$

Ein Track-length Schätzer mittels Delta Tracking erzeugt mehrere Interaktionen mit dem fiktiven Medium an den Stellen  $x_i = x + z_i \omega$ . An jeder Interaktion wird probabilistisch entschieden, ob das Verfahren durch eine reale Interaktion abgebrochen wird oder die nächste Interaktion gesampelt wird. Für jede Distanz  $z_i$  wird die Transmittance durch 1 geschätzt, bis das Tracking abgebrochen wird und die Transmittance durch 0 geschätzt wird. Novák, Selle u. a. (2014) beobachten, dass das Delta Tracking einen zufälligen Pfad auf dem Strahl konstruiert, der durch russisches Roulette terminiert wird. Das probabilistische Terminieren führt zu einer relativ hohen Varianz des Transmittanceschätzers (Novák, Georgiev u. a., 2018). Novák, Selle u. a. (2014) ersetzen die Terminierung durch russisches Roulette mit einer Gewichtung des Pfades, wodurch die Varianz in der geschätzten Transmittance reduziert werden kann. Der Pfad wird erst terminiert, wenn die Distanz  $z$  zurückgelegt wurde. Wie Gleichung 3.20 zeigt, berechnet sich dieses Gewicht als Produkt der Wahrscheinlichkeiten, dass ein Photon an den Stellen  $x_i$  nicht mit dem realen Medium interagiert. Novák, Selle u. a. (2014) nennen diesen Algorithmus Ratio Tracking.

$$\langle T(z) \rangle_{RT} = \prod_{i=1}^K \left( 1 - \frac{\mu_t^{real}(x_i)}{\tilde{\mu}_t} \right) \quad (3.20)$$

### 3.2.3 Schätzung der Richtung

Um das Integral aus Gleichung 3.4 zu schätzen, muss eine Richtung  $\omega_i$  gesampelt werden, gegeben die Richtung  $\omega$ . Dazu müssen zwei Winkel  $\theta$  und  $\phi$  berechnet werden. Wie bereits erläutert, ist  $\theta$  der Winkel zwischen den Richtungen  $\omega_i$  und  $\omega$ .  $\phi$  ist der Azimuthwinkel von  $\omega_i$  mit  $\omega$  als Bezugsrichtung. Wie in Gleichung 3.5 zu sehen ist, ist die Henyey-Greenstein Phasenfunktion invariant gegenüber dem Winkel  $\phi$ .  $\phi$  wird deshalb uniform aus dem Intervall  $[0, 2\pi]$  gezogen. Um  $\theta$  zu bestimmen, wird die Henyey-Greenstein Phasenfunktion aus Gleichung 3.5 als PDF mit  $\cos\theta$  als Argument betrachtet. Durch Invertierung der CDF ergibt sich Gleichung 3.21, mit der  $\cos\theta$  durch Importance Sampling gezogen werden kann (Zhang, 2019).

$$\cos\theta = \frac{1}{2g} \left( 1 + g^2 - \left( \frac{1 - g^2}{1 + g - 2g\zeta} \right)^2 \right) \quad \text{mit } \zeta = \text{uniform}(0, 1) \quad (3.21)$$

### 3.2.4 Next Event Estimation

Betrachtet man eine zu beleuchtende Stelle  $x$  auf einer Oberfläche oder als Punkt in einem Medium, kann es Lichtquellen geben, die nur einen kleinen Raumwinkel abdecken, aber einen großen Beitrag zur Beleuchtung an der Stelle  $x$  leisten (Dittebrandt u. a., 2020). Wird die Richtung ausschließlich nach der Phasenfunktion gesampelt und dieser Effekt nicht berücksichtigt, erzeugt der Path Tracer Ausgaben mit einer hohen Varianz. Um dem entgegen zu wirken, werden Lichtquellen direkt gesampelt. Dieses Verfahren wird „Next Event Estimation“ (NEE) genannt. Wie Abbildung 3.3 zeigt, wird für jede simulierte Interaktion mit dem Medium zusätzlich eine Lichtquelle gesampelt. Der Beleuchtungsbeitrag durch diese Lichtquelle wird mit einem Schattenstrahl überprüft, da Verdeckung durch Geometrie oder Auslöschung durch das Medium berücksichtigt werden müssen.

### 3.2.5 Path Tracing Algorithmus

Werden die besprochenen Komponenten zur effizienten Schätzung der Distanz, Transmittance und Richtung kombiniert, entsteht ein volumetrischer Path Tracer. Durch das abwechselnde Schätzen von Distanzen und Richtungen generiert der Path Tracer einen Pfad, der aus mehreren Vertices  $x_i$  besteht (siehe Abbildung 3.3). Der volumetrische Path Tracer berechnet den Beleuchtungsbeitrag dieses Pfades. Zusätzlich wird an jedem Vertex eine Next Event Estimation durchgeführt. Algorithmus 3 zeigt das volumetrische Path Tracing.

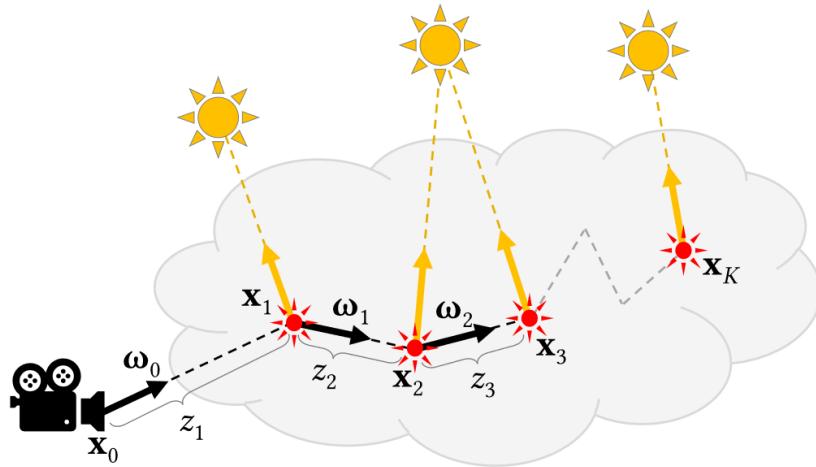


Abbildung 3.3: Volumetrisches Path Tracing eines Pfades mit Next Event Estimation (Lin u. a., 2021)

---

**Algorithmus 3 : Monte Carlo Path Tracing in einem Medium**

---

```

1  $l_{total} = [0, 0, 0]$ 
2 for  $i = 0$  to  $K$  do
3    $z_i = \text{DeltaTracking}(x_i, \omega_i)$ 
4    $x_{i+1} = x_i + z_i \omega_i$ 
5    $x_{NEE}, l_{NEE} = \text{SampleLightSource}()$ 
6    $z_{NEE} = \|x_{i+1} - x_{NEE}\|_2$ 
7    $\omega_{NEE} = (x_{i+1} - x_{NEE})/z_{NEE}$ 
8    $T_{NEE} = \text{RatioTracking}(x_{i+1}, -\omega_{NEE}, z_{NEE})$ 
9    $l_{total} = l_{total} + l_{NEE} \cdot \text{phase}_{hg}(-\omega_i, \omega_{NEE}) \cdot T_{NEE}$ 
10   $\omega_{i+1} = \text{ImportanceSampleHG}(\omega_i)$ 
11 end
12 return  $l_{total}$ 

```

---

### 3.3 Multilayer Perzepton

Künstliche neuronale Netze (NN) gehören zu den beliebtesten Methoden des maschinellen Lernens und können vielseitig eingesetzt werden. Das Multilayer Perzepton (MLP) ist eine von vielen Varianten der neuronalen Netze (Delashmit u. a., 2005). MLPs sind dazu in der Lage aus großen Datensätzen zu lernen und so verschiedenste Probleme zu lösen. Ziel dieser Arbeit ist die Anwendung von MLPs, um die Radianzverteilung in einem Medium zu approximieren. Dazu werden im folgenden Abschnitt die Struktur und Funktionsweise von MLPs erläutert. Zusätzlich werden Erweiterungen und Optimierungen vorgestellt, die das Lernen eines MLPs beschleunigen.

### 3.3.1 Aufbau

Ein MLP besteht aus mehreren Einheiten, die Neuronen genannt werden. In einem MLP sind Neuronen in Schichten (Layer) angeordnet. Die Eingabeschicht (Input Layer) erhält einen Vektor  $X \in \mathbb{R}^n$  als Eingabe und die Ausgabeschicht (Output Layer) gibt einen Vektor  $y \in \mathbb{R}^m$  aus. Somit implementiert ein MLP eine Abbildung  $f_{model} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . MLPs besitzen mindestens einen zusätzlichen Layer (Hidden Layer) zwischen der Eingabe- und Ausgabeschicht. Ein MLP wird in dieser Arbeit als vollständig verbunden (fully connected) betrachtet. Die Neuronen des Hidden und Output Layer in einem solchen MLP sind mit allen Neuronen der vorigen Schicht verbunden.

### 3.3.2 Inferenz

Die Inferenz bezeichnet im Kontext der neuronalen Netze das Verwenden des neuronalen Netzes, um eine Vorhersage zu treffen. Um diesen Prozess zu verstehen werden im Folgenden die grundlegenden Mechanismen eines MLP erläutert.

Betrachten wir nun eine Schicht mit  $k$  Neuronen und die vorige Schicht mit  $h$  Neuronen. Der Ausgabevektor der vorigen Schicht  $v \in \mathbb{R}^h$  sei gegeben. Gleichung 3.22 zeigt, wie die Ausgabe des  $i$ -ten Neurons der aktuellen Schicht berechnet wird. Das  $i$ -te Neuron besitzt die Gewichte  $w_{ij}$ , den Biaswert  $b_i$  und die Aktivierungsfunktion  $f_i$ .

$$z_i = f_i(b_i + \sum_{j=1}^h w_{ij} v_j) \quad (3.22)$$

Wie Gleichung 3.23 zeigt, lässt sich die Berechnung des Ausgabevektors  $z$  mittels Matrix-Vektor Schreibweise weiter vereinfachen. Die Matrix  $W \in \mathbb{R}^{h \times k}$  enthält die Gewichte  $w_{ij}$  und der Vektor  $B \in \mathbb{R}^k$  enthält die Biaswerte  $b_i$ .  $f_{activ}$  wendet die Aktivierungsfunktionen der einzelnen Neuronen auf die Komponenten des Vektors  $a \in \mathbb{R}^k$  an.

$$z = f_{activ}(a) \quad \text{mit } a = Wv + B \quad (3.23)$$

Die Funktionsweise eines MLPs wird maßgeblich von den verwendeten Aktivierungsfunktionen beeinflusst. Wie bereits erwähnt sind Aktivierungsfunktionen nicht-linear, wodurch ein MLP in der Lage ist, komplexere Funktionen zu implementieren. Abbildung 3.4 zeigt drei häufig verwendete Aktivierungsfunktionen, die Logistische Funktion, den Tangens Hyperbolicus ( $\tanh$ ) und das Rectified Linear Unit (ReLU). Innerhalb einer Schicht wird in der Regel dieselbe Aktivierungsfunktion verwendet. Oft nutzen alle Schichten dieselbe Aktivierungsfunktion mit Ausnahme der Ausgabeschicht.

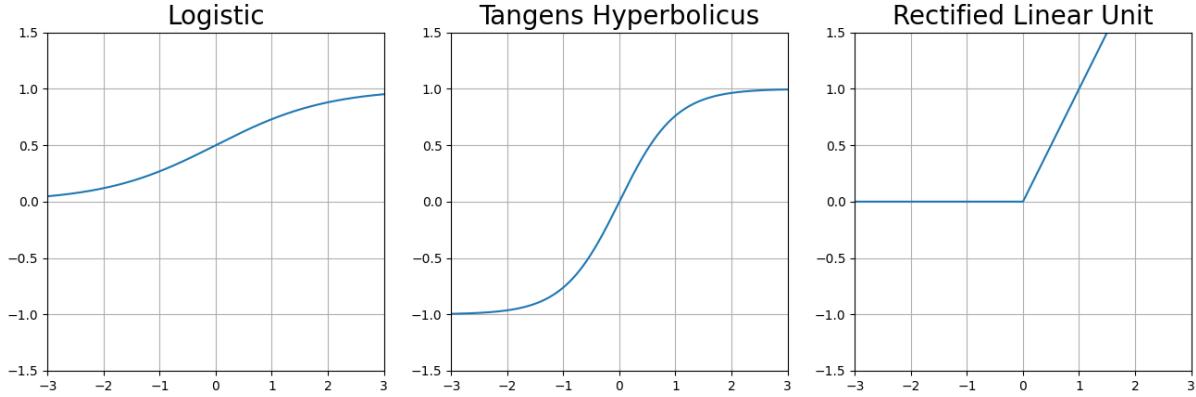


Abbildung 3.4: Aktivierungsfunktionen: Logistische Funktion, Tanh, ReLU

### 3.3.3 Training

Ziel der Anwendung von einem MLP ist das Approximieren einer Funktion  $f_{target} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . Wie gut ein MLP diese Funktion approximiert ist abhängig von der Struktur (Anzahl an Neuronen, Anzahl an Schichten, Wahl der Aktivierungsfunktionen) und den Parametern (Gewichte und Biaswerte). In der Regel ist die optimale Auswahl von Struktur und Parametern nicht bekannt. Während die Struktur meistens von dem Nutzer gewählt wird, können Gewichte und Biaswerte erlernt werden.

#### Gradientenabstieg

Sei eine Zielfunktion  $f_{target}$  und ein MLP mit gewählter Struktur gegeben. Sei weiterhin ein Trainingsdatensatz  $T$  gegeben mit  $N$  Elementen  $({}^p X, {}^p \hat{y}) \in T$ , so dass  $f_{target}({}^p X) = {}^p \hat{y}$ . Gegeben eine Eingabe  ${}^p X$  berechnet das MLP eine Prognose  ${}^p y$ . Sei  $E : \mathbb{R}^m \rightarrow \mathbb{R}$  eine Fehlerfunktion, die die Differenz einer Zielausgabe  ${}^p \hat{y}$  und einer Prognose  ${}^p y$  auf einen reellen Fehlerwert abbildet. Der Globale Fehler  $E_{global}$  (siehe Gleichung 3.25) ist ein Maß für Qualität der Approximation auf den Trainingsdaten  $T$ . Ein kleinerer globaler Fehler bedeutet eine bessere Approximation.

$${}^p E = E({}^p \hat{y} - {}^p y) \quad (3.24)$$

$$E_{global} = \frac{1}{N} \sum_{{}^p \in T} {}^p E \quad (3.25)$$

Ziel des Lernens ist es, den globalen Fehler unter Anpassung der Gewichte und Biaswerten zu minimieren. Ist die Fehlerfunktion nach den Parametern des MLPs differenzierbar, kann zur Minimierung das Gradientenabstiegsverfahren verwendet werden. Dazu wird jeder Parameter  $w$

### 3.3. MULTILAYER PERZEPTRON

---

um einen Wert  $\Delta w$  angepasst. Gleichung 3.26 zeigt, wie sich  $\Delta w$  aus der partiellen Ableitung des globalen Fehlers nach dem Parameter  $w$  und einem Faktor  $\eta \in (0, 1]$  berechnen lässt.  $\eta$  wird auch Learning Rate genannt und bestimmt die Schrittgröße in einem Gradientenabstiegsschritt. Das Gradientenabstiegsverfahren wird mehrfach wiederholt, bis der globale Fehler ausreichend gering ist.

$$\Delta w = -\eta \frac{\partial E_{global}}{\partial w} = -\eta \frac{1}{N} \sum_{\partial \in T} \frac{\partial^p E}{\partial w} \quad (3.26)$$

#### Stochastischer Gradientenabstieg

Enthält der Trainingsdatensatz sehr viele Elemente, ist das Gradientenabstiegsverfahren sehr teuer, da für alle Trainingsdaten die partielle Ableitung nach allen Parametern berechnet werden muss. Wie in Gleichung 3.27 zu sehen ist, ist der Erwartungswert eines Fehlers  ${}^p E$  mit einem uniform zufällig gezogenen  $p \in T$  gleich dem  $\Delta w$  des Gradientenabstiegsverfahrens.

$$\mathbf{E}\left[-\eta \frac{\partial^p E}{\partial w}\right] = -\eta \frac{1}{N} \sum_{p \in T} \frac{\partial^p E}{\partial w} = -\eta \frac{\partial E_{global}}{\partial w} = \Delta w \quad (3.27)$$

Der stochastische Gradientenabstieg (SGD) nutzt diesen Zusammenhang, um den globalen Fehler effizienter zu minimieren. Statt in jeder Iteration den globalen Fehler zu berechnen, wird ein Tupel  $({}^p X, {}^p \hat{y})$  aus den Trainingdaten gesampelt und der Fehler  ${}^p E$  für den Gradientenabstieg verwendet. Da SGD die Parameteranpassung  $\Delta w$  mit einem einzigen Sample aus den Trainingsdaten schätzt, ist die Varianz des geschätzten  $\Delta w$  relativ hoch. Diese Varianz kann die Anzahl der benötigten Gradientenschritte erhöhen. Das so genannte Mini Batch Learning zieht mehrere Tupel aus den Trainingsdaten, wodurch die Varianz des Schätzers verringert wird. Die Teilmenge der Trainingsdaten, die Mini Batch Learning in einem Gradientenschritt verwendet, wird Batch genannt.

Es existieren mehrere Erweiterungen des stochastischen Gradientenabstiegs, mit dem Ziel den Lernprozess zu beschleunigen. Zwei der am häufigsten verwendeten Erweiterungen sind SGD mit Momentum (SGDM) (Qian, 1999) und Adam (Kingma und Ba, 2014). Zweiteres wurde von Müller, Rousselle u. a. (2021) verwendet, um den Neural Radiance Cache zu trainieren.

#### Backpropagation of Error

Im Folgenden betrachten wir die Berechnung der partiellen Ableitungen des Fehlers  ${}^p E$  nach den Gewichten und Biaswerten eines MLP. Wie Gleichung 3.22 zeigt, ist die Ausgabe eines Hidden Layer lediglich von der Eingabe des vorigen Layer und den eigenen Gewichten und Biaswerten abhängig. Somit berechnet sich die Ausgabe eines MLP durch eine Verkettung von Funktionen.

Der Algorithmus Backpropagation of Error (Delashmit u. a., 2005) nutzt diesen Zusammenhang gemäß Kettenregel, um die partiellen Ableitungen effizienter zu berechnen. Backpropagation of Error speichert bereits berechnete partielle Ableitungen und spart so Laufzeitkosten. Gleichungen 3.28 und 3.29 zeigen, wie die partiellen Ableitungen des Fehlers nach den Biaswerten  $B$  und Gewichten  $W$  berechnet werden, wenn die partielle Ableitung des Fehlers  ${}^p E$  nach der Ausgabe  $z$  der Schicht gegeben ist. Gleichung 3.30 zeigt, wie die partielle Ableitung nach der Ausgabe  $v$  der vorigen Schicht berechnet wird.

$$\frac{\partial {}^p E}{\partial B} = \frac{\partial a}{\partial B} \cdot \frac{\partial z}{\partial a} \frac{\partial {}^p E}{\partial z} = \frac{\partial z}{\partial a} \frac{\partial {}^p E}{\partial z} \quad (3.28)$$

$$\frac{\partial {}^p E}{\partial W} = \frac{\partial a}{\partial W} \cdot \frac{\partial z}{\partial a} \frac{\partial {}^p E}{\partial z} = v \cdot \left( \frac{\partial z}{\partial a} \frac{\partial {}^p E}{\partial z} \right)^T \quad (3.29)$$

$$\frac{\partial {}^p E}{\partial v} = \frac{\partial a}{\partial v} \cdot \frac{\partial z}{\partial a} \frac{\partial {}^p E}{\partial z} = W^T \cdot \frac{\partial z}{\partial a} \frac{\partial {}^p E}{\partial z} \quad (3.30)$$

#### 3.3.4 Input Encoding

In vielen Anwendungen werden die Eingabevektoren  ${}^p X$  verändert, bevor sie dem MLP übergeben werden. Ein Beispiel ist das Normalisieren der Komponenten des Eingabevektors anhand von Mittelwert und Standardabweichung in den Trainingsdaten. Dadurch kann numerischen Problemen vorgebeugt werden. Wie Müller, Mcwilliams u. a. (2019) und Müller, Evans u. a. (2022) zeigen, können Veränderungen der Eingabe, insbesondere das Abbilden in einen höher-dimensionalen Raum, die benötigte Netzwerkgröße und Trainingszeit verringern. Solche Veränderungen der Eingabe werden Input Encoding genannt. Im Folgenden werden einige Input Encodings vorgestellt, die für die Verwendung zusammen mit einem Neural Radiance Cache in Frage kommen.

##### Positional Encoding

Das Positional Encoding projiziert jede Komponente  $x$  des Eingabevektors  ${}^p X$  in einen höher-dimensionalen Raum, wie in Gleichung 3.31 zu sehen ist. Mildenhall u. a. (2020) zeigen, dass das Positional Encoding die Qualität eines Neural Radiance Fields wesentlich verbessert.

$$f_{pos}(x) = (\sin(2^0 \pi x), \cos(2^0 \pi x), \dots, \sin(2^{L-1} \pi x), \cos(2^{L-1} \pi x)) \quad (3.31)$$

Müller, Rousselle u. a. (2021) approximieren das Positional Encoding mit dem Triangle Wave Encoding, um die Berechnungskosten zu minimieren. Wie in Gleichung 3.32 zu sehen ist, werden statt rechnerisch aufwändigen trigonometrischen Funktionen einfache Operationen verwendet.

### 3.3. MULTILAYER PERZEPPTRON

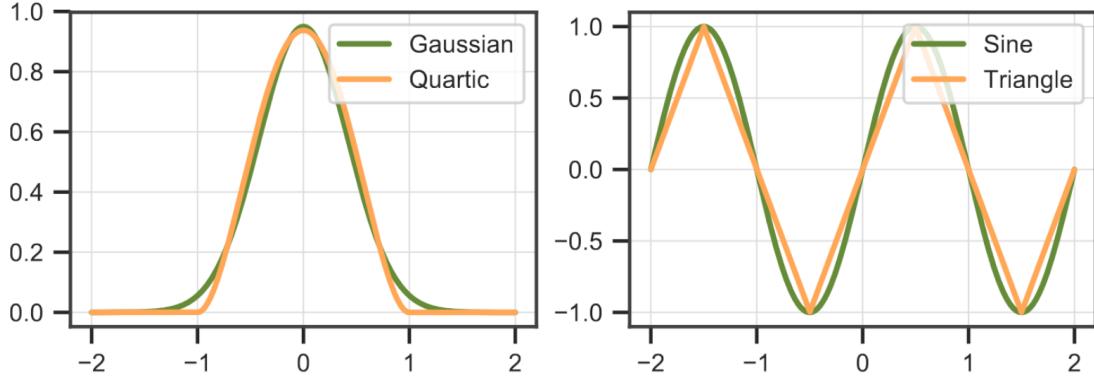


Abbildung 3.5: Approximation des One-Blob und Positional Encodings (Müller, Rousselle u. a., 2021)

$$\begin{aligned} \text{tri}(x) &= 2|x \bmod 2 - 1| - 1 \\ f_{\text{tri}}(x) &= (\text{tri}(2^0 x), \dots, \text{tri}(2^{L-1} x)) \end{aligned} \quad (3.32)$$

### One-Blob Encoding

Das bereits erwähnte One-Blob Encoding wird von Müller, Mcwilliams u. a. (2019) vorgestellt und neben dem Neural Importance Sampling auch erfolgreich auf das Neural Radiance Caching angewandt (Müller, Rousselle u. a., 2021). Für das One-Blob Encoding wird die Eingabe zunächst auf das Intervall  $[0, 1]$  normalisiert. Das One-Blob Encoding wertet an  $k$  gleichverteilten Stellen innerhalb dieses Intervalls eine Gaußsche Glockenfunktion mit Standardabweichung  $1/k$  und Mittelwert  $x$  aus. Das One-Blob Encoding realisiert so auch eine Abbildung ( $f_{\text{blob}} : \mathbb{R} \rightarrow \mathbb{R}^k$ ) einer einzelnen Eingabedimension in einen höher-dimensionalen Raum.

Wie auch für das Positional Encoding nutzen Müller, Rousselle u. a. (2021) eine optimierte Approximation des One-Blob Encodings für das Neural Radiance Caching. Wie Gleichung 3.33 zeigt, wird die Gaußsche Glockenfunktion mittels eines quartischen Polynoms angenähert. Dadurch wird die teure Auswertung der Exponentialfunktion eingespart. Mit diesen Annäherungen für das Positional und das One-Blob Encoding, die in Abbildung 3.5 zu sehen sind, können Müller, Rousselle u. a. (2021) pro berechnetem Bild 0.25 ms sparen.

$$quartic(x) = \frac{15}{16}(1 - x^2)^2 \quad (3.33)$$

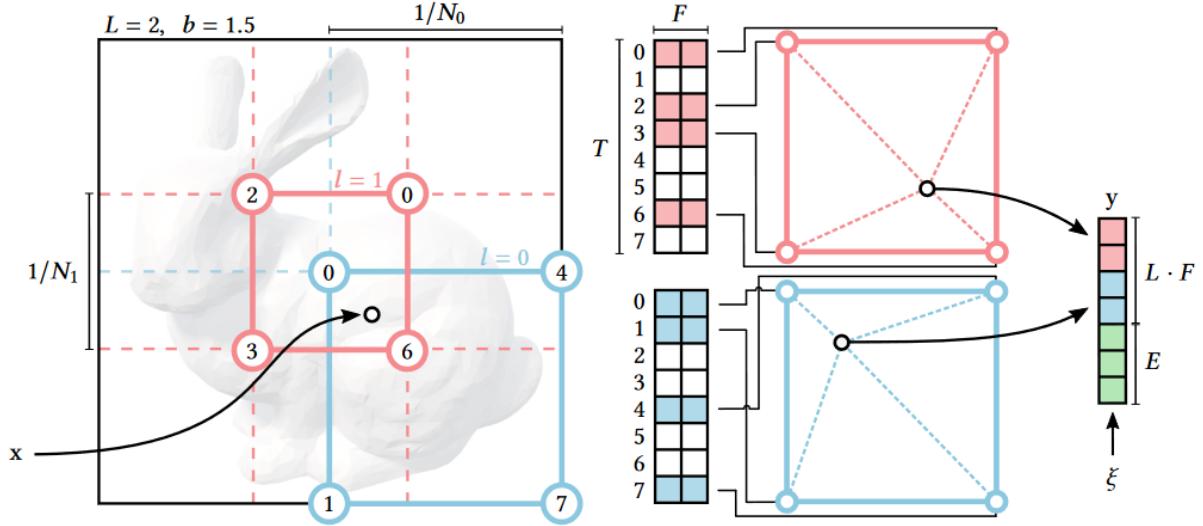


Abbildung 3.6: Multiresolution Hash Encoding (Müller, Evans u. a., 2022)

### Multiresolution Hash Encoding

Das von Müller, Evans u. a. (2022) vorgestellte Multiresolution Hash Encoding (MRHE) extrahiert aus einer Eingabe  ${}^P X$  mehrere sogenannte Features, die wie Gewichte und Biaswerte eines MLP mittels Gradient Descent trainiert werden können. Dabei wird der Eingabevektor in zwei Vektoren  $x$  und  $\xi$  unterteilt. Für das Extrahieren der Features wird nur  $x \in \mathbb{R}^d$  betrachtet.  $\xi \in \mathbb{R}^E$  wird später mit den berechneten Features konkateniert und bildet so das fertige Encoding  $y$ . Wie auch beim One-Blob Encoding wird angenommen, dass die Komponenten der Teileingabe  $x_i$  auf das Intervall  $[0, 1]$  normalisiert sind.

Wie in Abbildung 3.6 zu sehen ist, spannt das MRHE mehrere  $d$ -dimensionale Gitter in unterschiedlichen Auflösungen auf. Die Auflösung zwischen zwei Gittern unterscheidet sich um den Faktor  $b$ . Gleichung 3.34 zeigt, wie sich  $b$  durch die minimale und maximale Auflösungen  $N_{min}$  und  $N_{max}$  berechnen lässt. Der Parameter  $L$  gibt die Anzahl der Gitter an.

$$b = \exp\left(\frac{\ln N_{max} - \ln N_{min}}{L - 1}\right) \quad (3.34)$$

$$N_l = \lfloor N_{min} \cdot b^l \rfloor \quad (3.35)$$

Für jedes dieser Gitter wird ein Featurevektor mit  $F$  Komponenten aus der Teileingabe  $x$  berechnet. Im Folgenden wird erläutert, wie für ein Gitter mit Auflösung  $N_l$  der Featurevektor berechnet wird. Zunächst wird  $x$  mit der Auflösung des Gitters  $N_l$  skaliert. Durch Ab- und Aufrunden aller Komponenten entstehen zwei Vektoren  $\lceil x_l \rceil$  und  $\lfloor x_l \rfloor$  (siehe Gleichung 3.36), die zusammen

### 3.3. MULTILAYER PERZEPTRON

---

einen Hyperwürfel mit  $2^d$  Eckpunkten aufspannen.

$$\begin{aligned} \lceil x_l \rceil &= \lceil x \cdot N_l \rceil \\ \lfloor x_l \rfloor &= \lfloor x \cdot N_l \rfloor \end{aligned} \quad (3.36)$$

Das MRHE verwaltet für jedes Gitter eine Tabelle mit  $T$  Einträgen, die jeweils einen Featurevektor enthalten. Die Eckpunkte des Hyperwürfels werden jeweils auf einen Eintrag abgebildet. Für niedrige Auflösungen ist die Abbildung von Eckpunkten auf Einträge in der Tabelle injektiv. Ist die Auflösung hoch genug, existieren in einem Gitter mehr mögliche Gitterpunkte als Featurevektoren in der Tabelle. Dann wird eine Hashfunktion  $h$  verwendet, die einen Eckpunkt auf eine ganze Zahl abbildet, die als Index für die Tabelle genutzt wird. Wie Gleichung 3.37 zeigt, nutzt die Hashfunktion den XOR Operator, um die Komponenten eines Eckpunktes miteinander zu verknüpfen. Als Vorfaktoren  $\pi_i$  werden große Primzahlen verwendet.

$$h(x) = \left( \bigoplus_{i=1}^d \pi_i x_i \right) \bmod T \quad (3.37)$$

Die  $2^d$  Featurevektoren werden  $d$ -linear interpoliert, um den Featurevektor für dieses Gitter zu berechnen. Die Featurevektoren werden zusammen mit  $\xi$  konkatiniert und ergeben so das finale Encoding  $y \in \mathbb{R}^{L \cdot F + E}$ . Beim Training des MLP werden auch die  $2^d$  Featurevektoren der Eckpunkte mit einem gradientenbasierten Lernalgorithmus (z.B. SGD, SGDM oder Adam) optimiert.

# 4

# Methodik

---

## 4.1 Neural Radiance Cache

Der Neural Radiance Cache (NRC) hat das Ziel, die 5-dimensionale Radianzverteilung einer Szene effizient mit einem MLP zu approximieren (Müller, Rousselle u. a., 2021). Das Rendering mittels NRC erfolgt hybrid als Kombination von Path Tracer und MLP. Der Path Tracer terminiert frühzeitig an einem Pfadvertex  $x_i$ . Die Position des Vertices und die Richtung zum nächsten Vertex  $\omega_i$  werden dem NRC als Eingabe übergeben. Der NRC schätzt die Radianz des restlichen Subpfads. Die Ergebnisse des Path Tracers und des NRC werden kombiniert, um die finale Beleuchtung zu berechnen. Für eine Teilmenge der Bildschirmpixel werden  $x_i$  und  $\omega$  einem Path Tracer übergeben, dessen Ausgabe für das Training des NRC verwendet wird.

### 4.1.1 Aufbau

Der NRC nutzt ein MLP mit mehreren versteckten Schichten, die jeweils entweder 64 oder 128 Neuronen besitzen. Eine Besonderheit des NRC ist, dass dieser keine Biasvektoren verwendet. Nach Müller, Rousselle u. a. (2021) haben diese keinen Einfluss auf die Qualität der Ergebnisse und werden deshalb nicht in der Implementierung von Müller (2021) unterstützt. Die begrenzte Auswahl an Neuronen pro Schicht ist durch die Implementierung des MLP begründet, die in Sektion 4.3 näher erläutert wird. Die Anzahl der Neuronen der Eingabeschicht ist abhängig von dem gewählten Input Encoding. Die Ausgabe ist ein drei-dimensionaler Farbvektor. ReLU wird als Aktivierungsfunktion der versteckten Schichten verwendet. Die Ausgabeschicht nutzt keine Aktivierungsfunktion. Sowohl für das Training als auch die Inferenz wird eine Batchgröße zwischen  $2^{14}$  und  $2^{21}$  verwendet.

### 4.1.2 Training

Als Fehlerfunktion wird der Relative L2 Loss verwendet, der zusätzlich durch die Leuchtdichte normiert wird (Müller, Rousselle u. a., 2021), wie in Gleichung 4.1 zu sehen ist. Die Prognose des MLP wird im Nenner als Konstante betrachtet, wodurch die Gradienten nicht durch diesen

## 4.1. NEURAL RADIANCE CACHE

---

Term beeinflusst werden. Als Optimierer wird der Adam Algorithmus genutzt.

$${}^p E = \frac{\| {}^p \hat{y} - {}^p y \|_2^2}{\text{luminance}({}^p y)^2 + \epsilon} \quad \text{mit } \epsilon = 0.01 \quad (4.1)$$

$$\text{luminance}(x) = (0.299, 0.587, 0.114)^T \cdot x \quad (4.2)$$

### 4.1.3 Pfadterminierung

Für einen Pfad muss entschieden werden, an welchem Vertex der Pfad in den Neural Radiance Cache terminiert wird. In dieser Arbeit wird die Pfadterminierung durch zwei Parameter gesteuert. Es kann eine Mindestanzahl an Vertices pro Pfad festgelegt werden. Zusätzlich kann eine Wahrscheinlichkeit  $P(\text{continue})$  angegeben werden. Nachdem die Mindestanzahl an Vertices pro Pfad erreicht wurde, wird mit der Wahrscheinlichkeit  $P(\text{continue})$  ein nächster Vertex berechnet. Wie in Kapitel 5 dargelegt wird, hat die Wahl der Pfadterminierung einen wesentlichen Einfluss auf die Performanz, den Bias und die Varianz der Renderers.

### 4.1.4 Stabilität

Damit der NRC unter dynamischen Szenen funktionieren kann, muss dieser sich schnell anpassen können. Dazu wird eine relativ hohe Lernrate verwendet. Dies hat zur Folge, dass ein starkes Flimmern zu sehen ist. Müller, Rousselle u. a. (2021) begegnen diesem Problem, indem ein exponentiell gleitendes Mittel (EMA) der Netzwerkparameter  $\bar{W}$  während der Inferenz verwendet wird. Der Parameter  $\alpha \in [0, 1)$  steuert die Stärke des EMA. Wie in Gleichung 4.3 zu sehen ist, wird während des  $k$ -ten Trainingsschritt ein zusätzlicher Parameter  $\mu_k$  berechnet.  $\mu_k$  dämpft den EMA zu Beginn des Trainings. Dadurch werden die Netzwerkparameter (für die Inferenz) trotz EMA am Anfang schnell angepasst.

$$\bar{W}_{k+1} = \frac{1 - \alpha}{\mu_k} W_{k+1} + \alpha \mu_k \bar{W}_k \quad \text{mit } \mu_k = 1 - \alpha^k \quad (4.3)$$

### 4.1.5 Input Encoding

Wie bereits erläutert, können gut gewählte Input Encodings wesentlich dazu beitragen, die benötigten Trainingsiterationen und die Netzwerkomplexität zu minimieren. Im Fall einer Radianzverteilung können kleine Änderungen in der Position eine große Änderung der Radianz nach sich ziehen (Müller, Rousselle u. a., 2021). Ein Encoding für die Position sollte deshalb in der Lage sein, auch kleine Änderungen im Eingaberaum angemessen zu repräsentieren. Aus diesem Grund werden für das Encoding der Position vor allem das Positional Encoding (Müller,

Rousselle u. a., 2021) und das MRHE (Müller, Evans u. a., 2022) betrachtet. Wie bereits erwähnt, schlagen Müller, Rousselle u. a. (2021) das One-Blob Encoding (mit insgesamt 8 Dimensionen) für die Richtung  $\omega$  vor. Um numerischen Problemen beim Training vorzubeugen, werden die Komponenten  $c_i$  der Zielradianz auf das Intervall  $[0, c_{max}]$  begrenzt.

## 4.2 Rendering Pipeline

In diesem Abschnitt wird der Aufbau des volumetrischen Echtzeitrenderers mit Neural Radiance Cache betrachtet und auf die einzelnen Bestandteile und deren Aufgaben eingegangen. Darüber hinaus werden der „Infer-Filter-Buffer“ und der „Train-Ringbuffer“ präsentiert, mit dem Ziel, Rechenleistung einzusparen und effizienter einzusetzen.

### 4.2.1 Generate Rays

Der „Generate-Rays“ Pass erzeugt für alle Pixel auf dem Bildschirm die Pfade, die für den Path Tracer benötigt werden. Der Path Tracer folgt diesen Pfaden und berechnet das Beleuchtungsmodell. Er bricht frühzeitig ab und speichert den folgenden Subpfad, um diesen später durch den NRC berechnen zu lassen. Das Abbrechen des Pfades erfolgt nach der in Sektion 4.1.3 besprochenen Pfadterminierung des Neural Radiance Cache. Falls an einem Pixel kein Streuereignis stattfand, wird diese Information für die folgenden Komponenten gespeichert.

### 4.2.2 Prepare Infer Rays

In dem „Prepare-Infer-Rays“ Pass werden die Subpfade für die Inferenz vorbereitet. Die Subpfade bestehen aus je einem dreidimensionalen Positions- und Richtungsvektor. Die Komponenten des Positionsvektors werden auf das Intervall  $[-1, 1]$  normiert und der Richtungsvektor wird auf eine zweidimensionale Darstellung mittels sphärischer Koordinaten reduziert. Diese Repräsentation des Subpfades wird in einen Buffer gespeichert, der später durch das MLP ausgelesen werden kann.

Es ist möglich, dass das Volumen nicht auf den gesamten Bildschirm projiziert wird oder dass einzelne Kamerapfade innerhalb des Volumens kein Scatteringevent auslösen. In diesem Fall muss die Beleuchtung dieser Pixel nicht durch das MLP berechnet werden. Um solche unnötigen Berechnungen zu vermeiden, werden diese Pfade nicht dem MLP übergeben. Da Pfade nicht einzeln, sondern in größeren Batches durch das MLP berechnet werden, wird für jedes Batch überprüft, ob dieses einen Pfad enthält, der durch ein Scatteringevent erzeugt wurde. Enthält ein Batch mindestens einen solchen Pfad, wird diese Information in dem Infer-Filter-Buffer gespeichert und später beim Starten der Inferenz ausgelesen. Die Anzahl an eingesparten unnötigen Berechnungen ist abhängig von der Größe der Batches. Durch kleinere Batchgrößen können mehr Pfade übersprungen werden. Zu kleine Batchgrößen führen zu langer Inferenzzeit

### 4.3. IMPLEMENTIERUNG

---

durch das MLP.

#### 4.2.3 Prepare Train Rays

Der „Prepare-Train-Rays“ Pass bereitet die Subpfade, die für das Training benutzt werden, für den NRC vor. Wie in dem Prepare-Infer-Rays Pass wird die sechsdimensionale Darstellung in eine fünfdimensionale konvertiert, die dem MLP übergeben wird. Zusätzlich wird auf den Subpfaden ein Monte Carlo Path Tracer ausgeführt, der die Trainingsdaten generiert.

Auch die Subpfade, die für das Training verwendet werden, können ungültig sein. Eine Möglichkeit, dem zu entgegnen, ist die Methode, die in dem Prepare-Infer-Rays Pass angewandt wird. Da wesentlich weniger Trainingspfade existieren, sind die Trainingsbatches sehr weit über das Bild verteilt. Somit ist es unwahrscheinlicher, dass ein Trainingbatch existiert, das keinen gültigen Trainingspfad enthält. Stattdessen werden gültige Subpfade in dem Train-Ringbuffer gespeichert. Wird ein ungültiger Trainingspfad gefunden, wird dieser mit einem Subpfad aus dem Train-Ringbuffer ersetzt.

#### 4.2.4 Inferenz und Training

Nachdem die Pfade für die Inferenz und das Training vorbereitet wurden, werden diese durch den NRC ausgelesen und verarbeitet. Zuerst wird die Inferenz ausgeführt. Dabei werden die Inferenzbatches nacheinander dem MLP übergeben und die Ergebnisse in einen Buffer geschrieben, der später beim finalen Rendern ausgelesen wird. Daraufhin werden die Trainingsbatches nacheinander genutzt, um die Parameter des NRCs anzupassen.

#### 4.2.5 Rendering

In dem letzten Pass, dem „Rendering“ Pass, des Echtzeitrenderers wird die Ausgabe des NRCs mit dem Ergebnis des frühzeitig abgebrochenen Path Tracers kombiniert, um das finale Bild zu erzeugen. In diesem Pass besteht auch die Möglichkeit, die Ausgabe des Echtzeitrenderers mit anderen Szenenelementen wie Geometrie zusammenzufügen.

### 4.3 Implementierung

#### 4.3.1 Tiny CUDA Neural Networks

Die „Tiny CUDA Neural Networks“(TCNN) Bibliothek von Müller (2021) wird als Implementierung für die Inferenz und das Lernen von neuronalen Netzen verwendet. TCNN nutzt die Programmierschnittstelle CUDA des Hardwareherstellers Nvidia, um MLP-spezifische Operatio-

nen auf Grafikkarten zu beschleunigen. Neben den Operationen für neuronale Netze werden auch sämtliche Input Encodings, die in dieser Arbeit verwendet werden, durch TCNN implementiert.

### 4.3.2 Fully Fused Neural Networks

Wie Müller, Rousselle u. a. (2021) feststellen, können die langen Zugriffszeiten auf den Hauptspeicher der Grafikkarte (VRAM) die Geschwindigkeit einer MLP Implementierung negativ beeinflussen. Müller (2021) umgeht dieses Problem, indem sämtliche die für die Matrixoperationen eines MLP benötigten Daten, in dem so genannten Shared Memory verwaltet werden. Auf Grafikkarten des Herstellers Nvidia wird mit Shared Memory ein Speicher bezeichnet, der wesentlich kürzere Zugriffszeiten bei wesentlich weniger Speicherplatz bietet als der VRAM. Da der Shared Memory begrenzt ist, werden zu große Batches in kleinere Batches unterteilt und parallel bearbeitet. Müller (2021) nennt diese Implementierung „Fully Fused Neural Networks“. Die Speicherplatzlimitierung des Shared Memory führt auch zu einer Limitierung der Neuronen pro Schicht. Die TCNN Bibliothek unterstützt Schichten mit 16, 32, 64 und 128 Neuronen.

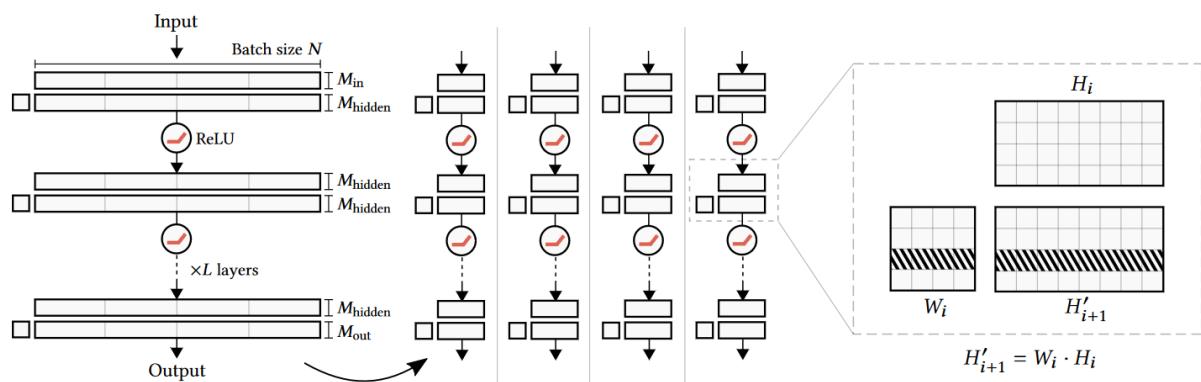


Abbildung 4.1: Fully Fused MLP mit 64 Neuronen pro Schicht (Müller, Rousselle u. a., 2021)

### 4.3.3 Tensor Cores

Zusätzlich zu der Optimierung durch kürzere Speicherzugriffe nutzt die Fully Fused Implementierung Nvidias Tensor Cores (Müller, Rousselle u. a., 2021). Mit Tensor Cores können Matrixmultiplikationen hardwarebasiert beschleunigt werden. Dazu werden Matrizen in  $16 \times 16$  große Blöcke unterteilt, die von den Tensor Cores bearbeitet werden. Die Gleitkommazahlen werden bei Tensor Core Operationen mit 16 bit statt 32 bit dargestellt.

### 4.3.4 Vulkan-CUDA-Interop

Die Rendering Pipeline wurde basierend auf der Vulkan Programmierschnittstelle in C++ programmiert. Die Generate-Rays, Prepare-Infer-Rays, Prepare-Train-Rays und Rendering Komponenten der Pipeline werden jeweils in einem Vulkan Compute Shader umgesetzt. Da die

### 4.3. IMPLEMENTIERUNG

---

Inferenz- und Trainingskomponente der Pipeline in CUDA implementiert ist, müssen die Daten und der zeitliche Ablauf zwischen Vulkan und CUDA synchronisiert werden. Sowohl Vulkan als auch CUDA bieten Schnittstellen an, die den Zugriff auf Daten und die zeitliche Synchronisation zwischen den zwei Programmierschnittstellen ermöglichen.

Die Buffer, in denen die Position und Richtung der Strahlen für die Inferenz und die Zielradianzen für das Training gespeichert sind, werden von Vulkan auf der Grafikkarte allokiert. CUDA erhält einen Verweis, über den der Zugriff auf diese Buffer möglich ist. Dieser Mechanismus benötigt insbesondere keine teuren Datentransfers zwischen Grafikkarte und Prozessor. Für die zeitliche Synchronisation stellt Vulkan Semaphoren bereit, auf die CUDA zugreifen kann. Durch die Semaphoren wird keine Intervention des Prozessors benötigt, um die korrekte Reihenfolge der Komponenten sicherzustellen.

Die einzige Ausnahme ist der Infer-Filter-Buffer, da die Inferenz der einzelnen Batches von dem Prozessor aus gestartet wird. Da der Infer-Filter-Buffer relativ klein ist, entstehen durch die Datentransfers keine großen Kosten. Jedoch muss der Datentransfer synchronisiert werden. Deshalb erfordert das Nutzen des Infer-Filter-Buffers eine Interaktion zwischen Grafikkarte und Prozessor, die in der Vulkan Programmierschnittstelle mit einer sogenannten Fence umgesetzt wird.

# 5

# Evaluation

---

## 5.1 Versuchsaufbau

Der Neural Radiance Cache wurde in einer Auflösung von  $1920 \times 1080$  Pixel auf einem Windows 11 System mit einem AMD Ryzen 7 3700x Prozessor und einer Nvidia RTX 3070 Grafikkarte getestet. Die Dichteverteilung des Mediums wird aus dem Disney Cloud Datensatz geladen.



Abbildung 5.1: Szene 0



Abbildung 5.2: Szene 1

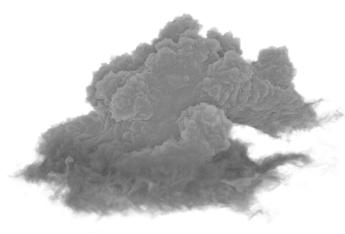


Abbildung 5.3: Szene 2

Abbildungen 5.1, 5.2 und 5.3 zeigen die drei verwendeten Testszenen. Diese unterscheiden sich in der Art der Beleuchtung. Szene 0 nutzt ein gerichtetes Licht, Szene 1 nutzt eine Punktlichtquelle in der Mitte des Mediums und Szene 2 nutzt eine uniforme weiße Environment Map als Beleuchtungsquelle. In den vorliegenden Abbildungen ist die Disney Cloud mit einem maximalen Streuungskoeffizienten von 0.6 und der Henyey-Greenstein Phasenfunktion mit  $g = 0.8$  zu sehen.

## 5.2 Laufzeitkosten

Um die Performanz des NRC Renderers zu messen, werden an mehreren Stellen in der Rendering Pipeline Zeitmessungen durchgeführt. Diese Zeitmessungen lassen sich mit sogenannten Timestamp Queries der Vulkan Programmierschnittstelle ohne Intervention des Hostsystems umsetzen.

## 5.2. LAUFZEITKOSTEN

---

Generate Rays Laufzeit in ms								
$P(\text{continue})$	Szene 0				Szene 1			
	1	2	3	4	1	2	3	4
0.0	5.63	14.36	23.89	35.04	2.14	3.61	5.72	8.63
0.25	10.12	18.49	27.51	36.17	3.38	5.03	7.31	9.93
0.5	17.28	25.03	34.9	42.78	5.54	7.79	10.21	13.22
0.75	36.07	42.23	50.16	55.83	11.78	13.77	16.57	18.76

Abbildung 5.4: Laufzeit des Generate Rays Shader

Laufzeit von Inferenz und Training in ms										
	T2 / OB4		T12 / OB4		F12 / OB4		MRHE8 / OB4		MRHE12 / OB4	
Hidden	64	128	64	128	64	128	64	128	64	128
3	4.95	7.3	5.87	8.46	7.02	9.49	8.67	11.05	11.37	14.56
4	5.93	8.74	6.53	10.52	7.83	11.8	9.51	12.87	12.61	16.15
5	6.73	10.81	7.55	12.14	10.64	14.31	10.64	14.31	12.94	17.08
6	7.51	12.36	8.16	13.77	11.36	16.47	11.36	16.47	13.68	18.96

Abbildung 5.5: Laufzeit von Inferenz und Training mit verschiedenen Input Encodings

### 5.2.1 Rendering Pipeline

Die Laufzeit des Generate-Rays Compute Shaders ist hauptsächlich von der Anzahl an simulierten Streuereignissen und der Szenenbeleuchtung abhängig. Wie in Sektion 4.1.3 erläutert, wird die Anzahl an simulierten Streuereignissen durch eine feste Mindestanzahl und durch eine Wahrscheinlichkeit  $P(\text{continue})$  bestimmt. Abbildung 5.4 zeigt, die Laufzeit in Abhängigkeit dieser zwei Parameter; zusätzlich werden Szene 0 und Szene 1 verglichen.

Die Laufzeit des Prepare-Infer-Rays Compute Shaders liegt bei maximal 0.59 ms. Die Überprüfung der Inferenzbatches wirkt sich nicht auf die Laufzeit aus.

Die Laufzeit des Prepare-Train-Rays Compute Shaders ist linear in der Anzahl an Trainingssamples. Für  $2^{16}$  Trainingssamples liegt die Laufzeit bei 0.36 ms, wobei der Ring Buffer auch  $2^{16}$  Elemente enthält.

Der Rendering Compute Shader benötigt ca. 0.3 ms.

### 5.2.2 Inferenz und Training

Abbildung 5.5 zeigt die Dauer der Inferenz und des Trainings in Millisekunden unter verschiedenen Input Encodings. Alle Encodings wurden in Netzwerken mit entweder 64 oder 128 Neuronen pro Schicht und mit drei bis sechs Hidden Layern getestet. In diesem Versuch wurde eine Inferenzbatch der Größe  $2^{21}$  und vier Trainingsbatches der Größe  $2^{14}$  verwendet. Die Kamera

## KAPITEL 5. EVALUATION

---

Laufzeit des Trainings in ms						
log2 Batch Größe	1	2	3	4	5	6
14	1.25	2.43	3.37	4.11	5.15	6.14
15	1.32	2.51	3.55	4.63	6.6	7.12
16	1.94	3.05	4.75	6.64	8.06	9.54
17	2.52	4.68	6.81	9.36	11.41	13.85

Abbildung 5.6: Laufzeit des Training mit „T12 / OB4“-Encoding und Netzwerkkonfiguration ( $6 \times 64$ )

wurde so positioniert, dass das Medium auf dem gesamten Bild zu sehen ist.

Die Encodings wurden dabei wie folgt abgekürzt: TN → Triangle Wave Encoding mit  $N$  Frequenzen. OBN → One-Blob Encoding mit  $N$  Bins. FN → Positional Encoding mit  $N$  Frequenzen. MRHEN → Multiresolution Hash Encoding mit  $N$  Auflösungen. Da das Identitäts-Encoding unregelmäßig zu numerischen Problemen führte, wird dieses nicht weiter betrachtet. Nach Mildenhall u. a. (2020) sollten andere Encodings wie das Positional Encoding oder das Triangle Wave Encoding ohnehin besser für die vorliegende Aufgabe geeignet sein. Aus diesem Grund wird das „T2 / OB4“-Encoding als Maßstab für die geringsten Laufzeitkosten verwendet. Aus dem gleichen Grund wurde lediglich das One-Blob Encoding für die Richtung betrachtet.

Erwartungsgemäß steigen die Kosten in allen Input Encodings mit der Anzahl an Frequenzen. Dieser Effekt ist besonders stark bei dem Multiresolution Hash Encoding zu sehen, was vermutlich durch die komplexen Operationen und das zusätzliche Gradientenabstiegsverfahren verursacht wird. Wie von Müller, Rousselle u. a. (2021) berichtet, verursacht das Positional Encoding (nur mit Sinustermen) höhere Kosten als das Triangle Wave Encoding (durchschnittlicher Unterschied von 1.99 ms) bei gleicher Anzahl an Frequenzen. Das „T2 / OB16“-Encoding verursacht in der kleinsten Netzwerkkonfiguration ( $3 \times 64$ ) 0.47 ms und in der größten Netzwerkkonfiguration ( $6 \times 128$ ) 0.96 ms mehr Kosten.

## 5.3 Qualität

### 5.3.1 Stabilität

Durch die hohe Lernrate von 0.01 entsteht ein sehr starkes Flimmern. Dieses kann durch den EMA der Netzwerkparameter gedämpft werden. Die Ergebnisse dieser Arbeit decken sich mit denen von Müller, Rousselle u. a. (2021). Um das Flimmern zu dämpfen, sollte  $\alpha$  auf den Wert 0.99 gesetzt werden.

### 5.3. QUALITÄT

---



Abbildung 5.7: Farbliche Abweichung des Triangle Wave Encoding (links), Positional Encoding (mitte) und Multiresolution Hash Encoding (rechts) kurz nach Start

#### 5.3.2 Netzwerkstruktur und Encoding

Sowohl das Triangle Wave Encoding als auch das Positional Encoding (mit 12 Frequenzen) zeigen eine starke farbliche Abweichung, die auch nach mehreren Sekunden noch zu sehen ist. Wie in Abbildung 5.7 zu sehen ist, ist die farbliche Abweichung bei dem Triangle Wave Encoding wesentlich höher. Das Multiresolution Hash Encoding zeigt zu Beginn auch eine farbliche Abweichung, die aber im Gegensatz zu dem Positional und Triangle Wave Encoding schnell verschwindet. Zusätzlich sind teilweise die von Müller, Rousselle u. a. (2021) beschriebenen achsenparallelen Artefakte bei dem Triangle Wave und dem Positional Encoding leicht sichtbar.

Bei dem Triangle Wave und dem Positional Encoding kommt es unabhängig von der Netzwerkstruktur zu einem Vergessen von Radianzen. Dies wird besonders deutlich, wenn die Kamera für eine kurze Dauer auf helle Regionen und danach auf eine dunkle Bildregion gerichtet wird. Dieser Effekt ist in einer abgeschwächten Form auch bei dem Multiresolution Hash Encoding zu erkennen.

Bei einer Netzwerkstruktur mit 6 Hidden Layern und 64 Neuronen pro Schicht zeigen das Positional und Triangle Wave Encoding einen höheren Bias als das Multiresolution Hash Encoding. Die relative Varianz ist bei allen Input Encodings ähnlich. Gleiches gilt für ein MLP mit 3 Hidden Layern und 64 Neuronen pro Schicht.

#### 5.3.3 Vergleich zu Path Tracer

Für den Vergleich zwischen Neural Radiance Cache und Path Tracer wurde ein NRC mit 6 Hidden Layern und 64 Neuronen pro Schicht verwendet. Als Input Encoding wird das Multiresolution Hash Encoding mit 12 Auflösungen für die Position und das One-Blob Encoding mit 4 Bins für die Richtung verwendet. Die Lernrate beträgt 0.01 und  $\alpha$  beträgt 0.99. Der NRC wird mit 4 jeweils  $2^{14}$  großen Trainingsbatches trainiert. Der Path Tracer beleuchtet Pfade mit 32 Streuereignissen.

### Beispiel 1

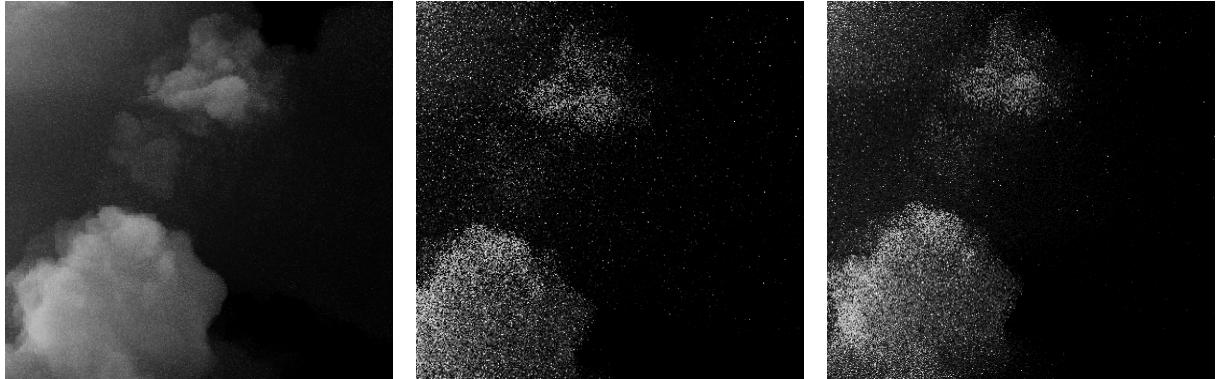


Abbildung 5.8: Ausschnitt aus Szene 0.6. Path Tracer mit 1024 SPP (links). Path Tracer mit 1 SPP (mitte; MSE = 0.403; rBias  $\in [-0.01, 0.01]$ ; rVar = 3.09). NRC mit 1 SPP (rechts; MSE = 0.311; rBias = -0.076; rVar = 2.38)

Beispiel 1 vergleicht den Neural Radiance Cache mit einem Path Tracer in Szene 0 mit einem mittleren Dichtemultiplikator von 0.6 (siehe Abbildung 5.8). Der Neural Radiance Cache terminiert den Pfad nach mindestens einem Streuereignis und danach mit einer Wahrscheinlichkeit von  $P(\text{continue}) = 0.5$ . Es ist deutlich zu erkennen, dass der NRC bei einem Sample pro Pixel (SPP) wesentlich weniger Rauschen zeigt als der Path Tracer. Dies spiegelt sich in der niedrigeren Varianz wieder. Vergleicht man die Ergebnisse des NRC mit dem Path Tracer bei 1024 SPP, so fällt auf, dass der NRC die Beleuchtung in besonders dunklen Regionen unterschätzt (negativer Bias).

### Beispiel 2

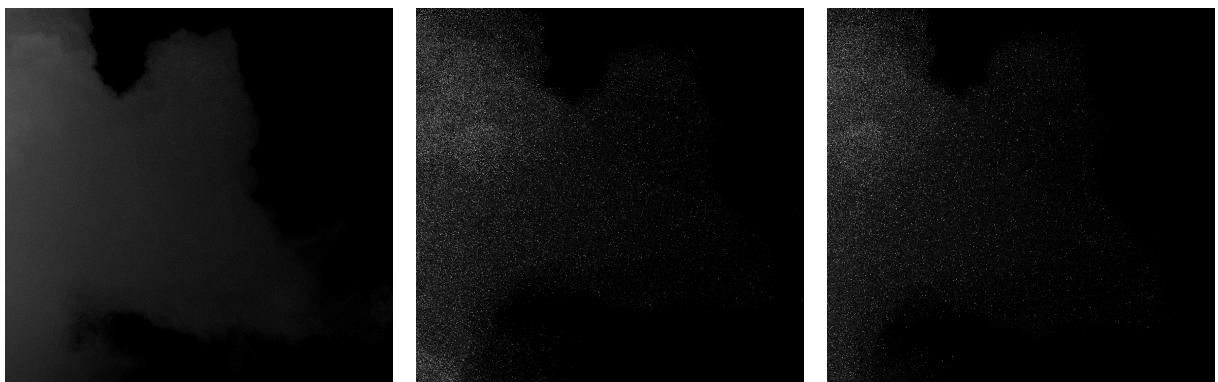


Abbildung 5.9: Ausschnitt aus Szene 0 (Dichte 0.25). Path Tracer mit 1024 SPP (links). Path Tracer mit 1 SPP (mitte; MSE = 0.067; rBias  $\in [-0.01, 0.01]$ ; rVar = 1.02). NRC mit 1 SPP (rechts; MSE = 0.071; rBias = -0.048; rVar = 1.078)

Beispiel 2 untersucht die Ergebnisse eines Path Tracers und eines NRC in Szene 0 mit einem

### 5.3. QUALITÄT

---

niedrigen Dichtemultiplikator von 0.25 (siehe Abbildung 5.9). Der NRC und der Path Tracer unterscheiden sich kaum in ihrer Qualität.

#### Beispiel 3

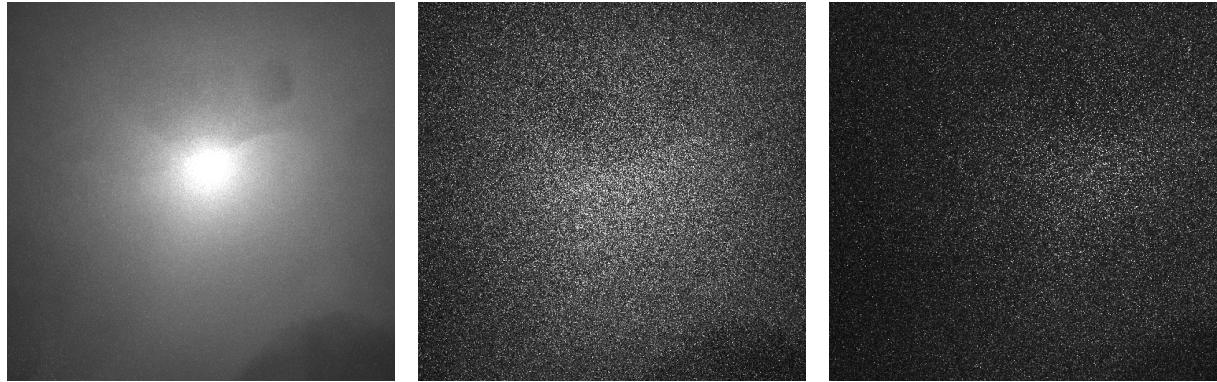


Abbildung 5.10: Ausschnitt aus Szene 1 (Dichte 0.6). Path Tracer mit 1024 SPP (links). Path Tracer mit 1 SPP (mitte; MSE = 0.117; rBias  $\in [-0.01, 0.01]$ ; rVar = 5.38). NRC mit 1 SPP (rechts; MSE = 0.071; rBias = -0.048; rVar = 1.078)

Beispiel 3 zeigt Szene 1 mit einem Dichtemultiplikator von 0.6. Beispiel 3 führt zu ähnlichen Ergebnisse wie Beispiel 1. Der NRC zeigt ein geringeres Rauschen und einen leichten negativen Bias. Der Fehler beider Renderer ist nahezu gleich.

# 6

## Fazit

---

In dieser Arbeit wurde das von Müller, Rousselle u. a. (2021) entwickelte Neural Radiance Caching in Heterogeneous Participating Media angewandt und unter verschiedenen Bedingungen getestet. Im Vergleich zu einem volumetrischen Path Tracer erzielt das Neural Radiance Caching rauschfreiere Ergebnisse bei teils höheren Bildraten von mehr als 30 Bildern pro Sekunde. Dieser Vorteil ist stärker bei höheren Dichten zu erkennen. Zusätzlich wurde der Einfluss verschiedener Parameter auf den Neural Radiance Cache untersucht. Dabei fällt vor allem das Multiresolution Hash Encoding auf, welches trotz höherer Kosten, durch schnelleres Lernen, weniger Artefakte und geringere Netzwerkanforderungen überzeugen kann.

Da der Neural Radiance Cache in dieser Arbeit ausschließlich für volumetrische Daten verwendet wurde, ist auf einer Teilmenge der Bildschirmpixel kein Medium sichtbar. Deshalb wurden der Infer-Filter-Buffer und der Train-Ringbuffer als Erweiterungen vorgeschlagen. Der Infer-Filter-Buffer ermöglicht die Einsparung von nicht benötigten Inferenzschritten ohne viel zusätzliche Laufzeit in Anspruch zu nehmen. Eine Möglichkeit, die Effizienz dieses Filters bei gleicher Batchgröße zu erhöhen, wäre eine bessere Verteilung der Inferenzbatches auf dem Bildschirm. Durch den Train-Ringbuffer kann die zur Verfügung stehende Rechenleistung effizienter genutzt werden. Je nach Position der Kamera kann der Neural Radiance Cache durch den Ringbuffer sichtbar schneller lernen.

Eine interessante Erweiterungsmöglichkeit ist ein besserer Train-Ringbuffer. Durch besseres Auswählen der dort gespeicherten Trainingspfade könnte zum Beispiel dem Problem des Vergessens von früheren Radianzen begegnet werden. Dafür könnten Pfade, die eine wesentlich andere Position und Richtung vorweisen als die Kamera, länger in dem Buffer erhalten bleiben, damit auch andere Regionen in dem Volumen ständig trainiert werden. Alternativ können für denselben Zweck, zusätzliche Trainingbatches verwaltet werden, die unabhängig von der Kamera ausgesuchte Pfade zum Trainieren verwenden.

In Bildsegmenten, die hauptsächlich durch den Neural Radiance Cache und nicht durch den Path Tracer beleuchtet werden, entsteht ein großer Anteil des Rauschens durch die Pfadauswahl des NRC. Müller, Rousselle u. a. (2021) schlagen hier besseres Path Guiding durch das Neural Importance Sampling von Müller, Mcwilliams u. a. (2019) vor. Eine alternative Lösung ist

---

das Kombinieren des Neural Radiance Caches mit dem von Lin u. a. (2021) vorgestellten volumetrischen ReSTIR Algorithmus. Um Kosten zu sparen, wäre vorstellbar, das Resampling nicht auf den gesamten Pfad, sondern lediglich auf den Subpfad des NRC und auf die ersten wenigen Lichtquellen anzuwenden.

# Literatur

---

- Bitterli, B., C. Wyman, M. Pharr, P. Shirley, A. Lefohn und W. Jarosz (Aug. 2020). „Spatiotemporal Reservoir Resampling for Real-Time Ray Tracing with Dynamic Direct Lighting“. In: *ACM Trans. Graph.* 39.4. URL: <https://doi.org/10.1145/3386569.3392481>.
- Bittner, K. (Jan. 2020). *The Current State of the Art in Real-Time Cloud Rendering With Raymarching*.
- Delashmit, W. H., L. M. Missiles und M. T. Manry (2005). „Recent Developments in Multilayer Perceptron Neural Networks“. In.
- Dittebrandt, A., J. Hanika und C. Dachsbacher (2020). „Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing“. In: *Eurographics Symposium on Rendering - DL-only Track*. Hrsg. von C. Dachsbacher und M. Pharr. The Eurographics Association.
- Henyey, L. G. und J. L. Greenstein (Jan. 1940). „Diffuse radiation in the Galaxy“. In: *Annales d'Astrophysique* 3, S. 117.
- Högfeldt, R. (2016). „Convincing Cloud Rendering – An Implementation of Real-Time Dynamic Volumetric Clouds in Frostbite“. In: URL: <http://publications.lib.chalmers.se/records/fulltext/241770/241770.pdf>.
- Jarosz, W., C. Donner, M. Zwicker und H. W. Jensen (März 2008). „Radiance Caching for Participating Media“. In: *ACM Transactions on Graphics (Presented at SIGGRAPH)* 27.1, 7:1–7:11.
- Jarosz, W., D. Nowrouzezahrai, R. Thomas, P.-P. Sloan und M. Zwicker (Dez. 2011). „Progressive Photon Beams“. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 30.6.
- Kallweit, S., T. Müller, B. McWilliams, M. Gross und J. Novák (Nov. 2017). „Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks“. In: *ACM Trans. Graph. (Proc. of Siggraph Asia)* 36.6. URL: <http://doi.acm.org/10.1145/3130800.3130880>.

## LITERATUR

---

- Kingma, D. P. und J. Ba (2014). *Adam: A Method for Stochastic Optimization*. URL: <https://arxiv.org/abs/1412.6980>.
- Lin, D., C. Wyman und C. Yuksel (Dez. 2021). „Fast Volume Rendering with Spatiotemporal Reservoir Resampling“. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2021)* 40.6, 278:1–278:18. URL: <http://doi.acm.org/10.1145/3478513.3480499>.
- Majercik, Z., J.-P. Guertin, D. Nowrouzezahrai und M. McGuire (Juni 2019). „Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields“. In: *Journal of Computer Graphics Techniques (JCGT)* 8.2, S. 1–30. URL: <http://jcgt.org/published/0008/02/01/>.
- Marco, J., A. Jarabo, W. Jarosz und D. Gutierrez (Apr. 2018). „Second-order occlusion-aware volumetric radiance caching“. In: *ACM Transactions on Graphics (Presented at SIGGRAPH)* 37.2.
- Mie, G. (1908). „Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen“. In: *Annalen der Physik* 330.3, S. 377–445. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/andp.19083300302>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19083300302>.
- Mildenhall, B., P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi und R. Ng (2020). „NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis“. In: *CoRR* abs/2003.08934. arXiv: 2003.08934. URL: <https://arxiv.org/abs/2003.08934>.
- Müller, T. (Apr. 2021). *tiny-cuda-nn*. Version 1.7. URL: <https://github.com/NVlabs/tiny-cuda-nn>.
- Müller, T., A. Evans, C. Schied und A. Keller (Juli 2022). „Instant Neural Graphics Primitives with a Multiresolution Hash Encoding“. In: *ACM Trans. Graph.* 41.4, 102:1–102:15. URL: <https://doi.org/10.1145/3528223.3530127>.
- Müller, T., B. Mcwilliams, F. Rousselle, M. Gross und J. Novák (Okt. 2019). „Neural Importance Sampling“. In: *ACM Trans. Graph.* 38.5. URL: <https://doi.org/10.1145/3341156>.
- Müller, T., F. Rousselle, J. Novák und A. Keller (Aug. 2021). „Real-time Neural Radiance Caching for Path Tracing“. In: *ACM Trans. Graph.* 40.4, 36:1–36:16. URL: <https://doi.org/10.1145/3450626.3459812>.
- Novák, J., I. Georgiev, J. Hanika und W. Jarosz (Mai 2018). „Monte Carlo methods for volumetric light transport simulation“. In: *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)* 37.2.
- Novák, J., A. Selle und W. Jarosz (Nov. 2014). „Residual Ratio Tracking for Estimating Attenuation in Participating Media“. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 33.6.

## LITERATUR

---

- Qian, N. (1999). „On the momentum term in gradient descent learning algorithms“. In: *Neural Networks* 12.1, S. 145–151. URL: <https://www.sciencedirect.com/science/article/pii/S0893608098001166>.
- Woodcock, E. (1965). „Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry“. In: *Proceedings of the conference on applications of computing methods to reactor problems, 1965* 557. URL: <https://cir.nii.ac.jp/crid/1573668925582592640>.
- Zhang, J. (2019). „On sampling of scattering phase functions“. In: *Astronomy and Computing* 29, S. 100329. URL: <https://www.sciencedirect.com/science/article/pii/S2213133719300125>.