



# Rendering Denoising

Path Tracing Denoising mit neuronalen Netzen

**Bachelorthesis**

Studiengang:

Computer Perception and Virtual Reality

Autor:

Pascal Luc Cornu

Betreuer:

Prof. Marcus Hudritsch

Experte:

Dr. Harald Studer

Datum:

12.06.2025



Berner  
Fachhochschule

Technik und Informatik

## Management Summary

Diese Arbeit untersucht, wie die Bildqualität beim physikalisch korrekten Rendern (Path Tracing) effizient verbessert werden kann. Path Tracing ist ein verbreitetes Verfahren zur realistischen Bildsynthese in der Computergrafik. Es simuliert den Weg von Lichtstrahlen in einer virtuellen Szene sehr genau, ist jedoch mit hohem Rechenaufwand verbunden. Besonders bei Szenen mit indirekter oder komplexer Beleuchtung entsteht deutlich sichtbares Rauschen, wenn nicht genügend Rechenzeit aufgewendet wird. Dieses Rauschen äussert sich in Form von körnigen oder fleckigen Bildern. Ziel der Arbeit war es, ein Verfahren zu entwickeln, das solche verrauschten Bilder automatisch verbessern kann – und zwar mithilfe Künstlicher Intelligenz. Dafür wurde ein eigener Trainingsdatensatz erstellt, bestehend aus verrauschten und hochwertigen Vergleichsbildern. Anschliessend wurde ein künstliches neuronales Netz trainiert, das lernt, wie ein Bild mit wenig Rechenzeit aussieht und wie es idealerweise aussehen sollte. Es kann dann auf neue Bilder angewendet werden, um diese deutlich zu verbessern.

Die Bildqualität konnte dadurch messbar gesteigert werden: Der durchschnittliche Unterschied zur Originalqualität wurde um rund 70 % verringert. Gleichzeitig verbesserte sich die visuelle Ähnlichkeit zu den perfekten Bildern deutlich. Subjektiv bedeutet das: Die vom Modell bearbeiteten Bilder wirken deutlich sauberer, detailreicher und natürlicher als die ursprünglichen verrauschten Versionen – und das ohne zusätzliche Rechenzeit beim Rendern.

Damit zeigt diese Arbeit das Potenzial von Deep Learning zur Beschleunigung realistischer Bildsynthese – insbesondere in Anwendungsfeldern wie Animation, Produktvisualisierung oder Computerspielen. Auch wenn ein Vergleich mit industriellen Denoisern aus Zeitgründen nicht durchgeführt werden konnte, belegen die Ergebnisse die Leistungsfähigkeit des entwickelten Modells. Die Resultate legen zudem nahe, dass sich Deep-Learning-gestützte Entrauschung in bestehende Workflows integrieren lässt, um Renderzeiten zu senken, ohne die Bildqualität zu beeinträchtigen. Einschränkungen bestehen derzeit vor allem in der begrenzten Datenvielfalt und dem fehlenden Vergleich mit etablierten Denoisern. Zukünftige Arbeiten könnten diese Aspekte adressieren und das Verfahren so weiter in Richtung praktischer Anwendung führen.

# Inhaltsverzeichnis

Path Tracing Denoising mit neuronalen Netzen	1
Management Summary	2
Inhaltsverzeichnis	3
1 Einleitung	5
1.1 Forschungsfrage und Zielsetzung	6
1.2 Hintergrund	6
1.2.1 Anwendungen	6
1.3 Path Tracing	7
1.3.1 Rendering-Gleichung	7
1.3.2 Monte-Carlo-Integration beim Path Tracing	7
1.3.3 Path Tracing Algorithmus	8
1.3.4 Path Tracing Rauschen	9
1.4 Denoising	10
1.4.1 Klassische Filtermethoden	10
1.4.2 Statistische Modelle	10
1.4.3 Lernbasierte Methoden	10
1.5 Verwandte Arbeiten	11
2 Projektmanagement	12
2.1 Projektziele	12
2.1.1 Theoretische Untersuchung von Path Tracing und Denoising	12
2.1.2 Datensatzgenerierung	12
2.1.3 Entwicklung und Training eines Denoising-Modells	12
2.1.4 Vergleich mit bestehenden Lösungen	12
2.1.5 Integration des Denoising-Modells	12
2.2 Meilensteine	13
2.2.1 Planung	13
2.2.2 Forschung	13
2.2.3 Datensatz	13
2.2.4 Modell	13
2.2.5 Präsentation und Dokumentation	13
2.3 Projektablauf	14
2.4 Kommunikation und Dokumentation	16
2.5 Risikomanagement	16
2.6 Reflexion des Projektmanagements	16
3 Implementation	17
3.1 Erstellung des Datensatzes	17
3.1.1 Szenenvorbereitung in Blender	17
3.1.2 Automatisierte Bildgenerierung per Python-Skript	18
3.1.3 Bildausschnitte	20
3.1.4 Organisation und Verwendung des Datensatzes	20
3.2 Erstellung des Modells	20
3.2.1 Modell Architektur	20
3.2.2 Entwicklung der Modellarchitektur	24
3.2.3 Optimierung der Hyperparameter	25
3.2.4 Schutz vor Overfitting	26
3.3 Modell Training	27
3.4 Grafische Anwendung	29
3.4.1 Inferencing-Performance	31
3.4.2 Behandlung von Artefakten an Patch-Grenzen	31
4 Resultate	33
4.1 Bewertungsmethoden	33
4.2 Visuelle Ergebnisse	33

4.3 Quantitative Ergebnisse	34
4.4 Zielerreichung	36
4.4.1 Theoretische Untersuchung von Path Tracing und Denoising	37
4.4.2 Datensatzgenerierung	37
4.4.3 Entwicklung und Training eines Denoising-Modells	37
4.4.4 Vergleich mit bestehenden Lösungen	37
4.4.5 Integration des Denoising Modells	37
5 Fazit	38
5.1 Verbesserungspotenzial	38
5.2 Ausblick	38
6 Abbildungsverzeichnis	40
7 Tabellenverzeichnis	40
8 Glossar	41
9 Literaturverzeichnis	43
10 Anhang	44
10.1 Weitere Ergebnisse	44
10.2 SLProject4 Resultat	48
10.3 Meeting Protokoll	49
10.3.1 Kick-off Meeting (24.02.2025 09:00)	49
10.3.2 Abgleich 1 (13.03.2025 11:00)	49
10.3.3 Abgleich 2 (24.03.2025 09:00)	49
10.3.4 Expertentreff (07.04.2025 09:00)	50
10.3.5 Abgleich 3 (22.04.2025 09:00)	50
10.3.6 Abgleich 4 (05.05.2025 09:00)	50
10.3.7 Abgleich 5 (19.05.2025 09:00)	51
10.3.8 Abgleich 6 (02.06.2025 09:00)	51
11 Selbständigkeitserklärung	52

# 1 Einleitung

Path Tracing ist eine physikalisch genaue Rendering-Technik, die in der Computergrafik weit verbreitet ist, um das Verhalten von Licht zu simulieren. Sie ist in der Lage, äusserst realistische Bilder zu erzeugen, indem sie die Pfade einzelner Lichtstrahlen bei der Interaktion mit Oberflächen in einer Szene nachzeichnet. Diese Methode ist jedoch rechenintensiv und anfällig für Rauschen, insbesondere in Szenarien mit komplexen Beleuchtungseffekten wie globaler Beleuchtung, Kaustik und indirekter Beleuchtung.

Die Hauptquelle des Rauschens beim Path Tracing liegt in der stochastischen Natur der Monte-Carlo-Integration, die zur Annäherung an die Rendering-Gleichung verwendet wird. Da die Farbe jedes Pixels durch Mittelwertbildung aus einer endlichen Anzahl zufälliger Lichtpfade geschätzt wird, führt eine geringe Anzahl von Stichproben zu einer hohen Varianz, die sich im endgültigen Bild als körniges oder fleckiges Rauschen äussert. Eine Erhöhung der Anzahl der Abtastwerte pro Pixel kann das Rauschen zwar verringern, doch ist dieser Ansatz aufgrund der ansteigenden Rendering-Zeit oft nicht praktikabel.

Um dieses Problem zu lösen, wurden Entrauschungstechniken entwickelt, welche qualitativ hochwertige Bilder mit deutlich weniger Stichproben erzeugen. Diese Entrauschungsverfahren nutzen statistische, analytische und auf maschinellem Lernen basierende Methoden, um ein sauberes Bild aus einem verrauschten Eingangssignal zu rekonstruieren. Herkömmliche Entrauschungsmethoden beruhen auf Filtertechniken, die verrauschte Regionen glätten und dabei wichtige Details erhalten, während moderne, auf Deep Learning basierende Entrauschungsmethoden neuronale Netzwerke verwenden, die auf grossen Datensätzen trainiert wurden, um Rauschen effektiv vorherzusagen und zu entfernen. Durch die intelligente Unterscheidung von Rauschen und echten Details ermöglichen diese Verfahren schnellere Rendering-Workflows und machen Path Tracing für Echtzeitanwendungen wie interaktives Rendering und Spielgrafik möglich.

Diese Arbeit erforscht die Rauschunterdrückung beim Path Tracing mit Verwendung von neuronalen Netzwerken.



Abbildung 1: Vergleich zwischen verrauschemtem und entrauschemtem Bild aus der Demo-Szene «Barbershop Interior» von der Blender Foundation [19], Lizenz: CC-BY

## 1.1 Forschungsfrage und Zielsetzung

In dieser Arbeit wird untersucht, wie ein Autoencoder als neuronales Netz zur Rauschunterdrückung in Path-Tracing-basierten Renderings eingesetzt werden kann. Um den Forschungsbeitrag klar einzugrenzen, wird die zentrale Frage dieser Arbeit wie folgt formuliert:

***Inwiefern kann ein neuronales Netz, insbesondere ein Autoencoder, zur effektiven Rauschunterdrückung in Path-Tracing-Renderings mit niedriger Abtastrate eingesetzt werden, um qualitativ hochwertige Bilder bei möglichst geringem Rechenaufwand zu rekonstruieren?***

### Zielsetzung:

Ziel dieser Arbeit ist es, ein neuronales Netz zur Rauschunterdrückung («Denoising») in Path-Tracing-Bildern zu entwickeln und zu evaluieren. Dabei soll insbesondere untersucht werden:

1. Wie effektiv ein Autoencoder bei der Rekonstruktion verrauchter Renderings mit wenigen Samples ist
2. Wie die Netzwerkarchitektur (Tiefe, Skip-Connections etc.) die Denoising-Leistung beeinflusst.
3. Wie gut das Netz auf unterschiedlichen Rauschstufen generalisiert, d. h. bei verschiedenen Sample-Zahlen.

Ein weiterführendes Ziel war der Vergleich des entwickelten Denoisers mit bestehenden Verfahren. Aufgrund von Zeitbeschränkungen wurde dieser Teil jedoch nicht umgesetzt.

## 1.2 Hintergrund

Die realitätsnahe Bildsynthese spielt eine zentrale Rolle in zahlreichen Bereichen der Computergrafik, darunter Filmproduktion, Architekturvisualisierung, Produktdesign sowie Echtzeitanwendungen wie Videospiele und Virtual Reality. Ein wesentliches Ziel dieser Disziplin ist es, physikalisch korrektes Lichtverhalten zu simulieren, um Bilder zu erzeugen, die möglichst fotorealistisch wirken. Eine der genauesten Methoden zur Erreichung dieses Ziels ist das Path Tracing, ein Monte-Carlo-basiertes Verfahren zur globalen Beleuchtung.

Hier setzt die Entrauschung («Denoising») an: Sie ermöglicht es, mit deutlich weniger Rechenaufwand qualitativ hochwertige Bilder zu erzeugen, indem Rauschen nachträglich entfernt wird.

Denoising ist somit ein zentraler Bestandteil moderner Renderpipelines. Sie erlaubt nicht nur eine signifikante Reduktion der Renderzeit, sondern erweitert auch die Einsatzmöglichkeiten von physikalisch basierten Renderern auf interaktive Anwendungen [1].

### 1.2.1 Anwendungen

Denoising wird in verschiedenen Bereichen eingesetzt:

- **Film und Animation:** In der professionellen Filmproduktion (z. B. mit Renderern wie Arnold oder RenderMan) werden pro Frame häufig Tausende Samples benötigt. Mit effektiven Denoisern kann diese Zahl drastisch reduziert werden, ohne sichtbaren Qualitätsverlust.
- **Interaktives Rendering:** In Game Engines wie Unreal Engine oder Unity werden Denoiser verwendet, um Echtzeitschatten, Reflexionen oder Beleuchtung zu verbessern.
- **Medizinische Bildgebung:** Auch in der CT- oder MRT-Bildgebung kommen Denoising-Verfahren zum Einsatz, um bei geringer Strahlendosis oder kurzen Scanzeiten rauschfreie Bilder zu rekonstruieren [2].
- **Wissenschaftliche Visualisierung:** In der Astrophysik oder Molekulardynamik-Simulation werden häufig visualisierte Daten entrauscht, um Interpretierbarkeit zu erhöhen.

### 1.3 Path Tracing

Path Tracing ist ein globaler Beleuchtungs-Rendering-Algorithmus, der die Physik des Lichttransports genau simuliert. Im Gegensatz zu rasterbasierten Methoden, die sich auf Heuristiken stützen, um Beleuchtungseffekte anzunähern, löst Path Tracing die Rendering-Gleichung direkt durch die Verfolgung von Lichtpfaden auf physikalische Weise. Aufgrund der Komplexität der Lichtinteraktionen in realen Szenen ist es nicht möglich, die Rendering-Gleichung analytisch zu lösen. Stattdessen kommt beim Path Tracing die Monte-Carlo-Integration zum Einsatz, ein statistisches Verfahren, das die Gleichung mithilfe von Zufallsstichproben approximiert.

#### 1.3.1 Rendering-Gleichung

Die Rendering-Gleichung [3], die von Kajiya (1986) eingeführt wurde, beschreibt den Transport von Licht in einer Szene:

$$L_0(x, \omega_0) = L_e(x, \omega_0) + \int_{\Omega} f_r(x, \omega_i, \omega_0) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

- $L_0(x, \omega_0)$  ist die ausgehende Strahldichte (radiance) am Punkt  $x$  in Richtung  $\omega_0$ .
- $L_e(x, \omega_0)$  ist die emittierte Strahldichte (radiance) von  $x$ , zum Beispiel von einer Lichtquelle.
- $f_r(x, \omega_i, \omega_0)$  ist die bidirektionale Reflexionsverteilungsfunktion (BRDF), die beschreibt, wie das Licht gestreut wird.
- $L_i(x, \omega_i)$  ist die einfallende Strahldichte (radiance) aus der Richtung  $\omega_i$ .
- $(\omega_i \cdot n)$  berücksichtigt den Kosinus des Einfallswinkels und sorgt für eine korrekte Energieerhaltung.
- Das Integral summiert die Beiträge aus allen einfallenden Lichtrichtungen  $\omega_i$  über die Hemisphäre  $\Omega$ .

Diese Gleichung modelliert, wie Licht mit Oberflächen interagiert. Aber sie direkt für jedes Pixel zu lösen, ist aufgrund ihrer rekursiven Natur unpraktisch. Licht, das an einem Punkt ankommt, kann mehrfach zurückgeworfen worden sein, bevor es die Kamera erreicht.

#### 1.3.2 Monte-Carlo-Integration beim Path Tracing

Die Monte-Carlo-Integration bietet eine praktische Möglichkeit zur Annäherung an die Rendering-Gleichung, indem eine endliche Anzahl von Stichproben ausgewertet wird, anstatt das vollständige Integral analytisch zu berechnen. Die Monte-Carlo-Approximation für ein Integral der Form

$$I = \int_D f(x) dx$$

wird gegeben durch:

$$I \approx \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{p(x_i)}$$

wobei:

- $N$  ist die Anzahl der Proben.
- $x_i$  sind zufällig ausgewählte Punkte in der Domäne  $D$ .
- $p(x)$  ist die Wahrscheinlichkeitsdichtefunktion (PDF), für die Stichprobe  $x$ .

Durch Anwendung auf die Rendering-Gleichung schätzt der Path-Tracing-Algorithmus die ausgehende Strahldichte, indem er Lichtpfade abtastet und deren Beiträge mittelt:

$$L_0(x, \omega_0) \approx \frac{1}{N} \sum_{i=1}^N \frac{f_r(x, \omega_i, \omega_0) L_i(x, \omega_i) (\omega_i \cdot n)}{p(x_i)}$$

### 1.3.3 Path Tracing Algorithmus

Path Tracing folgt einem stochastischen Ansatz zur Annäherung an die globale Beleuchtung:

1. **Strahlenerzeugung:** Für jedes Pixel an einer zufälligen Position wird ein Primärstrahl von der Kamera in die Szene geworfen.
2. **Strahlenschnittpunkt:** Der erste Schnittpunkt mit der Szene wird berechnet, typischerweise mit Hilfe von Bounding Volume Hierarchies (BVH) zur Beschleunigung der Ray-Scene-Intersection.
3. **Schattierung und BRDF-Sampling:** Die Oberflächen-BRDF bestimmt, wie das Licht reflektiert oder durchgelassen wird.
4. **Lichtbeteiligung und direkte Beleuchtung** (optional aber oft verwendet): Es kann explizites Sampling der Lichtquellen erfolgen, um direkte Beleuchtung effizienter zu berechnen.
5. **Rekursive Pfadverfolgung:** Ein neuer Strahl wird in eine zufällige Richtung gemäss der BRDF gesendet. Dieser Prozess wiederholt sich rekursiv, wobei Lichtbeiträge entlang des Pfades aufsummiert werden. Der Pfad endet, wenn:
  - eine Lichtquelle erreicht wird
  - die maximale Pfadlänge überschritten ist
  - per russischem Roulette stochastisch terminiert wird. Dabei wird die Fortführung eines Pfades probabilistisch auf Basis der reflektierten Strahlungsmenge (z. B. Helligkeit bzw. Reflektivität) entschieden. Perfekt schwarze Materialien führen mit hoher Wahrscheinlichkeit zum Abbruch, während reflektierende oder helle Oberflächen den Pfad mit hoher Wahrscheinlichkeit fortsetzen. Dieses Verfahren reduziert den Rechenaufwand, ohne statistischen Bias einzuführen.
6. **Akkumulierung:** Die entlang eines Pfades gesampelten Strahldichten (radiance contributions) werden aufsummiert, um den Lichtanteil am Startpunkt (z. B. Kamerapixel) zu berechnen.
7. **Monte-Carlo-Schätzung:** Die Schritte 1–6 werden mehrfach mit zufälligen Strahlrichtungen wiederholt. Die Ergebnisse werden gemittelt, um eine statistisch konvergente Schätzung der tatsächlichen Strahldichte zu erhalten.

Jeder Pfad ist eine Monte-Carlo-Stichprobe der Rendering-Gleichung. Bei einer ausreichenden Anzahl von Pfaden konvergiert die Annäherung zur richtigen Lösung.

Die Monte-Carlo-Integration ist zwar unverzerrt, konvergiert aber langsam und erfordert tausende von Stichproben pro Pixel, um rauschfreie Ergebnisse zu erzielen. Niedrige Stichprobenzahlen führen zu Monte-Carlo-Rauschen, das als körnige Artefakte sichtbar wird. Dies ist die Hauptmotivation für Rauschunterdrückungstechniken («Denoising»), die darauf abzielen, qualitativ hochwertige Bilder aus verrauschten Renderings mit geringer Abtastung zu rekonstruieren.

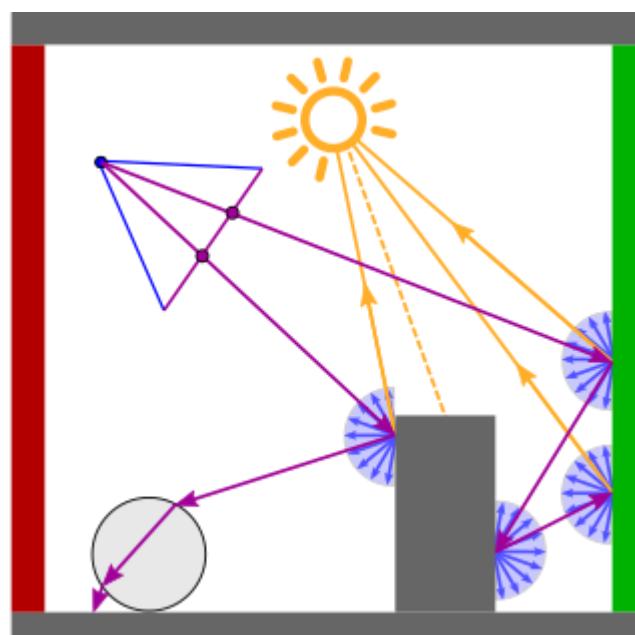


Abbildung 2: Path Tracing Illustration. Quelle: [20]

### 1.3.4 Path Tracing Rauschen

Das durch Path Tracing erzeugte Rauschen hat deutliche Merkmale, die es vom natürlichen Bildrauschen unterscheiden. Es ist das Ergebnis des Monte-Carlo-Samplings, bei dem die Farbe jedes Pixels anhand einer begrenzten Anzahl von zufälligen Lichtpfaden geschätzt wird. Dies führt zu einer hochfrequenten Varianz, die als gesprenkelte oder körnige Muster erscheint, insbesondere in Bereichen mit komplexer Beleuchtung wie indirekter Beleuchtung, Kaustik und weichen Schatten. Im Gegensatz zu natürlichem Rauschen, wie etwa dem Sensorrauschen von Digitalkameras, das in der Regel zufällig, additiv und statistisch gleichmässig über das Bild verteilt ist, ist das Rauschen durch Path Tracing stark strukturiert und szenenabhängig. Seine Intensität und Verteilung variiert mit der Anzahl der Proben pro Pixel, den Oberflächenmaterialien, den Lichtverhältnissen und der geometrischen Komplexität. Natürliches Rauschen ist in der Regel unabhängig vom Bildinhalt, während das Rauschen durch Path Tracing stark mit den visuellen und physikalischen Eigenschaften der Szene korreliert.

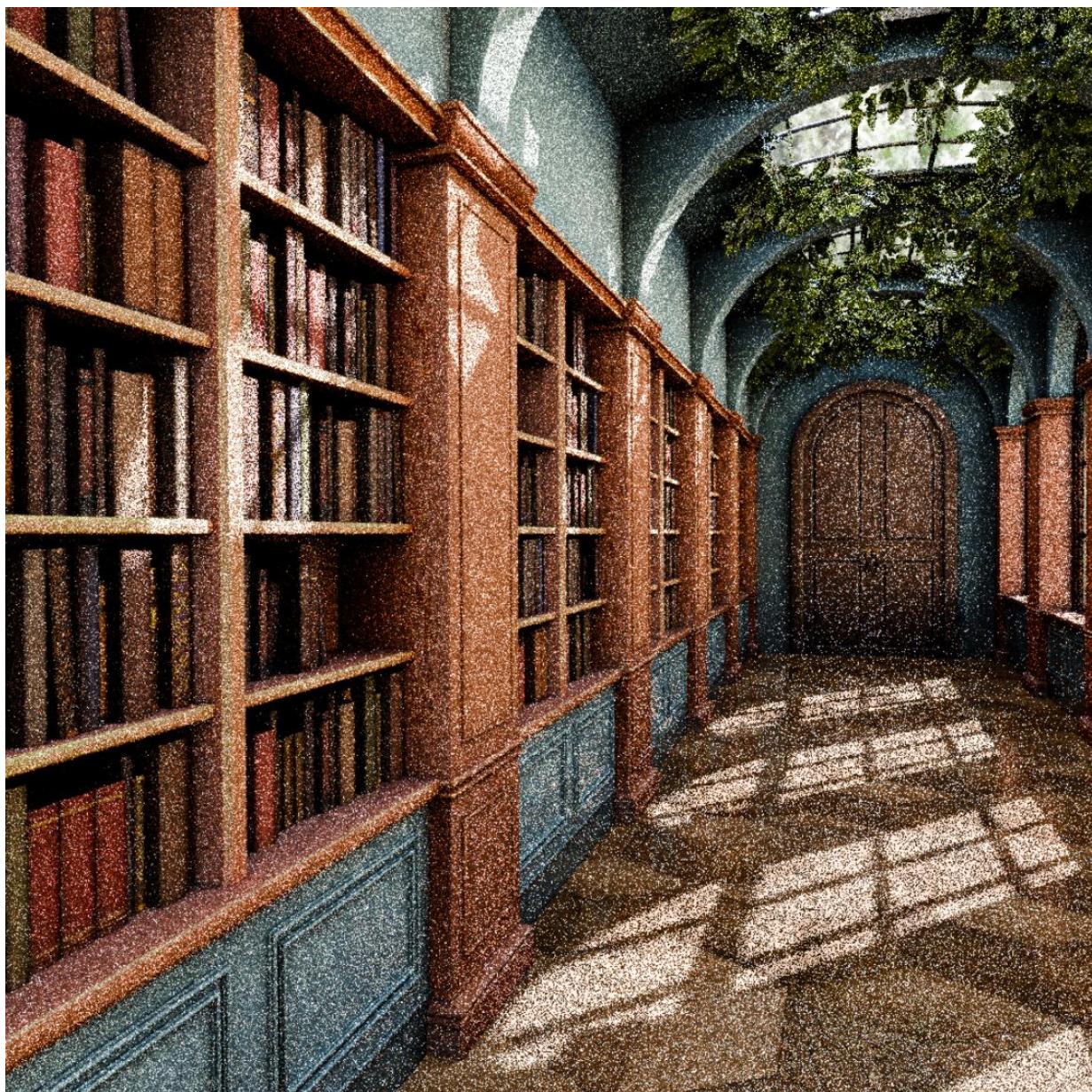


Abbildung 3: Verrausches Bücherregal mit Blender (Cycles) mit 5 Samples per Pixel. Quelle der Blender Szene: [4], Lizenzfrei

## 1.4 Denoising

Denoising bezeichnet den Prozess der Entfernung von unerwünschtem Rauschen aus digitalen Bildern, ohne dabei relevante Details oder Strukturen zu verfälschen. Insbesondere in der Path-Tracing-basierten Bildsynthese ist Denoising ein essenzieller Bestandteil der Nachbearbeitung, da das visuelle Rauschen bei geringen Samples per Pixel deutlich sichtbar ist. Ziel ist es, aus verrauschten Zwischenergebnissen möglichst hochwertige, rauschfreie Bilder zu rekonstruieren.

In der Bildverarbeitung existieren verschiedene Ansätze zur Rauschunterdrückung, die sich hinsichtlich Methodik, Komplexität und Anwendungsbereich unterscheiden. Grundsätzlich lassen sich Denoising-Methoden in drei Hauptkategorien einteilen: klassische filterbasierte Verfahren, statistische bzw. modellbasierte Verfahren und lernbasierte Methoden.

### 1.4.1 Klassische Filtermethoden

Zu den einfachsten Ansätzen zählen lineare und nichtlineare Filter, die direkt auf das verrauschte Bild angewendet werden. Bekannte Beispiele sind:

- **Mittelwertfilter:** Glättet das Bild durch Ersetzen jedes Pixels mit dem Durchschnittswert seiner Nachbarn. Dieser Ansatz reduziert Rauschen, führt jedoch oft zu sichtbarem Detailverlust.
- **Medianfilter:** Ersetzt jeden Pixel mit dem Median seiner Umgebung. Besonders effektiv bei impulsartigem («Salt-and-Pepper») Rauschen.
- **Gaussfilter:** Wendet eine gewichtete Glättung an, wobei näherliegende Pixel stärker berücksichtigt werden. Geeignet zur Rauschreduktion mit geringem Detailverlust.
- **Bilateralfilter:** Kombiniert räumliche Nähe und Ähnlichkeit der Farbwerte, um Kanten zu erhalten, während gleichzeitig Rauschen reduziert wird. Im Gegensatz zum Gaussfilter werden dabei auch Unterschiede in der Intensität berücksichtigt, wodurch ein besserer Kompromiss zwischen Glättung und Kantenerhalt erzielt wird.

Diese Methoden sind schnell und leicht implementierbar, stoßen aber bei komplexeren Rauschmustern, wie sie beim Path Tracing auftreten, schnell an ihre Grenzen.

### 1.4.2 Statistische Modelle

Moderne bildverarbeitende Verfahren nutzen statistische Modelle und Annahmen über die Bildstruktur:

- **Wiener-Filter:** Ein lineares Verfahren, das das Rauschen unter Kenntnis der Signal- und Rauschstatistik optimal unterdrückt.
- **BM3D (Block-Matching and 3D Filtering):** Ein nichtlineares Verfahren, das auf blockweiser Ähnlichkeit basiert. Es gruppiert ähnliche Bildbereiche, transformiert sie gemeinsam in einen Frequenzbereich und filtert dort gezielt Rauschen heraus. BM3D gilt lange Zeit als einer der leistungsstärksten klassischen Algorithmen für Denoising.

Diese Methoden erzielen oft bessere Ergebnisse als einfache Filter, sind jedoch rechenintensiver und schwer auf stark strukturierte oder nicht-stationäre Rauscharten anzupassen.

### 1.4.3 Lernbasierte Methoden

Mit dem Aufkommen leistungsfähiger Hardware und grosser Bilddatensätze haben sich neuronale Netze als äusserst effektive Werkzeuge zur Rauschunterdrückung etabliert. Hierbei wird ein Modell auf verrauschten und sauberen Bildpaaren trainiert, um eine Abbildung vom verrauschten zum rauschfreien Bild zu erlernen.

- **Convolutional Neural Networks (CNNs):** Klassische Deep-Learning-Modelle, die auf lokalen Bildmerkmalen basieren. Sie sind in der Lage, komplexe Rauschmuster zu erkennen und gezielt zu entfernen.
- **Autoencoder:** Netzwerke, die lernen, eine komprimierte Darstellung des Bildes zu erzeugen und daraus die rauschfreie Version zu rekonstruieren.
- **U-Net-Architekturen:** Erweiterte Autoencoder mit Skip-Connections, die besonders in der medizinischen Bildverarbeitung und beim Denoising in der Computergrafik verwendet werden.

- **Recurrent und Transformer-basierte Modelle:** Für spezielle Denoising-Aufgaben, insbesondere bei zeitlich korrelierten Bildern (wie Rendersequenzen), können auch rekurrente Strukturen oder Attention-Mechanismen sinnvoll sein.

Lernbasierte Methoden bieten den grossen Vorteil, dass sie an spezifische Rauschtypen angepasst und durch Training optimiert werden können. In der Praxis zeigen sie bei realistischen Renderbildern oft deutlich bessere Resultate als klassische Verfahren – insbesondere bei niedrigem Sample Count.

## 1.5 Verwandte Arbeiten

In den letzten Jahren wurden zahlreiche Verfahren zur Bildentrausung entwickelt, insbesondere im Bereich des physikalisch basierten Renderings (PBR), bei dem durch Path Tracing erzeugte Bilder stark verrauscht sein können. Zur Verbesserung der Bildqualität haben sich sowohl klassische als auch lernbasierte Ansätze etabliert.

Ein verbreiteter klassischer Denoiser ist der Open Image Denoise (OIDN) von Intel, der auf Deep Learning basiert und speziell für ray-traced Bilder optimiert ist [5]. OIDN verwendet neuronale Netze, um das Rauschen in Bildern zu reduzieren, und ist mittlerweile fester Bestandteil vieler 3D-Render-Engines wie Blender und LuxCoreRender. Das Modell wurde so trainiert, dass es sowohl Farbbilder als auch zusätzliche Informationen wie Albedo und Normalen nutzen kann, um die Qualität des Denoisings zu verbessern.

Ein weiterer etablierter Denoiser ist NVIDIA OptiX, der Teil der RTX-Technologie ist [6]. Dieser Denoiser basiert auf einem auf künstlicher Intelligenz trainierten Modell, das in Echtzeit Denoising ermöglicht, insbesondere auf GPUs mit Tensor Cores. NVIDIA verwendet hierfür ein Deep Learning-gestütztes Regressionsmodell, das sowohl Geschwindigkeit als auch hohe Qualität liefert. OptiX ist besonders im professionellen Bereich beliebt, da er in viele Anwendungen wie Autodesk Arnold, V-Ray aber auch in Blender integriert ist.

AMD Radeon ProRender verwendet ebenfalls einen KI-basierten Denoiser, der entweder auf OpenCL oder auf dem MIOpen-Framework basiert. Der Fokus liegt hier auf plattformübergreifender Unterstützung und der Integration in professionelle Workflows wie in Blender oder SOLIDWORKS [7].

Neben diesen industriellen Lösungen existieren zahlreiche Forschungsarbeiten, die auf verschiedenen Architekturen neuronaler Netze basieren. Zum Beispiel wurden Autoencoder, U-Net-Architekturen und Rekurrente Neuronale Netze (RNNs) erfolgreich für das Denoising von Path Tracing Bildern eingesetzt [8] [9]. Ein bekanntes Modell in diesem Bereich ist DnCNN (Denoising Convolutional Neural Network), das ursprünglich für das allgemeine Bildentrauschen entwickelt wurde, sich jedoch auch gut auf Path-Tracing-Daten übertragen lässt. Es basiert auf einem tiefen, voll-konvolutionalen Netzwerk mit Residual Learning, was die Konvergenz beschleunigt und das Denoising verbessert [10].

Zudem existieren hybride Ansätze, die klassische Filtertechniken wie den Bilateralfilter oder Non-Local Means mit Deep Learning kombinieren, um sowohl strukturelle Details zu bewahren als auch Artefakte zu vermeiden [11].

Aktuell gewinnen auch Diffusion-Models zunehmend an Bedeutung in der Bildsynthese und -restaurierung. Diese Modelle, die ursprünglich für die Bildgenerierung entwickelt wurden, zeigen vielversprechende Resultate bei Aufgaben wie Super-Resolution oder Denoising. Auch wenn ihr Einsatz im Bereich Path-Tracing-Denoising bislang kaum untersucht wurde, könnten sie in Zukunft durch ihre Fähigkeit zur schrittweisen Verbesserung verrauschter Daten eine relevante Rolle spielen [12].

## 2 Projektmanagement

### 2.1 Projektziele

Im Rahmen dieser Arbeit wurde das Ziel verfolgt, sich tiefgehend mit den Konzepten des Path Tracing und dem Denoising auseinanderzusetzen. Aufbauend auf theoretischem Wissen sollten geeignete Methoden zur Rauschunterdrückung analysiert, ein Datensatz generiert und ein neuronales Netz zur Bildverbesserung trainiert werden. Die konkreten Ziele der Arbeit waren:

#### 2.1.1 Theoretische Untersuchung von Path Tracing und Denoising

- Erarbeitung eines fundierten Verständnisses der Path-Tracing-Technik im Kontext der globalen Beleuchtung.
- Analyse der Ursachen für Bildrauschen in Path-Tracing-Renderings.
- Recherche und Bewertung bestehender Denoising-Techniken – insbesondere unter Verwendung neuronaler Netze.

#### 2.1.2 Datensatzgenerierung

- Untersuchung öffentlich verfügbarer Denoising-Datensätze auf die Eignung für Path-Tracing-Anwendungen.
- Falls erforderlich, Generierung eines eigenen Datensatzes mit Blender unter Verwendung unterschiedlicher Sample-Raten und einer Ground-Truth.

#### 2.1.3 Entwicklung und Training eines Denoising-Modells

- Entwicklung eines neuronalen Netzes zur Rauschunterdrückung, insbesondere in Form eines Autoencoders oder U-Net-ähnlichen Modells.
- Training des Modells auf den generierten Datensätzen unter Verwendung geeigneter Verlustfunktionen.
- Evaluierung der Ergebnisse auf Basis visueller Qualität und quantitativer Metriken.

#### 2.1.4 Vergleich mit bestehenden Lösungen

- Vergleich der erzielten Resultate mit bestehenden Denoising-Verfahren, z. B. dem OptiX Denoiser von NVIDIA oder den integrierten Denoisern in Blender.
- Analyse der Stärken und Schwächen des eigenen Ansatzes im Vergleich zu etablierten Methoden.

#### 2.1.5 Integration des Denoising-Modells

- Entwicklung und Anpassung des Modells zur Integration in eine Anwendung zur Bildverbesserung.
- Die Integration kann entweder direkt in den Path Tracer SLProject4 oder in eine neue, auf OpenCV basierende Applikation erfolgen.
- Ziel ist die Einsetzbarkeit des Modells zur automatischen Rauschunterdrückung in zukünftigen Rendering-Workflows.

## 2.2 Meilensteine

Im Folgenden werden die einzelnen Meilensteine des Projekts detailliert beschrieben. Sie dienen der zeitlichen und inhaltlichen Strukturierung der Arbeit und bilden die Grundlage für das Projektmanagement sowie die Überprüfung des Fortschritts.

### 2.2.1 Planung

- Regelmässige Meetings mit dem Betreuer wurden zu Beginn des Projekts vereinbart. Diese dienen dem fortlaufenden Austausch, der Diskussion des Fortschritts sowie der Klärung offener Fragen.
- Die übergeordneten Projektziele sowie die konkreten Arbeitspakete und Meilensteine wurden in Absprache mit dem Betreuer definiert und dokumentiert.
- Ein detaillierter Projektzeitplan wurde erstellt, der die einzelnen Phasen – von der Recherche bis zur Präsentation – umfasst. Dieser Plan dient als Leitfaden für die termingerechte Umsetzung der Projektziele.

### 2.2.2 Forschung

- Eine umfassende Literaturrecherche zum Thema Path Tracing sowie zu aktuellen Ansätzen im Bereich der Denoising wurde durchgeführt. Die gewonnenen Erkenntnisse wurden dokumentiert und bilden die theoretische Grundlage für die weitere Arbeit.
- Der im SLProject4 enthaltene Path Tracer wurde analysiert. Dabei wurden bestehende Fehler identifiziert und – soweit möglich – behoben.

### 2.2.3 Datensatz

- Bestehende öffentliche Datensätze im Bereich des Bilddenoising wurden recherchiert und auf ihre Eignung für die spezifischen Anforderungen des Projekts hin evaluiert.
- Sollte ein geeigneter Datensatz gefunden worden sein, wurde dieser übernommen und in die Projektinfrastruktur integriert.
- Falls kein passender Datensatz verfügbar war, wurde mithilfe von Blender ein eigener Datensatz generiert. Dabei wurden sowohl verrauschte als auch entrauschte Renderings mithilfe unterschiedlicher Sampling-Raten und der Cycles-Engine erstellt. Der Datensatz wurde strukturiert gespeichert und für das Training der Modelle aufbereitet.

### 2.2.4 Modell

- Basierend auf dem erstellten oder ausgewählten Datensatz wurde ein erstes neuronales Netzwerk-Modell implementiert. Dieses wurde auf die Aufgabe des Bilddenoisings trainiert. Das Modell ist in der Lage, verrauschte Bilder zu verarbeiten und eine entrauschte Version als Ausgabe zu liefern.
- Falls unterschiedliche Modellansätze (z. B. Autoencoder, CNN, U-Net) als sinnvoll erachtet wurden, wurden mindestens zwei verschiedene Architekturen entwickelt, trainiert und miteinander verglichen. Die Auswahl der Modelle basierte auf ihrer Leistungsfähigkeit und Komplexität.

### 2.2.5 Präsentation und Dokumentation

- Alle Präsentationsmedien – darunter die schriftliche Dokumentation, ein Erklärvideo, ein Poster sowie die Folien für den Vortrag – wurden erstellt, abgestimmt und finalisiert.
- Die abschliessende Präsentation wurde geübt, um einen sicheren und professionellen Auftritt während der Ausstellung zu gewährleisten.
- Zusätzlich wurde der Aufbau für die Abschlussausstellung geplant und vorbereitet, um die Projektergebnisse anschaulich zu präsentieren.

## 2.3 Projektablauf

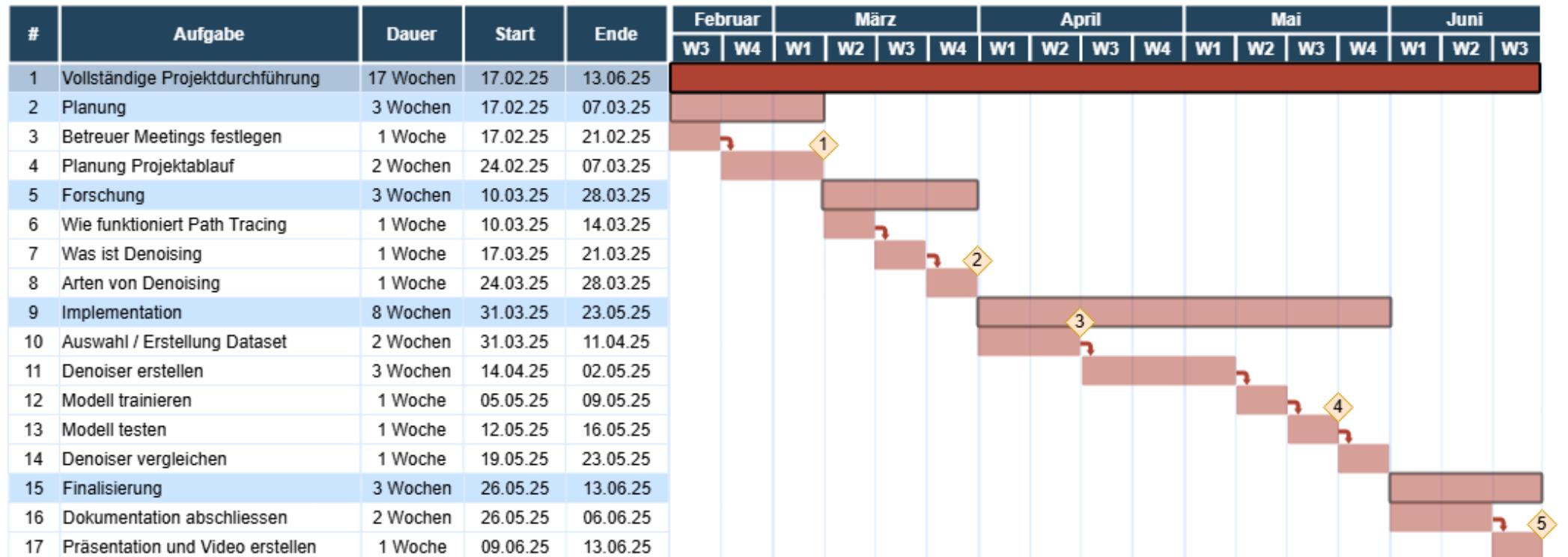


Abbildung 4: Projektablauf Gantt Diagramm

Aufgabe	Start	Ende	Zeit (h)	Anmerkungen / Was wurde gemacht
Projektplanung und Organisation	17.02.2025	01.03.2025	18	Aufsetzung der Meetings, Dokumentation, Projektablauf (Gantt Diagramm), Ziele und Meilensteine.
Forschung über Path Tracing und Denoising	24.02.2025	24.03.2025	50	Recherche und Dokumentation von Path Tracing und Denoising. Vergleich von SmallPT und SLProject4. Analyse der Fireflies von SLProject4. Simpler Autoencoder mit Tutorial nachgebaut.
Erstellung des Datensatzes	13.03.2025	14.04.2025	75	Recherche von öffentlich verfügbaren Datensätzen. Skript für das automatische Rendern von mehreren Kameras in Blender Szenen. Auswahl von 15 Szenen, aufgeteilt auf Trainings-, Validierungs-, und Testdatensatz. Aufteilung der Bilder per Skript in kleinere Patches für ein Autoencoder Modell.
Präsentation Expertentreff	01.04.2025	07.04.2025	8	Vorbereitung und Durchführung vom Expertentreff.
Erstellung des Autoencoders	14.04.2025	06.05.2025	56	Erstellung eines Autoencoders mit U-Net style Skip-Connections und Massnahmen gegen Overfitting. Hyperparameter-Tuning mit Optuna. Beinhaltet auch das erstellte Monitoring für die Tunings- und Trainingsruns.
Training des Autoencoders	06.05.2025	16.05.2025	25	Zugang zur «Management Platform for Deep Learning» (MLMP) angefragt. Skript Anpassungen für MLMP. Einrichtung des Projektes auf dem Server. Optuna und Modell Training durchgeführt.
Poster und Book	16.05.2025	06.06.2025	14	Erstellung von Poster und Book.
Modell Testen und grafische Applikation	20.05.2025	02.06.2025	20	Anreicherung des Testdatensatzes und Skript für die Verwendung des Modells auf den Testdaten. Erstellung der grafischen Applikation.
Finale Dokumentation, Präsentation und Video	25.05.2025	12.06.2025	75	Finalisierung der Dokumentation, Erstellung von Präsentation und Video.
<b>Total</b>			<b>341</b>	

Tabelle 1: Tatsächlich investierte Zeit

## 2.4 Kommunikation und Dokumentation

Im Rahmen des Projekts fand alle zwei Wochen ein Meeting mit dem Betreuer statt. In diesen Treffen präsentierte ich die erzielten Ergebnisse und Fortschritte der letzten zwei Wochen und diskutierte mit dem Betreuer die nächsten Schritte sowie offene Fragestellungen. Diese regelmässigen Austauschtermine ermöglichen eine kontinuierliche Kontrolle des Projektfortschritts und eine frühzeitige Identifikation von Herausforderungen.

Zur Dokumentation wurden während des gesamten Projektverlaufs schriftliche Protokolle der Meetings geführt, welche die besprochenen Themen, Entscheidungen und To-Do-Punkte festhielten. Dies erleichterte die Nachverfolgung des Projektverlaufs. Zusätzlich wurde der Quellcode sowie die zugehörigen Datenversionen systematisch in einem Git-Repository [13] verwaltet, um Versionskontrolle und Verfügbarkeit zu gewährleisten.

## 2.5 Risikomanagement

Während der Projektplanung wurden potenzielle Risiken identifiziert und bewertet, um deren Einfluss auf den Projektverlauf zu minimieren.

Risiko	Wahrscheinlichkeit	Auswirkung	Gegenmassnahmen	Ergebnis
Komplexität der Implementierung des neuronalen Netzes	Mittel	Hoch	Zeitpuffer einplanen, iterative Entwicklung	Erfolgreiche Implementierung nach Zeitplan
Qualität und Verfügbarkeit des Datensatzes	Mittel	Hoch	Eigene Datensatzgenerierung mit Blender	Datensatz wurde erfolgreich generiert
Vergleich mit kommerziellen Denoisern nicht möglich	Hoch	Mittel	Fokus auf Entwicklung und Evaluierung des eigenen Modells	Vergleich konnte wegen Zeitmangel nicht durchgeführt werden
Verzögerungen im Projektverlauf	Mittel	Hoch	Meilensteinplanung, regelmässige Meetings	Projektverlauf blieb im Rahmen

Tabelle 2: Risikomanagement Tabelle

## 2.6 Reflexion des Projektmanagements

Das Projektmanagement während dieser Arbeit funktionierte insgesamt sehr gut. Die regelmässigen zweiwöchentlichen Meetings mit dem Betreuer haben sich als besonders wertvoll erwiesen. Sie sorgten für eine klare Struktur und einen festen Rhythmus, der mich kontinuierlich motivierte und gleichzeitig einen gewissen Leistungsdruck erzeugte, der half, die gesteckten Ziele nicht aus den Augen zu verlieren. Durch den kontinuierlichen Austausch konnten auftretende Probleme frühzeitig erkannt und besprochen werden, was den Projektverlauf positiv beeinflusste.

Trotz der guten Organisation gibt es auch Verbesserungspotenzial für zukünftige Projekte. So wäre es sinnvoll, ein etabliertes Projektmanagement-Framework, wie beispielsweise Scrum oder Kanban, einzuführen, um Aufgaben und Fortschritte noch strukturierter zu planen und zu verfolgen.

Insbesondere der Einsatz von Issue-Tracking-Systemen wie GitHub Issues könnte helfen, offene Aufgaben, Bugs und Ideen übersichtlich zu dokumentieren und den Fortschritt transparenter zu machen. Dies würde zudem die Zusammenarbeit mit anderen Beteiligten erleichtern und die Nachverfolgbarkeit verbessern.

Insgesamt war das Projektmanagement ein wichtiger Erfolgsfaktor für die Umsetzung der Arbeit, und die gewonnenen Erkenntnisse werden in künftigen Projekten zur weiteren Optimierung genutzt.

## 3 Implementation

### 3.1 Erstellung des Datensatzes

Für das Training und die Evaluierung des neuronalen Netzes zur Rauschunterdrückung wurde ein eigener Datensatz erstellt, der synthetisch mit Blender generiert wurde. Ziel war es, realitätsnahe gerenderte Bilder mit definierten Rausch-Leveln zu erzeugen und diese mit qualitativ hochwertigen Referenzbildern (Ground Truth) zu paaren. Die Erstellung des Datensatzes erfolgte in mehreren automatisierten und manuellen Schritten.

#### 3.1.1 Szenenvorbereitung in Blender

Zur Datengenerierung wurden mehrere 3D-Szenen in Blender vorbereitet. Dafür wurden kostenlose öffentliche Szenen von Blender verwendet. In jeder Szene wurden mehrere Kamerapositionen manuell platziert, um verschiedene Blickwinkel und Bildinhalte zu erfassen. Die Kameras wurden so positioniert, dass sie unterschiedliche Bereiche der Szene abdecken, um eine möglichst hohe Varianz im Datensatz zu gewährleisten. Folglich gab es mehr Kameras für eine Szene, wenn diese viele Details beinhaltet. Für jede Kamera wurde anschliessend eine Reihe von Bildern aus derselben Szene, aber mit unterschiedlichen Rauschstufen erzeugt.

Folgende Szenen wurden in den jeweiligen Datensätzen verwendet:

#### Trainingsdatensatz:

- |   |   |  |
|---|---|--|
| • [MySimsWorld]LibraryHall                      | - | 18 Kameraperspektiven  |
| • barbershop_interior                           | - | 19 Kameraperspektiven  |
| • BarcelonaPavilion                             | - | 8 Kameraperspektiven   |
| • classroom                                     | - | 8 Kameraperspektiven   |
| • Dice  | - | 1 Kameraperspektive<br>(Dice wurde später eingefügt, um ein Problem mit der synthetischen Grünen Farbe zu beheben) |
| • flat-archiviz                                 | - | 7 Kameraperspektiven   |
| • geometric_surface_patterns                    | - | 3 Kameraperspektiven   |
| • LivingRoom-02(packed)                         | - | 4 Kameraperspektiven   |
| • monster_under_the_bed_sss_demo_by_metin_seven | - | 9 Kameraperspektiven   |

#### Validierungsdatensatz

- |                            |   |                      |
|----------------------------|---|----------------------|
| • [MySimsWorld]TheBookNook | - | 9 Kameraperspektiven |
| • archiviz                 | - | 4 Kameraperspektiven |
| • loft                     | - | 1 Kameraperspektive  |

#### Testdatensatz

- |                     |   |                      |
|---------------------|---|----------------------|
| • cornell-box       | - | 2 Kameraperspektiven |
| • mr-elephant       | - | 6 Kameraperspektiven |
| • RestRoom-Interior | - | 2 Kameraperspektiven |

Folglich ein Beispiel aus der Szene «[MySimsWorld]LibraryHall» in Blender:



Abbildung 5: Beispilszene aus Blender, Quelle: [4], Lizenzfrei

### 3.1.2 Automatisierte Bildgenerierung per Python-Skript

Ein zentrales Element der Datensatz-Generierung ist ein eigens entwickeltes Python-Skript, das Blender im sogenannten Headless-Modus (d. h. ohne Benutzeroberfläche) startet. Dies ermöglicht eine vollautomatisierte Verarbeitung und eignet sich besonders für das Rendering auf Servern oder in CI-Umgebungen (Continuous Integration), also automatisierten Build- und Testsystemen ohne grafische Oberfläche.

Das Skript verarbeitet jede vorbereitete Szene sequenziell und führt folgende Schritte aus:

- Öffnen der .blend-Datei.
- Initialisierung der Blender-Cycles-Renderengine.
- Iteration über alle definierten Kameras.
- Für jede Kamera werden Bilder mit unterschiedlichen Samples per Pixel (SPP) gerendert: 2, 5, 10, 25, 50, 100, 200, 500. Diese Bilder enthalten das gewünschte Path Tracing Rauschen und dienen als Trainingsinput.
- Für jede Kameraperspektive wird zusätzlich ein Ground-Truth-Bild erzeugt. Dieses basiert auf nur 100 Samples pro Pixel, wird jedoch mit dem OptiX-Denoiser von Blender nachbearbeitet, um ein visuell möglichst rauschfreies Referenzbild zu erhalten.

Die Bilder werden in einer Auflösung von  $1024 \times 1024$  Pixeln gespeichert. Alle Ausgaben werden systematisch benannt, um eine eindeutige Zuordnung zwischen verrauschten und rauschfreien Bildern zu ermöglichen.

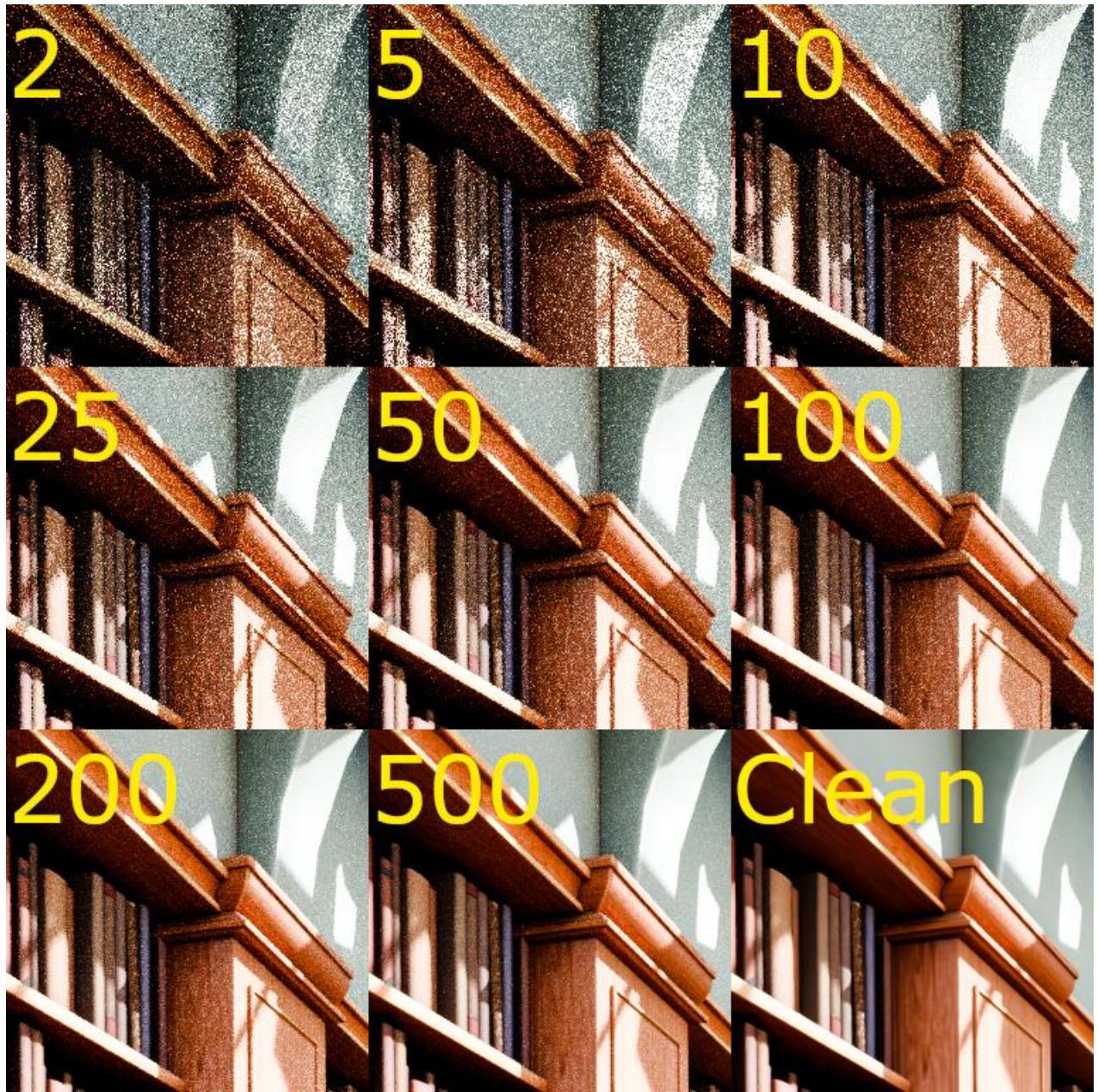


Abbildung 6: Visualisierung der verschiedenen Rauschstufen eines Patches

Die obige Abbildung zeigt ein Beispiel eines Bildpatches aus dem Datensatz. Es sind insgesamt neun Varianten desselben Ausschnitts dargestellt – acht mit unterschiedlichem Rauschgrad, erzeugt durch verschiedene Sample-Werte (Samples per Pixel), sowie ein entrausches Referenzbild (Ground Truth). Die Sample-Anzahl ist jeweils oben links im Bild angegeben; das Referenzbild ist mit «Clean» gekennzeichnet. Die zunehmende Bildqualität mit steigender Sample-Zahl ist deutlich erkennbar.

### 3.1.3 Bildausschnitte

Da neuronale Netze, insbesondere Autoencoder, typischerweise auf kleinere Eingabedimensionen trainiert werden, wurden die  $1024 \times 1024$  grossen gerenderte Bilder in kleinere Bildausschnitte (Patches) unterteilt. Ein separates Python-Skript übernimmt diese Aufgabe:

- Jedes Bild wird in 16 nicht überlappende Patches der Grösse  $256 \times 256$  Pixel unterteilt.
- Dabei werden sowohl die verrauschten Bilder als auch die entsprechenden Ground-Truth-Bilder verarbeitet.
- Die resultierenden Patches werden mit einem konsistenten Namensschema abgespeichert, das folgende Informationen enthält:
  - Szenenname
  - Kameranummer
  - Sampleanzahl (für verrauschte Bilder)
  - Patch-Index

Beispielhafte Dateinamen:

- Scene-name\_cam\_0\_noisy\_10\_patch\_5.png
- Scene-name\_cam\_0\_clean\_patch\_5.png

Diese Struktur ermöglicht eine effiziente Zuordnung der Trainingsdaten während des Ladevorgangs in das neuronale Netz.

### 3.1.4 Organisation und Verwendung des Datensatzes

Der vollständige Datensatz besteht aus 12928 Patch-Paaren (verrauschtes Eingabebild + Ground Truth). Die Daten wurden in drei Teilmengen aufgeteilt:

- Training – 9856 Patch-Paare
- Validierung – 1792 Patch-Paare
- Test – 1280 Patch-Paare

Die Verzeichnisse sind entsprechend gegliedert, um eine saubere Trennung für die Modellentwicklung zu gewährleisten. In welche Teilmenge ein Bild kommt wird schon bei der Blender Szene entschieden. Diese sind ebenfalls in die Verzeichnisse «train», «val» und «test» unterteilt. Ist eine Blender Szene im Testverzeichnis kommen die generierten Bilder automatisch in den Testdatensatz.

## 3.2 Erstellung des Modells

Zur Entrauschung der gerenderten Bilder wurde ein Autoencoder-Modell im Stil eines U-Net entwickelt. Dabei handelt es sich um ein neuronales Netzwerk, das speziell darauf ausgelegt ist, Bildinformationen zu verarbeiten und gleichzeitig unerwünschtes Rauschen zu entfernen. Die Struktur des Netzwerks ist symmetrisch aufgebaut: Ein Teil des Netzwerks (der Encoder) verkleinert schrittweise die Bildinformationen, um sie in einer kompakten Form zusammenzufassen. Der andere Teil (der Decoder) vergrössert diese Darstellung wieder, sodass am Ende ein rauschfreies Bild entsteht. Die in diesem Projekt verwendeten Netzwerkoperationen wurden mit der PyTorch-Bibliothek implementiert [14].

Das ursprüngliche U-Net wurde zwar für die semantische Segmentierung medizinischer Bilder entwickelt, eignet sich jedoch aufgrund seiner Architektur mit Encoder-Decoder-Struktur und Skip-Connections auch sehr gut für Bildrekonstruktionsaufgaben wie das Entrauschen. Diese Skip-Connections ermöglichen es, hochauflöste Bildinformationen aus dem Encoder direkt im Decoder zu nutzen und so feine Details im rekonstruierten Bild zu erhalten.

### 3.2.1 Modell Architektur

Der Encoder besteht aus mehreren Verarbeitungsschritten, bei denen das Bild immer weiter verkleinert wird, während gleichzeitig die wichtigsten Merkmale extrahiert werden. Das Ziel ist es, eine kompakte Darstellung zu erzeugen, die die wesentlichen Bildinhalte enthält. Diese Verdichtung endet in einem sogenannten «Bottleneck» (engl. für Flaschenhals), einem zentralen Abschnitt des Netzwerks, der nur eine stark reduzierte Version des Bildes enthält. Hier entscheidet sich, wie gut das Modell in der Lage ist, die Bildstruktur zu verstehen und später zu rekonstruieren.

Anschliessend beginnt der Decoder, der das Bild Schritt für Schritt wieder in die ursprüngliche Auflösung bringt. Dabei werden nicht nur die verdichteten Informationen aus dem Bottleneck verwendet, sondern auch sogenannte Skip-Connections. Diese verbinden frühere Verarbeitungsschritte des Encoders direkt mit passenden Stellen im Decoder. So kann das Modell auf Details zurückgreifen, die sonst beim Verkleinern des Bildes, verloren gehen würden. Man kann sich Skip-Connections wie Abkürzungen vorstellen, die es dem Netzwerk ermöglichen, wichtige Bildinformationen schneller und verlustfreier wiederzuverwenden. [15]

### **Vereinfachung der Architektur-Darstellung**

In der Architektur-Grafik des Modells wurden zur besseren Übersicht und Platzersparnis mehrere aufeinanderfolgende Operationen pro Layer zusammengefasst. Im Encoder-Teil wurde jeweils die Doppelfaltung (Double Convolution) zur Erhöhung der Feature-Kanäle mit dem darauffolgenden Max-Pooling kombiniert, das die räumliche Auflösung halbiert. Im Decoder-Teil wurde entsprechend die Transponierte Faltung (ConvTranspose), welche die Auflösung verdoppelt, gemeinsam mit der anschliessenden Double Convolution zur Halbierung der Feature-Kanäle dargestellt.

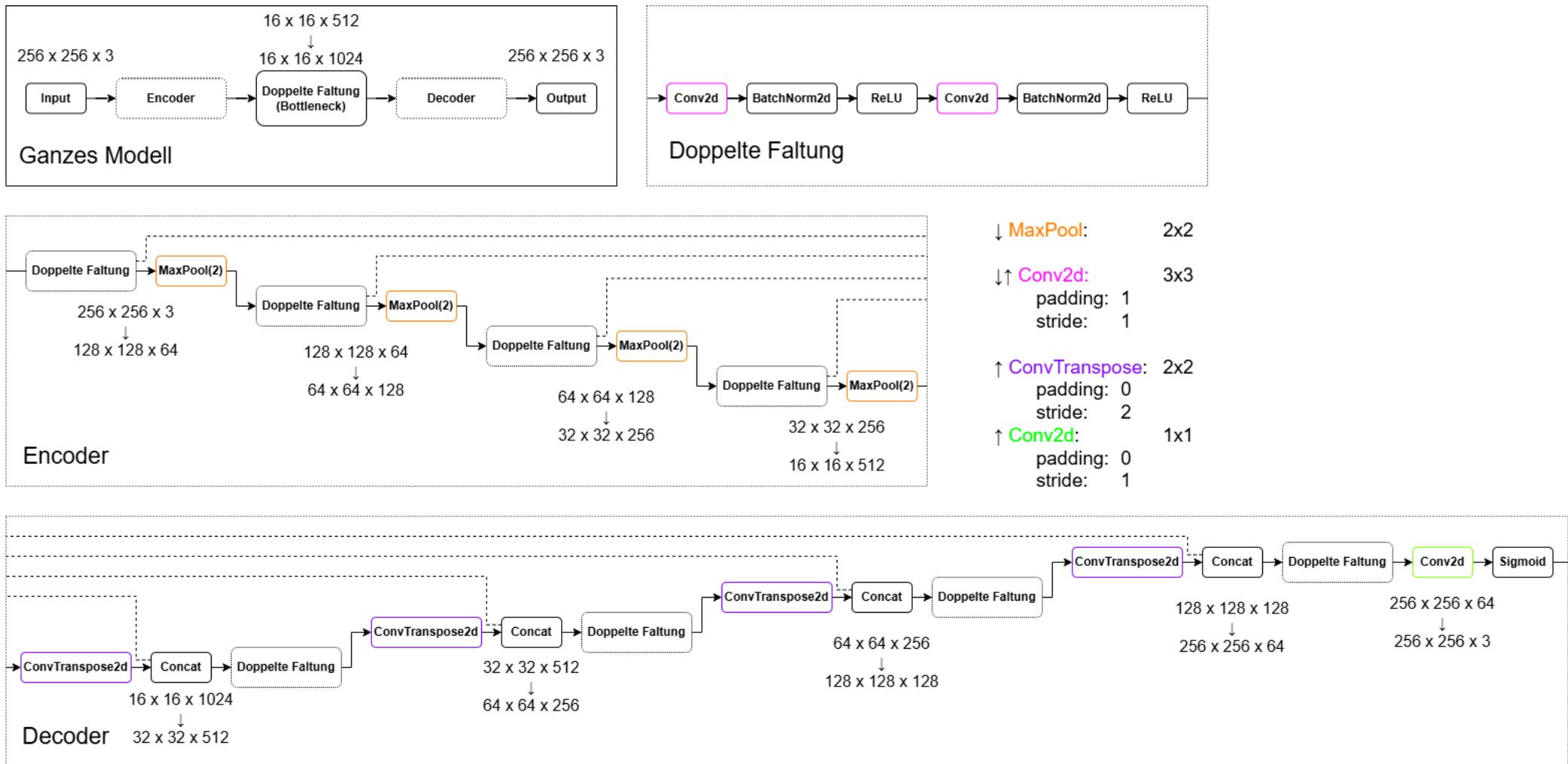


Abbildung 7: Vollständige Architektur des Autoencoder-Modells

### Encoder (Downsampling-Pfad):

Jede Encoder Stufe besteht aus zwei Teilen:

1. **Doppelte Faltung:** Zwei aufeinanderfolgende 2D-Faltungsschichten (Conv2d), jeweils gefolgt von Batch Normalization und einer ReLU-Aktivierungsfunktion. Diese Kombination wird verwendet, um robuste Merkmalsrepräsentationen zu extrahieren. Verdoppelt die Anzahl Feature-Kanäle, also die räumliche Tiefe.
2. **Max-Pooling:** Eine MaxPooling-Schicht reduziert die räumliche Auflösung um den Faktor 2, um schrittweise höherdimensionale Merkmale zu extrahieren.

### Decoder (Upsampling-Pfad):

Der Decoder spiegelt den Encoder und besteht ebenfalls aus vier Stufen. Jede Stufe umfasst:

1. **Transponierte Faltung** (ConvTranspose2d): Dient zum Upsampling der Feature Maps und zur Wiederherstellung der räumlichen Auflösung.
2. **Verknüpfung** (Concat): Die hochskalierten Feature Maps werden mit den entsprechenden Feature Maps aus dem Encoder (Skip-Connections) verknüpft, um Kontextinformationen aus früheren Stufen beizubehalten. Die Skip-Connections sind im Diagramm mit gestrichelten Linien gekennzeichnet.
3. **Doppelte Faltung:** Identisch aufgebaut wie im Encoder, um die kombinierten Merkmale weiter zu verarbeiten.

Im Folgenden werden die zentralen Bausteine der Netzwerkarchitektur im Detail beschrieben, jeweils mit ihrer Funktion und Bedeutung im Kontext des Denoising-Modells [16].

Die **Conv2d**-Schicht ist eine grundlegende Operation in Convolutional Neural Networks (CNNs), die ein kleines Faltungsfilter (Kernel) über das Eingabebild oder Feature Map verschiebt. Dabei wird lokal ein gewichtetes Mittel berechnet, was einer Merkmalsextraktion entspricht.

- **Zweck:** Extraktion lokaler Bildmerkmale wie Kanten, Texturen oder Formen.
- **Parameter:** Kernelgrösse (z. B. 3x3), Anzahl der Filter (Kanäle), Padding (zur Größenanpassung), Stride (Schrittweite).
- **Ausgabe:** Eine Feature Map pro Filter, wobei die Anzahl der Ausgabekanäle der Anzahl der Filter entspricht.

Die **MaxPool2d**-Schicht reduziert die räumliche Auflösung der Eingabe (Downsampling), indem sie aus nicht überlappenden Regionen (z. B. 2x2) jeweils das Maximum auswählt.

- **Zweck:** Verkleinerung der Feature Maps zur Reduktion der Rechenlast und zur Fokussierung auf dominante Merkmale.
- **Vorteile:** Effizientere Modellstruktur durch weniger Parameter und erhöhte Robustheit gegenüber Verschiebungen im Eingaberaum.

**BatchNorm2d** normalisiert die Aktivierungen innerhalb eines Mini-Batches auf Kanalbasis. Dies stabilisiert das Training, beschleunigt die Konvergenz und reduziert interne Verteilungsverschiebungen [17].

- **Zweck:** Normalisierung der Zwischenergebnisse (Mittelwert  $\approx 0$ , Varianz  $\approx 1$ ).
- **Effekt:** Sorgt für stabilere Gradienten und erlaubt höhere Lernraten.

Die **ReLU**-Aktivierungsfunktion ist definiert als:

$$\text{ReLU}(x) = \max(0, x)$$

Sie führt eine nichtlineare Transformation ein und sorgt dafür, dass das Netzwerk komplexe Zusammenhänge lernen kann.

- **Zweck:** Einführen von Nichtlinearität.
- **Eigenschaften:** Schnell berechenbar, keine Sättigung im positiven Bereich.

Die **Sigmoid-Funktion** ist eine Aktivierungsfunktion, die Werte aus dem gesamten reellen Zahlenbereich in einen Bereich zwischen 0 und 1 abbildet. Sie hat die folgende mathematische Form:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Diese S-förmige Funktion eignet sich besonders gut, um Wahrscheinlichkeiten oder Normalisierungen in neuronalen Netzen darzustellen, da sie den Ausgangswert auf einen festen Bereich einschränkt.

#### Eigenschaften:

- Ausgabe liegt immer im Intervall (0, 1).
- Nichtlinear, ermöglicht komplexe Modellierungen.
- Nachteil: Kann bei grossen Eingabewerten zu «Vanishing Gradients» führen, wodurch das Training erschwert wird.

Im Kontext von Denoising-Modellen kann die Sigmoid-Funktion im Ausgangslayer genutzt werden, um die Pixelwerte in einem normierten Bereich (z. B. [0, 1]) zu halten.

Folgende Verlustfunktion (Loss Function) wurde für das Training verwendet:

$$0.9 \cdot L1 + 0.1 \cdot (1 - ssim)$$

Für die Verlustfunktion wurde eine Kombination aus L1-Verlust und SSIM verwendet. Der L1-Verlust minimiert die absoluten Unterschiede zwischen dem vorhergesagten und dem Ground-Truth-Bild auf Pixelebene und sorgt somit für eine genaue Rekonstruktion. Der SSIM-Term hingegen bewertet die strukturelle Ähnlichkeit zwischen Bildern und fördert die Erhaltung von Texturen und Details, die für das menschliche Auge relevant sind. Die Kombination beider Verluste ermöglicht es, sowohl eine präzise als auch visuell ansprechende Entrauschung zu erzielen, was in der Literatur als effektive Strategie für Bildrestaurationsaufgaben beschrieben wird [18].

### 3.2.2 Entwicklung der Modellarchitektur

Zu Beginn der Implementation wurde ein einfacher Autoencoder als Ausgangspunkt für das Denoising-Modell implementiert. Diese frühe Version bestand aus einem sequenziellen Encoder-Decoder-Aufbau mit drei Convolution-Blöcken im Encoder und drei entsprechenden Transposed-Convolution-Blöcken im Decoder. Die Architektur war bewusst schlicht gehalten, um schnelle Experimente zur Verlustfunktion und zum Trainingsverhalten zu ermöglichen. Sie verzichtete dabei auf Skip-Connections, Batch Normalization und tiefere Netzwerkstrukturen. Dies führte jedoch zu suboptimalen Ergebnissen bei komplexeren Störungen, wie sie im Path-Tracing-Rauschen auftreten. Die finale Architektur basiert auf einer U-Net-ähnlichen Struktur, die speziell auf das Rauschverhalten von path-traced Bildern zugeschnitten wurde. Wesentliche Weiterentwicklungen umfassen:

- **Tiefere Netzwerkstruktur:** Die endgültige Version nutzt vier Encoder- und vier Decoder-Blöcke mit zunehmender Kanalanzahl pro Ebene. Dadurch kann das Modell feinere Details und kontextuelle Informationen auf verschiedenen Auflösungsebenen erfassen.
- **DoubleConv-Blöcke mit Batch Normalization:** Jeder Block besteht aus zwei Faltungen, jeweils gefolgt von Batch Normalization und ReLU-Aktivierung. Diese Kombination verbessert die Stabilität des Trainings und die Fähigkeit des Netzwerks, komplexe Bildstrukturen zu modellieren.
- **Skip-Connections:** Anders als im ursprünglichen Modell werden in der finalen Architektur Merkmalskarten aus dem Encoder direkt an die korrespondierenden Decoder-Blöcke weitergegeben. Dies unterstützt die Rekonstruktion feiner Details und hilft dabei, Informationen zu erhalten, die durch Pooling-Schritte verloren gehen könnten.
- **Bottleneck mit hoher Kapazität:** Der tiefste Punkt des Netzes besteht aus einem zusätzlichen DoubleConv-Block mit erhöhter Kanalanzahl. Dadurch kann das Netzwerk globale Bildkontakte besser verarbeiten.
- **Flexibilität über Parameter base\_channels:** Die Kanalbreiten sind parametrisiert, was ein einfaches Skalieren der Modellgrösse ermöglicht.

Zusammenfassend lässt sich sagen, dass die finale Architektur durch ihre Tiefe, die Verwendung von Skip-Connections und strukturierte Blockaufteilung eine deutlich höhere Kapazität besitzt, um die

komplexe Aufgabe des Bilddenoisings zu bewältigen. Sie stellt damit eine gezielte Weiterentwicklung der initialen, prototypischen Autoencoder-Variante dar.

Dies sind die Ergebnisse auf dem Testdatensatz mit der alten Modellarchitektur:

Metrik	Verrauscht (Eingabe)	Denoised (Modell)
L1-Loss	0.064722	0.026447
SSIM	0.405321	0.888432

Tabelle 3: Verlustmetriken auf der alten Modellarchitektur

Visuell ist das Modell inkonsistent. Folglich ein gutes und ein schlechtes Beispiel:



Abbildung 8: Schlechtes visuelles Beispiel mit der alten Modellarchitektur

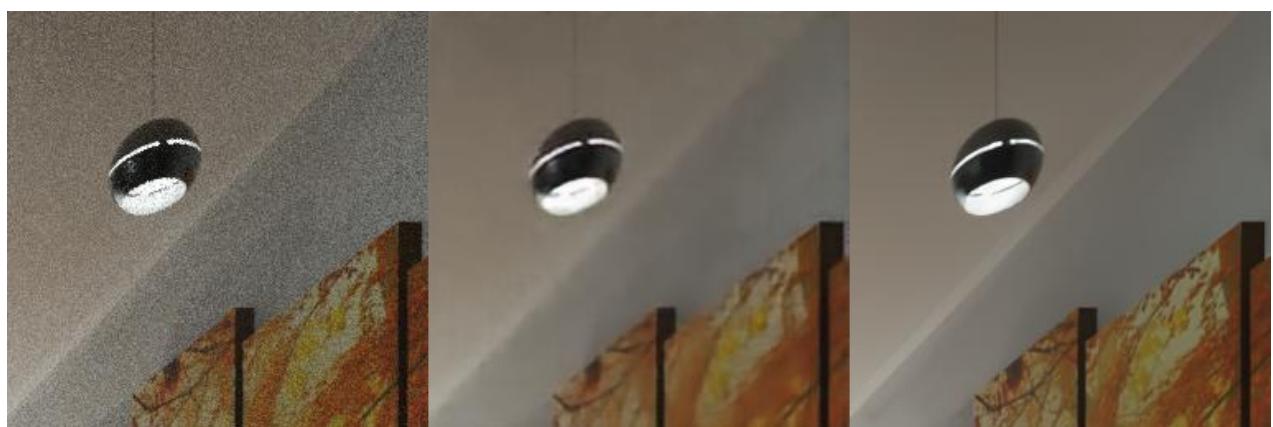


Abbildung 9: Gutes Ergebnis der alten Modellarchitektur im Vergleich

### 3.2.3 Optimierung der Hyperparameter

Um das Modell möglichst effektiv zu trainieren, wurde die Optimierungsbibliothek Optuna eingesetzt. Sie hilft dabei, wichtige Hyperparameter des Trainings automatisch zu testen und zu verbessern. Dazu gehören unter anderem die Batch-Grösse (wie viele Bilder gleichzeitig verarbeitet werden), die Lernrate (wie schnell das Modell lernt), die Anzahl der Kanäle in den Verarbeitungsschritten (also wie viele unterschiedliche Merkmale betrachtet werden) sowie weitere Hyperparameter.

Ein Optimierungsalgorithmus steuert, wie das Modell während des Trainings seine Parameter anpasst, um Fehler zu minimieren. Dabei wird der Adam-Optimizer verwendet, ein verbreiteter Algorithmus, der die Lernrate adaptiv anpasst und so schneller und stabiler zum Optimum konvergiert. Optuna automatisiert die Suche nach den besten Trainingsparametern, um die Leistung des Modells zu verbessern.

Bei der automatisierten Optimierung mit Optuna wurden gezielt verschiedene Hyperparameter getestet. Diese bestimmen grundlegende Eigenschaften des Trainingsprozesses und der Modellstruktur. Die wichtigsten dabei waren:

- **Batch-Grösse** (batch\_size): Gibt an, wie viele Bilder gleichzeitig verarbeitet werden. Getestet wurden Werte von 32 und 64.
- **Lernrate** (lr): Bestimmt, wie stark das Modell seine Gewichte bei jedem Lernschritt anpasst. Werte wurden im Bereich von 0.00001 bis 0.01 ausprobiert (in logarithmischen Abstufungen).
- **Anzahl der Basis-Kanäle** (base\_channels): Gibt die Breite der ersten Verarbeitungsschicht an und beeinflusst damit direkt die Tiefe und Kapazität des Modells. Es wurden 64 und 128 Kanäle getestet.
- **Beta1 und Beta2** (beta1, beta2): Diese beiden Parameter gehören zum verwendeten Adam-Optimierer und regeln, wie stark frühere Gradientenverläufe berücksichtigt werden. Die getesteten Bereiche lagen bei:
  - Beta1: zwischen 0.8 und 0.99
  - Beta2: zwischen 0.9 und 0.999
- **Epsilon** (eps): Eine kleine Zahl, die zur Vermeidung von Division durch Null dient. Getestet wurden Werte zwischen 1e-9 und 1e-6.
- **Gewichtsverfall** (weight\_decay): Eine Form der Regularisierung, die sehr grosse Gewichtswerte vermeidet und damit Overfitting entgegenwirkt. Der Bereich lag zwischen 1e-6 und 1e-2.

Jede Kombination dieser Werte wurde von Optuna bewertet, indem sie mit einem Trainingslauf getestet wurde. So konnte die beste Zusammenstellung gefunden werden, ohne dass man alle denkbaren Kombinationen manuell ausprobieren musste. Dieses Verfahren sparte nicht nur viel Zeit, sondern führte auch zu besseren Ergebnissen, als es mit festen Standardwerten möglich gewesen wäre.

Folgende Werte wurden von Optuna als beste Hyperparameter gefunden.

```
{
  "batch_size": 32,
  "lr": 0.00010135211376349008,
  "base_channels": 64,
  "beta1": 0.9175359182964491,
  "beta2": 0.9757206708695416,
  "eps": 3.422801253803442e-09,
  "weight_decay": 9.692445546369089e-05
}
```

Das gesamte Hyperparameter-Tuning dauerte etwa neuneinhalb Stunden. Insgesamt wurden 56 Trainingsläufe von Optuna gestartet. Viele davon brach Optuna jedoch frühzeitig ab, da sich bereits in den ersten Epochen abzeichnete, dass ihre Leistung schlechter war als die vorherigen Läufe. Wurde das Training beispielsweise durch einen Verbindungsverlust zum MLMP-Server unterbrochen, so wurde automatisch ein neuer Trainingslauf mit denselben Hyperparametern in die Warteschlange gestellt.

### 3.2.4 Schutz vor Overfitting

Ein wichtiges Ziel beim Training eines neuronalen Netzwerks ist es, dass es nicht nur die Trainingsbilder auswendig lernt, sondern auch auf neue, unbekannte Bilder gute Ergebnisse liefert. Dieses Problem nennt man Overfitting (engl. für Überanpassung). Um dem entgegenzuwirken, wurden mehrere Strategien eingesetzt:

- **Validierungsdaten:** Ein Teil der Bilddaten wurde nicht zum Lernen verwendet, sondern nur zur Überprüfung, wie gut das Modell auf unbekannten Bildern funktioniert. Der Datensatz wurde in 76 % Training-, 14 % Validierungs- und 10 % Testbilder aufgeteilt.

- **Frühzeitiges Abbrechen:** Wenn sich die Qualität auf den Validierungsdaten über längere Zeit nicht verbessert hat, also die Verlustfunktion auf dem Validierungsdatensatz sich über mehrere Epochen nicht kleiner wurde, wurde das Training automatisch gestoppt.
- **Regulierung der Gewichte (Weight Decay):** Das Netzwerk wurde leicht bestraft, wenn es sehr extreme Werte für seine internen Verbindungen verwendet hat. Diese Technik, auch als Weight Decay bekannt, sorgt dafür, dass die Gewichte kleiner bleiben, wodurch das Modell stabiler wird und sich besser auf neue, unbekannte Daten generalisieren kann.
- **Vielfalt durch unterschiedliche Rauschstufen:** Für jedes Ground-Truth-Bild wurden acht verschiedene Versionen mit variierender Rauschintensität (unterschiedliche Sample-Anzahlen pro Pixel) generiert. Diese Vielfalt an Eingangsbildern hilft dem Netzwerk, nicht auf ein bestimmtes Rauschmuster zu überfitten, sondern robust gegenüber verschiedenen Rauschstärken zu werden.

### 3.3 Modell Training

Das Training des Denoising-Autoencoders erfolgte auf einer leistungsstarken Serverplattform mit einer NVIDIA A100 Grafikkarte, die speziell für rechenintensive Aufgaben wie das Training neuronaler Netzwerke ausgelegt ist. Als Softwaregrundlage wurde PyTorch in der Version 2.6.0 verwendet, das auf CUDA 12.6 basiert, um die GPU-Beschleunigung optimal zu nutzen.

Während des Trainings wurden alle relevanten Messwerte und Fortschritte mit TensorBoard (Version 2.19.0) protokolliert. Dies ermöglichte eine übersichtliche Visualisierung von Kennzahlen wie Verlustwerten und Lernraten während des Trainings und erleichterte so die Überwachung und Optimierung des Modells.

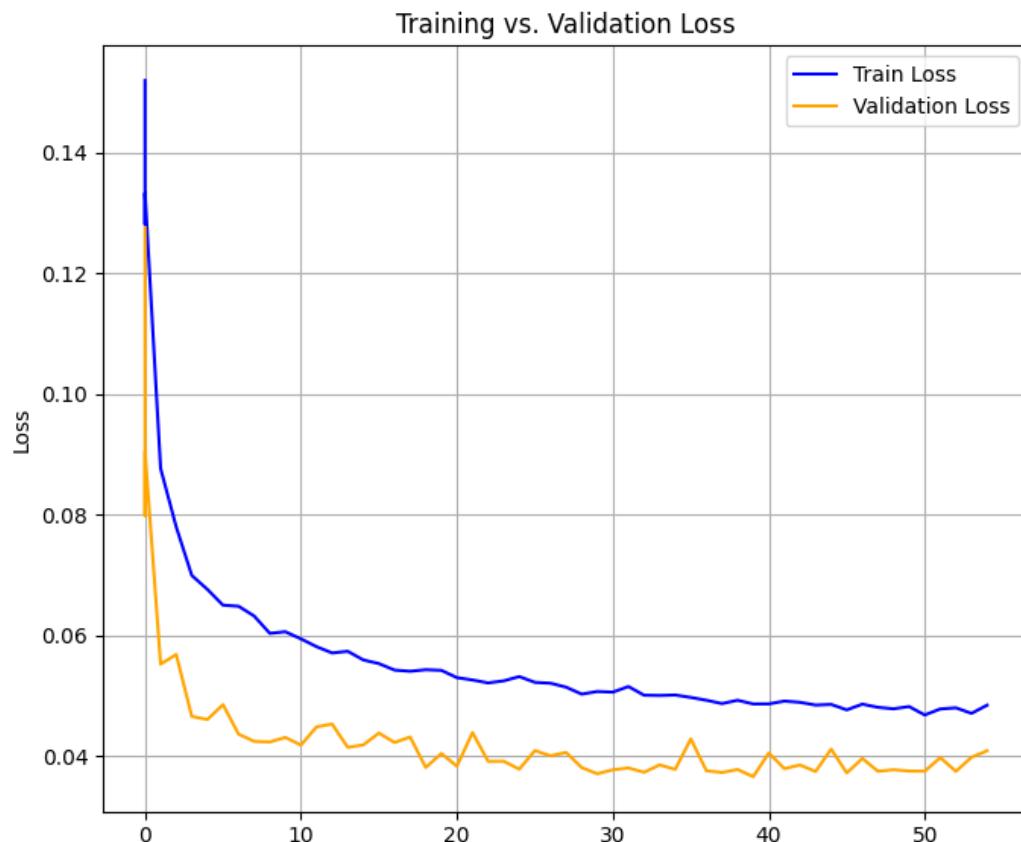


Abbildung 10: Verlauf des Training- und Validierungsverlusts

Die Abbildung zeigt den Verlauf des Trainings- und Validierungsverlusts über die Trainingsschritte hinweg. Beide Kurven basieren auf Werten, die während des Trainingsprozesses mithilfe von TensorBoard aufgezeichnet wurden. Ein niedrigerer Verlustwert deutet auf eine bessere

Modellleistung hin, da der Unterschied zwischen dem vorhergesagten und dem tatsächlichen Bild geringer ist. Idealerweise nähern sich beide Kurven mit fortschreitendem Training einem stabilen, niedrigen Wert an. Obwohl der Trainingsverlust am Ende sich weiter verbessert, wurde das Training wegen dem stagnierten Validierungsverlusts gestoppt.

Das Training dauerte 94 Minuten und brauchte die gesamten 80 GB VRAM der MLMP A100 Grafikkarte. Das bedeutet aber nicht, dass man 80 GB VRAM benötigt, um das Training durchzuführen. Teile des Speichers wurden wahrscheinlich für Caching verwendet und womöglich wurden bereits verwendete Bilder sofort aus dem VRAM gelöscht. Vermutlich kann das Training auf einer RTX4090 unverändert durchgeführt werden.

Schliesslich kann das Training mit lediglich folgenden Änderungen auf meiner RTX3070 8GB laufen:

- Batch\_size: 16
- Base\_channels: 32

Für Leserinnen und Leser, die das Modell selbst trainieren oder anpassen möchten, sind im zugehörigen GitHub-Repository [13] detaillierte Anleitungen zur Installation und Einrichtung der benötigten Softwarekomponenten im README-Dokument enthalten.

### 3.4 Grafische Anwendung

Im Rahmen dieser Arbeit wurde eine grafische Benutzeroberfläche (GUI) entwickelt, die es ermöglicht, beliebig grosse, verrauschte Bilder einzulesen, automatisch zu entrauschen und die Resultate direkt vergleichend anzuzeigen. Hierbei wird das Bild in passende Teilstücke (Patches) zerlegt, welche einzeln durch das trainierte neuronale Modell zum Denoising verarbeitet werden. Anschliessend werden die einzelnen Teile wieder zusammengesetzt, um das vollständige entrauschte Bild zu erhalten. Diese Vorgehensweise ermöglicht die Verarbeitung von Bildern beliebiger Grösse, ohne die Beschränkungen der Netzwerkarchitektur bezüglich der Eingabegrösse.

Das GUI zeigt jeweils entweder das originale verrauschte Bild oder das entrauschte Bild im Vollbildmodus an. Mit beliebigen Tastendrücken kann zwischen diesen beiden Darstellungen umgeschaltet werden. Zusätzlich stehen Zoom- und Verschiebefunktionen zur Verfügung, um Details an beliebiger Bildstelle genauer zu betrachten.

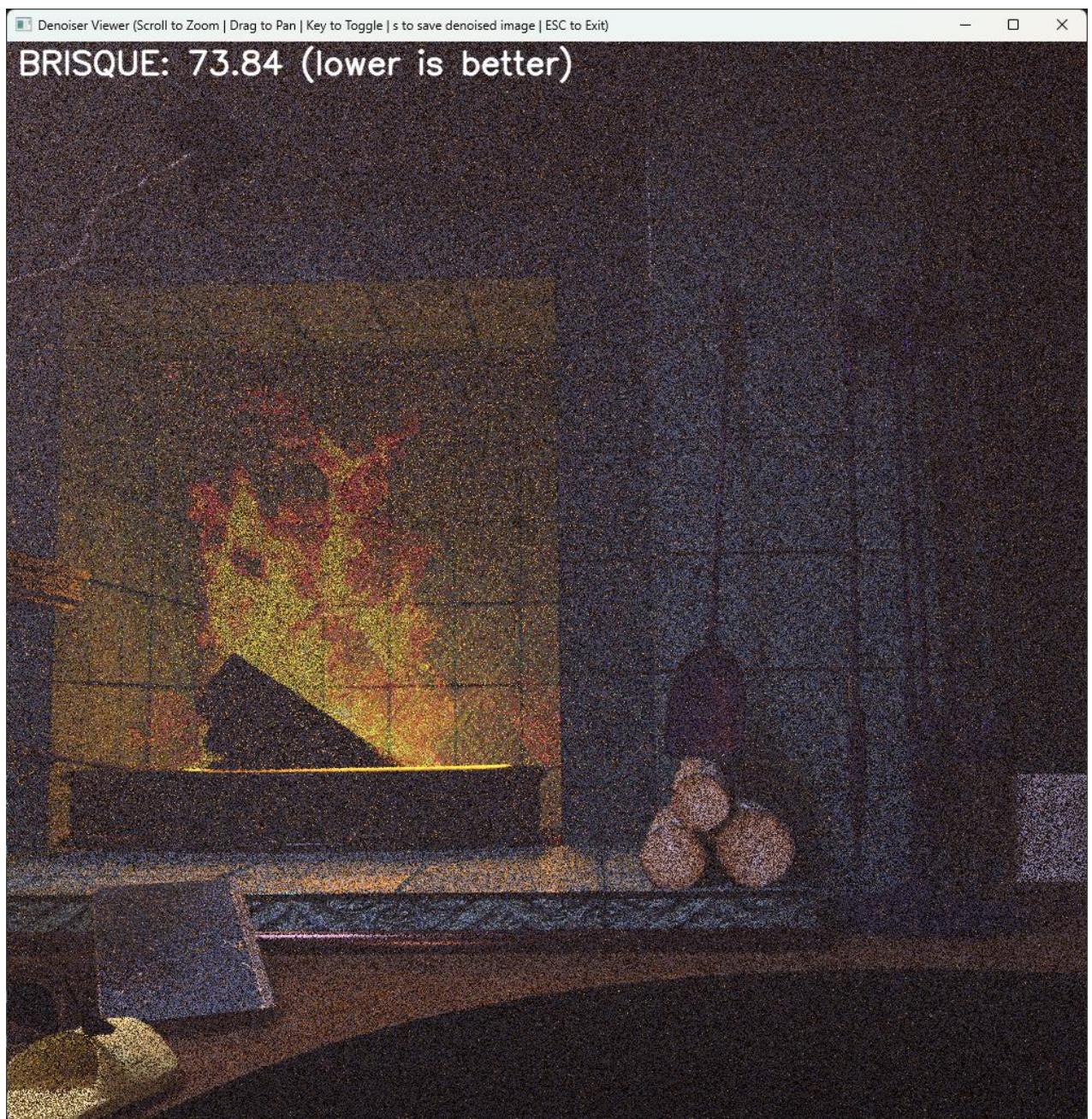


Abbildung 11: Beispiel der grafischen Anwendung mit einem 1024 x 1024 Pixel, verrauschem Bild mit 200 Samples per Pixel. Demo-Szene von Blender [19], erstellt von Glenn Melenhorst, verwendet unter Annahme einer CC-BY-Lizenz



Abbildung 12: Die grafische Applikation mit dem entrauschten Bild. Demo-Szene von Blender [19], erstellt von Glenn Melenhorst, verwendet unter Annahme einer CC-BY-Lizenz

Zu beachten ist der niedrigere BRISQUE-Wert der entrauschten Version im Vergleich zum verrauschten Bild. Dies spiegelt sich auch visuell in einer deutlichen Reduktion des Bildrauschens wider.

Um die visuelle Qualitätsverbesserung in der grafischen Anwendung objektiv zu messen, wird neben der subjektiven Betrachtung die BRISQUE-Metrik eingesetzt. BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator) ist ein referenzfreies Bildqualitätsmaß, das ohne Vergleich zum Originalbild die wahrgenommene Bildqualität quantifiziert. Dies ist besonders wichtig, da die GUI-Anwendung so ausgelegt ist, dass Nutzer beliebige neue Bilder zur Entrauschung eingeben können, für die kein sauberes Referenzbild vorliegt. Für den Vergleich mit einem Referenzbild kann stattdessen das Testing-Skript verwendet werden.

Durch die Berechnung von BRISQUE-Werten für das verrauschte und das entrauschte Bild kann die Verbesserung der Bildqualität numerisch erfasst und dem Nutzer angezeigt werden. Ein niedrigerer BRISQUE-Wert entspricht dabei einer besseren Bildqualität.

Obwohl BRISQUE ursprünglich für natürliche, fotografische Bilder entwickelt und trainiert wurde, lässt es sich dennoch gut als referenzfreies Qualitätsmaß für die gerenderten Bilder aus dem Path Tracing einsetzen. Da diese Bilder spezielle Rausch- und Strukturmerkmale aufweisen, ist BRISQUE nicht perfekt auf sie abgestimmt. Dennoch ermöglicht die Metrik, für beliebige neue Eingabebilder — bei denen keine sauberen Referenzen vorliegen — eine objektive Abschätzung der visuellen Qualitätsverbesserung durch das Entrauschen.

Darüber hinaus bietet die Anwendung die Möglichkeit, das entrauschte Bild auf Knopfdruck über den Datei-Explorer zu speichern.

### 3.4.1 Inferencing-Performance

Nach Auswahl eines Bildes mit beliebiger Auflösung über den Dateiexplorer beginnt automatisch die Inferenz durch das trainierte Modell. Dieser Prozess läuft vollständig auf der Grafikkarte ab. Auf meinem System mit einer NVIDIA RTX 3070 ergeben sich dabei folgende Verarbeitungszeiten:

- **1024 × 1024 Bild:** ca. 47 Millisekunden
- **3840 × 2160 Bild (4K):** ca. 1,92 Sekunden

Damit ist die Anwendung auch für hochauflösende Bilder in akzeptabler Zeit nutzbar.

### 3.4.2 Behandlung von Artefakten an Patch-Grenzen

Beim Denoising grosser Bilder wird das Eingabebild in kleinere Patches (in diesem Fall 256×256 Pixel) unterteilt, um diese effizient einzeln durch das neuronale Netzwerk zu verarbeiten. Eine naive Zusammensetzung dieser Patches führt jedoch häufig zu sichtbaren Artefakten an den Patch-Grenzen. Diese entstehen, weil das Netzwerk in jedem Patch unabhängig arbeitet und keine Information über benachbarte Bildbereiche hat. Besonders an den Rändern der Patches kann dies zu Inkonsistenzen und Kantenartefakten führen.

Um dieses Problem zu lösen, wurde eine Überlappung (Overlap) zwischen den Patches eingeführt. Jeder Patch überschneidet sich mit den angrenzenden Patches um einige Pixel (in diesem Fall 16 Pixel). Dadurch kann das Netzwerk die Bildinhalte an den Übergängen konsistenter rekonstruieren, da es in den überlappenden Bereichen zusätzliche Kontextinformation erhält.

Zusätzlich zur Überlappung wurde eine gewichtete Mittelung der Patch-Ausgaben mit einem Hanning-Fenster durchgeführt. Dieses Fenster gewichtet die Mitte des Patches stärker als die Ränder. Beim Zusammensetzen der Ausgabepatches werden die überlappenden Bereiche daher weich überblendet, sodass harte Übergänge vermieden und visuelle Artefakte reduziert werden. Diese Technik führt zu deutlich glatteren und natürlich wirkenden Ergebnissen, ohne dass die Patchgrenzen im finalen Bild sichtbar sind.

Folglich ein Beispiel eines entrauschten Bildes vor der Behandlung der Artefakte:

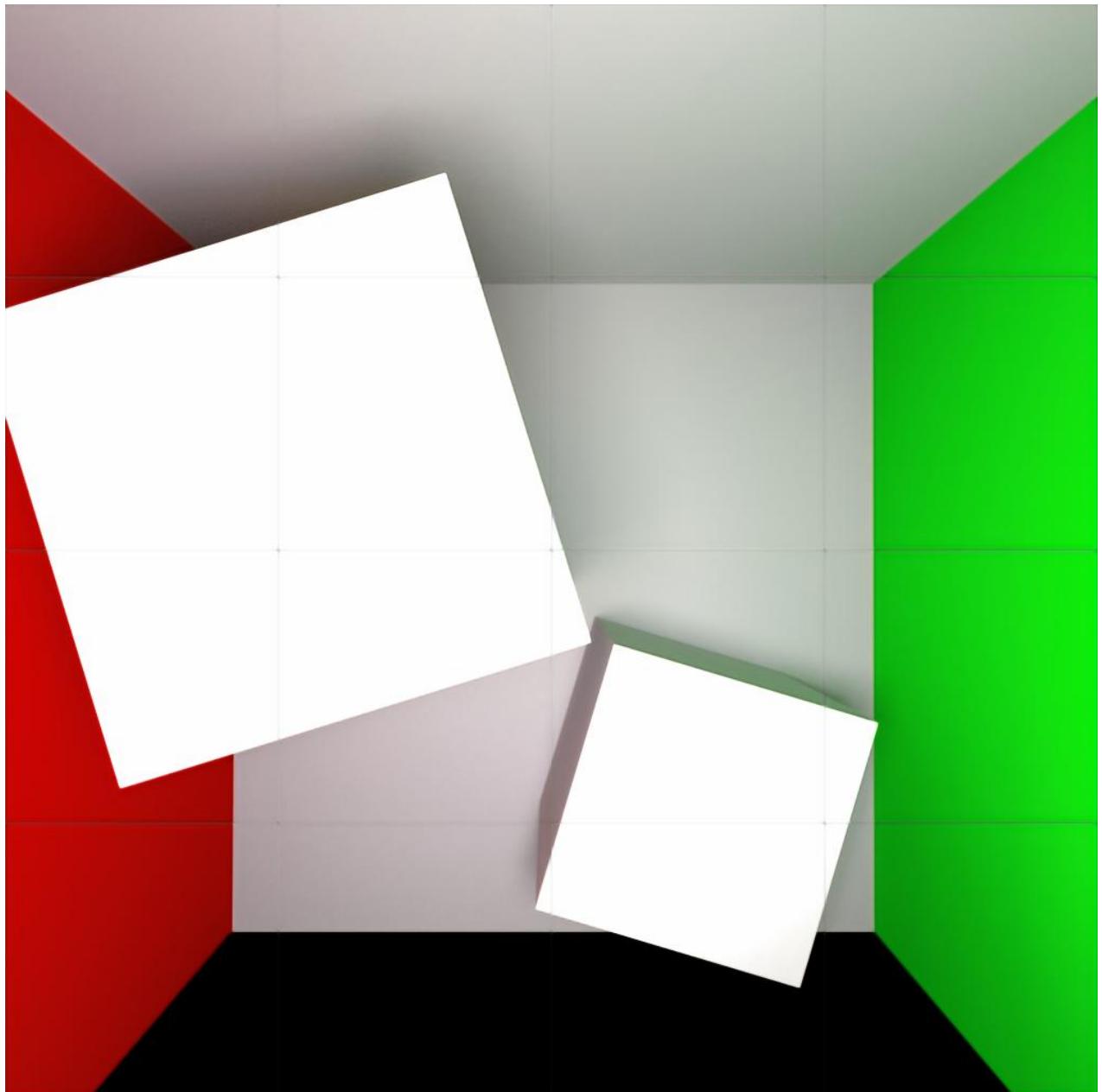


Abbildung 13: Entrausches Bild mit Patch-Artefakten (behoben)

## 4 Resultate

Zur Bewertung der Leistungsfähigkeit des trainierten neuronalen Netzwerks wurde ein separater Testdatensatz verwendet, der während des Trainingsprozesses nicht gesehen wurde. Ziel war es, die Bildqualität der vom Denoiser erzeugten Bilder sowohl quantitativ als auch visuell mit den ursprünglichen verrauschten Eingabebildern zu vergleichen.

### 4.1 Bewertungsmethoden

Zur objektiven Bewertung der Leistungsfähigkeit des entwickelten Entrauschungsmodells wurde ein automatisiertes Testskript implementiert, welches das trainierte Modell auf dem gesamten Testdatensatz ausführt und Qualitätsmetriken berechnet. Das Skript lädt dazu alle Testbilder mit ihrem zugehörigen sauberen Referenzbild, führt eine Inferenz mit dem ONNX-Modell durch und wertet die Ergebnisse aus.

Die Qualität der Bildrekonstruktion wurde anhand zweier gängiger Metriken beurteilt:

- L1-Loss (Mean Absolute Error): Diese Metrik misst die durchschnittliche absolute Differenz zwischen den Pixelwerten des verrauschten bzw. entrauschten Bildes und des sauberen Referenzbildes. Ein niedrigerer Wert entspricht einer höheren Genauigkeit und besseren Denoising.
- SSIM (Structural Similarity Index Measure): Diese Metrik bewertet, wie ähnlich zwei Bilder in Bezug auf Struktur, Helligkeit und Kontrast sind. Der SSIM-Wert liegt zwischen 0 und 1, wobei 1 perfekte Übereinstimmung bedeutet. Im Gegensatz zu einfachen Metriken wie dem L1-Loss berücksichtigt SSIM auch die wahrgenommene visuelle Qualität und Struktur im Bild.

### 4.2 Visuelle Ergebnisse

Zusätzlich zur numerischen Auswertung wurden Beispielbilder visuell verglichen. Dabei wurden pro Szene jeweils das verrauschte Eingabebild, das vom Modell bearbeitete Bild sowie das Ground-Truth-Bild gegenübergestellt. Zudem werden die Ergebnisse tabellarisch in einer CSV-Datei gespeichert, um eine spätere Analyse zu erleichtern.

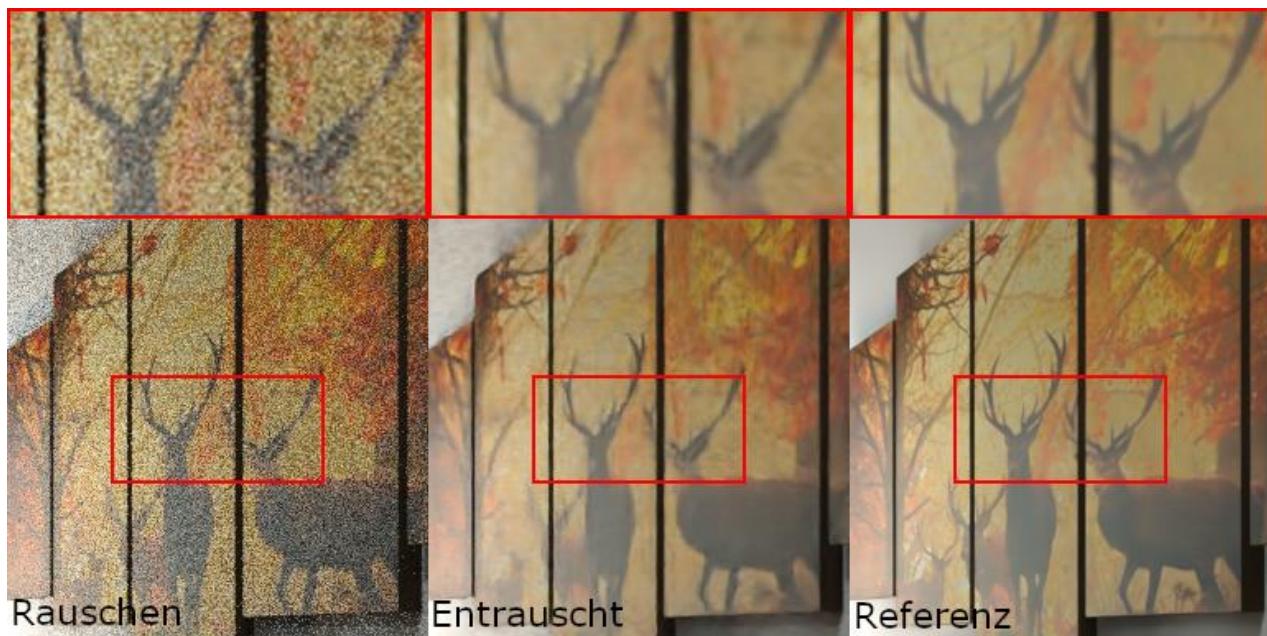


Abbildung 14: Visueller Unterschied zwischen Verrauscht (200 Samples), Entrauscht und Referenz

Diese visuellen Beispiele bestätigen den quantitativen Eindruck: Das Modell kann das Rauschen deutlich reduzieren und gleichzeitig wichtige Bilddetails erhalten.

Im Anhang sind weitere Ergebnisse zu finden, darunter auch ein Beispiel am SLProject4 Path Tracer.

### 4.3 Quantitative Ergebnisse

Die Ergebnisse zeigen, dass das trainierte Modell eine deutliche Verbesserung der Bildqualität erzielt hat. Die folgenden Durchschnittswerte wurden auf dem Testdatensatz gemessen:

Metrik	Verrauscht (Eingabe)	Denoised (Modell)
Durchschnittlicher L1-Loss	0.064722	0.019576
Durchschnittlicher SSIM	0.405321	0.900885

Tabelle 4: Verlustfunktion Verrauscht vs. Entrauscht

Diese Zahlen bedeuten konkret: Der durchschnittliche Unterschied zum idealen Bild konnte um etwa **70 % verringert** werden. Gleichzeitig stieg die strukturelle Ähnlichkeit der Bilder auf einen hohen Wert von über **0.90**, was auf eine sehr gute visuelle Übereinstimmung hinweist. Verglichen mit den verrauschten Bildern entspricht dies einer Steigerung des SSIM-Werts um etwa 122 %.

Folglich auf der nächsten Seite die vollständigen Testresultate des gesamten Testdatensatzes.

Comparison of L1 Loss and SSIM Before and After Denoising

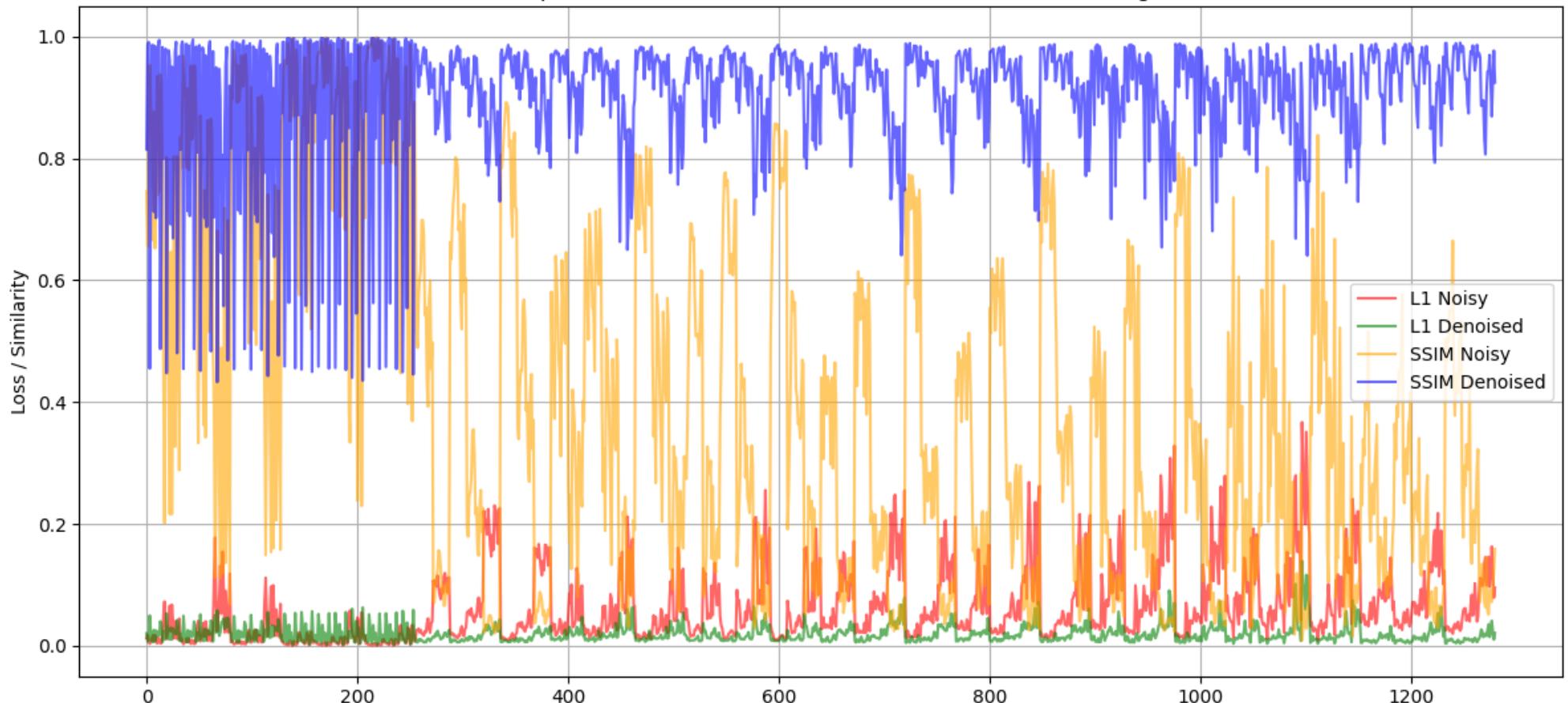


Abbildung 15: Qualitätsmetriken auf dem Testdatensatz

Dieser Plot zeigt anschaulich die Verbesserung durch das Modell: Die roten und orangen Kurven repräsentieren die Werte vor dem Denoising (verrauscht Bild), während die grünen und blauen Kurven die Werte nach dem Denoising darstellen.

In der Regel ist eine deutliche Verbesserung zu erkennen – der L1-Loss (niedriger ist besser) sinkt signifikant, während der SSIM-Wert (höher ist besser) steigt. Dies weist auf eine verbesserte Bildqualität nach der Entrauschen hin.

Die ersten 256 Testbilder stammen aus der Szene *Cornell Box*. In dieser Szene scheint der integrierte Denoiser von Cycles vereinzelt unzuverlässige Ground-Truths erzeugt zu haben, was sich in stark schwankenden Werten widerspiegelt. Trotz dieser Ausreisser sind die Ergebnisse visuell dennoch überzeugend.

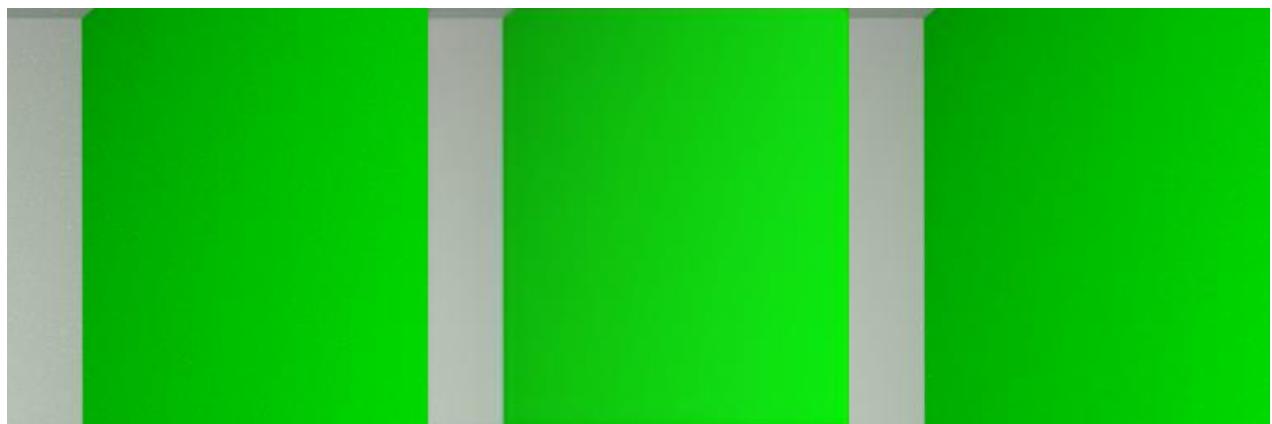


Abbildung 16: Statistischer Ausreisser. Von links nach rechts: Verrauscht, Entrauscht, Referenz

Zur qualitativen Beurteilung wurden für jedes Testbild Vergleichsbilder gespeichert, die das verrauschte Eingabebild, das vom Modell entrauschte Bild sowie das saubere Ground-Truth-Bild nebeneinander zeigen. Diese Darstellungen ermöglichen eine direkte visuelle Einschätzung der Modellleistung und bestätigen den positiven Effekt des Denoisings.

Im obigen Referenzbild ist ein leichtes Rauschen von blossem Auge zu erkennen. Dieses Rauschen ist bei der Entrauschten Version nicht vorhanden. Dies führte zum statistischen Ausreisser. Hier die Verlust-Metriken für diesen Patch:

Metrik	Verrauscht (Eingabe)	Denoised (Modell)
L1-Loss	0.002723	0.053406
SSIM	0.966870	0.455145

Tabelle 5: Verlustmetriken von einem statistischen Ausreisser

#### 4.4 Zielerreichung

Im Rahmen dieser Arbeit konnten die gesetzten Ziele grösstenteils erfolgreich umgesetzt werden:

Ziel	Status
Theoretische Untersuchung von Path Tracing und Denoising	✓
Datensatzgenerierung	✓
Entwicklung und Training eines Denoising-Modells	✓
Vergleich mit bestehenden Lösungen	✗
Integration des Denoising-Modells	✓

Tabelle 6: Zielerreichungstabelle

#### **4.4.1 Theoretische Untersuchung von Path Tracing und Denoising**

Ein fundiertes Verständnis der Path-Tracing-Technik wurde erarbeitet, insbesondere im Hinblick auf ihre Anwendung zur globalen Beleuchtung und die Entstehung von Bildrauschen durch Monte-Carlo-Stichproben. Zudem erfolgte eine umfassende Recherche zu aktuellen Denoising-Verfahren mit besonderem Fokus auf neuronale Netzwerke, wodurch die methodische Grundlage für die Entwicklung eines eigenen Modells gelegt wurde.

#### **4.4.2 Datensatzgenerierung**

Da bestehende Datensätze entweder qualitativ oder strukturell nicht auf die Anforderungen von Path-Tracing-Anwendungen zugeschnitten waren, wurde ein eigener Datensatz mit Blender erstellt. Dieser umfasst Renderings mit verschiedenen Sample-Raten sowie entsprechende Ground-Truth-Bilder und ermöglicht gezieltes Training auf verrauschte Eingaben. Das erstellte Skript ermöglicht eine leichte Erweiterung des Datensatzes.

#### **4.4.3 Entwicklung und Training eines Denoising-Modells**

Ein U-Net-ähnliches Autoencoder-Modell wurde erfolgreich implementiert und auf den erzeugten Datensätzen trainiert. Dabei kamen sowohl visuelle als auch quantitative Metriken wie SSIM und L1-Verlust zur Evaluierung der Ergebnisse zum Einsatz. Das Modell zeigt vielversprechende Ergebnisse hinsichtlich der Rauschunterdrückung bei niedrig gesampelten Path-Tracing-Bildern.

#### **4.4.4 Vergleich mit bestehenden Lösungen**

Der ursprünglich geplante Vergleich mit bestehenden Denoising-Verfahren wie dem NVIDIA OptiX Denoiser oder den integrierten Blender-Denoisern konnte aus zeitlichen Gründen nicht mehr durchgeführt werden. Dies, da Abschnitte «Modell trainieren» und «Modell testen» länger dauerten als ursprünglich gedacht. Für die Erstellung des Datensatzes wurde ebenfalls zu wenig Zeit eingeplant. Zusätzlich fehlte in der Planung die Zeit für das Poster und das Book.

Ein solcher Vergleich wäre jedoch wertvoll, um die Leistungsfähigkeit des entwickelten Modells im Verhältnis zu etablierten Verfahren zu analysieren, und stellt daher eine sinnvolle Erweiterung für zukünftige Arbeiten dar.

#### **4.4.5 Integration des Denoising Modells**

Das Ziel wurde erreicht durch die Entwicklung einer Python-GUI-Anwendung unter Verwendung von OpenCV. Diese Anwendung ermöglicht es, Bilder beliebiger Größe automatisch zu entrauschen und den Vergleich zwischen Eingangsbild und Ausgangsbild visuell darzustellen. Somit wurde die praktische Einsetzbarkeit des entwickelten Modells in einem flexiblen Tool demonstriert.

## 5 Fazit

Ziel dieser Arbeit war es, den Einsatz neuronaler Netzwerke zur Rauschunterdrückung im Kontext von Path Tracing zu untersuchen. Dazu wurden zunächst die theoretischen Grundlagen von Path Tracing sowie die physikalischen und rechnerischen Ursachen des Bildrauschen aufgearbeitet. In einem eigenen Datensatz wurden mithilfe von Blender verrauschte und hochqualitative Bilder erzeugt. Darauf aufbauend wurde ein Deep-Learning-Modell im Stil eines U-Nets mit Skip-Connections trainiert, um aus verrauschten Eingabebildern eine entrauschte Version zu generieren.

Die Ergebnisse zeigen, dass das trainierte Modell in der Lage ist, das Bildrauschen signifikant zu reduzieren: Der durchschnittliche L1 Loss konnte von 0.0647 auf 0.0196 gesenkt werden – eine Reduktion um rund 70 %. Der SSIM-Wert, der die strukturelle Ähnlichkeit zum Referenzbild misst, stieg von 0.405 auf 0.901. Visuelle Vergleiche bestätigen diese quantitativen Ergebnisse und zeigen, dass das Modell nicht nur Rauschen unterdrückt, sondern auch wichtige Bilddetails erhält.

Auch wenn das Modell nicht mit bestehenden Industrie-Standards wie dem OptiX-Denoiser von NVIDIA verglichen wurde, konnte gezeigt werden, dass bereits mit einfachen Architekturen und begrenztem Datensatz beachtliche Ergebnisse erzielt werden können.

Ein Rückblick auf die zentrale Forschungsfrage –

*Wie gut eignet sich ein auf neuronalen Netzen basierender Autoencoder zur Rauschunterdrückung in Path-Tracing-Renderings mit geringer Sample-Anzahl? –* zeigt, dass diese im Rahmen der Arbeit weitgehend beantwortet werden konnte. Der entwickelte Ansatz erzielte sowohl visuell als auch quantitativ überzeugende Resultate. Ein Vergleich mit etablierten Verfahren steht zwar noch aus, doch die erzielten Resultate deuten auf ein hohes Potenzial hin.

Insgesamt verdeutlicht diese Arbeit das Potenzial von Deep-Learning-basierten Methoden zur Rauschreduktion im Bereich Path Tracing. Der entwickelte Ansatz bietet eine solide Grundlage für weiterführende Forschung und lässt sich in zukünftigen Arbeiten hinsichtlich Daten, Modellarchitektur und Trainingsstrategie weiter ausbauen.

### 5.1 Verbesserungspotenzial

- **Größerer und vielfältigerer Datensatz:** Die Trainingsdaten basieren auf 15 Blender-Szenen. Eine breitere Auswahl an Geometrien, Lichtverhältnissen und Materialien könnte die Generalisierungsfähigkeit des Modells verbessern.
- **Dynamische Lernratensteuerung:** Die Lernrate wurde zwar über Optuna bestimmt, blieb jedoch während des gesamten Trainings konstant. Der Einsatz adaptiver Lernraten, z. B. durch einen Scheduler, könnte zu einer effizienteren Konvergenz führen.
- **Erweiterte Verlustfunktionen:** Aktuell kommt eine Kombination aus L1 Loss und SSIM zum Einsatz. Ergänzungen durch wahrnehmungsorientierte Metriken (Perceptual Loss) oder Verfahren mit konkurrierenden Netzwerken könnten die Bildqualität weiter steigern.

### 5.2 Ausblick

Die in dieser Arbeit entwickelten Grundlagen und Ergebnisse bieten vielfältige Ansätze für weiterführende Arbeiten im Bereich des Deep-Learning-basierten Denoisings für Path Tracing. Insbesondere eröffnen sich folgende Möglichkeiten zur Weiterentwicklung:

- **Vergleich mit industriellen Verfahren:** Ein zentraler nächster Schritt wäre der systematische Vergleich des entwickelten Modells mit etablierten Denoisern wie dem OptiX-Denoiser von NVIDIA oder einem anderen integrierten Denoiser von Blender. Dies würde eine genauere Einschätzung der praktischen Leistungsfähigkeit ermöglichen.
- **Erweiterung des Datensatzes:** Der verwendete Datensatz war durch Zeitliche Knappheit kompakt gehalten. Zukünftige Arbeiten könnten eine grösse Vielfalt an Szenen, Beleuchtungssituationen und Kameraeinstellungen integrieren, um die Generalisierbarkeit des Modells zu verbessern.

- **Modelloptimierung:** Obwohl das gewählte U-Net-ähnliche Modell bereits überzeugende Ergebnisse lieferte, könnten komplexere Architekturen wie Attention-Mechanismen, Residual-Verbindungen oder rekurrente Elemente untersucht werden, um das Rauschen noch robuster zu entfernen.
- **Temporale Konsistenz:** Für Anwendungen in Animationen ist es entscheidend, dass der Denoiser nicht nur pro Frame, sondern auch über Zeit hinweg konsistente Ergebnisse liefert. Ein Ausblick auf zeitliche Denoising-Modelle wäre hier vielversprechend.
- **Echtzeitfähigkeit und Integration:** Die Integration des Modells in bestehende Rendering-Pipelines – etwa als Plugin in Blender oder in Echtzeit-Engines – könnte die Praxistauglichkeit weiter erhöhen. Optimierungen im Hinblick auf Laufzeit und Speicherverbrauch wären dafür notwendig.

## 6 Abbildungsverzeichnis

Abbildung 1: Vergleich zwischen verrauschem und entrauschem Bild aus der Demo-Szene «Barbershop Interior» von der Blender Foundation [19], Lizenz: CC-BY	5
Abbildung 2: Path Tracing Illustration. Quelle: [20]	8
Abbildung 3: Verrausches Bücherregal mit Blender (Cycles) mit 5 Samples per Pixel. Quelle der Blender Szene: [4], Lizenzfrei	9
Abbildung 4: Projektablauf Gantt Diagramm	14
Abbildung 5: Beispieldszene aus Blender, Quelle: [4], Lizenzfrei	18
Abbildung 6: Visualisierung der verschiedenen Rauschstufen eines Patches	19
Abbildung 7: Vollständige Architektur des Autoencoder-Modells	22
Abbildung 8: Schlechtes visuelles Beispiel mit der alten Modellarchitektur	25
Abbildung 9: Gutes Ergebnis der alten Modellarchitektur im Vergleich	25
Abbildung 10: Verlauf des Training- und Validierungsverlusts	27
Abbildung 11: Beispiel der grafischen Anwendung mit einem 1024 x 1024 Pixel, verrauschem Bild mit 200 Samples per Pixel. Demo-Szene von Blender [19], erstellt von Glenn Melenhorst, verwendet unter Annahme einer CC-BY-Lizenz	29
Abbildung 12: Die grafische Applikation mit dem entrauschten Bild. Demo-Szene von Blender [19], erstellt von Glenn Melenhorst, verwendet unter Annahme einer CC-BY-Lizenz	30
Abbildung 13: Entrausches Bild mit Patch-Artefakten (behoben)	32
Abbildung 14: Visueller Unterschied zwischen Verrauscht (200 Samples), Entrauscht und Referenz	33
Abbildung 15: Qualitätsmetriken auf dem Testdatensatz	35
Abbildung 16: Statistischer Ausreisser. Von links nach rechts: Verrauscht, Entrauscht, Referenz	36
Abbildung 17: Cornell Box, Kamera 1, Patch 1, 10 Samples per Pixel	44
Abbildung 18: Cornell Box, Kamera 0, Patch 0, 50 Samples per Pixel	44
Abbildung 19: Cornell Box, Kamera 1, Patch 10, 10 Samples per Pixel	44
Abbildung 20: Mr Elephant, Kamera 1, Patch 10, 100 Samples per Pixel	45
Abbildung 21: Mr Elephant, Kamera 2, Patch 0, 10 Samples per Pixel	45
Abbildung 22: Mr Elephant, Kamera 2, Patch 7, 25 Samples per Pixel	45
Abbildung 23: Mr Elephant, Kamera 3, Patch 4, 200 Samples per Pixel	46
Abbildung 24: Mr Elephant, Kamera 6, Patch 10, 50 Samples per Pixel	46
Abbildung 25: RestRoom Interior, Kamera 0, Patch 6, 200 Samples per Pixel	46
Abbildung 26: RestRoom Interior, Kamera 0, Patch 3, 50 Samples per Pixel	47
Abbildung 27: RestRoom Interior, Kamera 0, Patch 5, 100 Samples per Pixel	47
Abbildung 28: RestRoom Interior, Kamera 1, Patch 9, 500 Samples per Pixel	47
Abbildung 29: Anwendung des Denoisers am SLProject Path Tracer. 10 Samples per Pixel, 0.5-fache Auflösung. Links ist verrauscht, rechts ist entrauscht	48

## 7 Tabellenverzeichnis

Tabelle 1: Tatsächlich investierte Zeit	15
Tabelle 2: Risikomanagement Tabelle	16
Tabelle 3: Verlustmetriken auf der alten Modellarchitektur	25
Tabelle 4: Verlustfunktion Verrauscht vs. Entrauscht	34
Tabelle 5: Verlustmetriken von einem statistischen Ausreisser	36
Tabelle 6: Zielerreichungstabelle	36

## 8 Glossar

### **Adam-Optimizer**

Ein verbreiteter Optimierungsalgorithmus für das Training neuronaler Netze, der adaptive Lernraten für jede Gewichtung berechnet.

### **Bildrauschen**

Störende visuelle Artefakte in Bildern, die besonders bei nicht konvergierten Path-Tracing-Renderings auftreten. Es erscheint oft als Körnung oder Flecken.

### **BRDF**

Die Bidirectional Reflectance Distribution Function (BRDF) beschreibt, wie Licht an einer Oberfläche in verschiedene Richtungen reflektiert wird. Sie definiert den Anteil des einfallenden Lichts, der in eine bestimmte Ausfallsrichtung gestreut wird, abhängig von Einfalls- und Ausfallswinkel. Die BRDF ist eine zentrale Komponente bei der realistischen Modellierung von Materialeigenschaften in der Computergrafik.

### **CNN**

Ein Convolutional Neural Network (Faltungsneuronales Netz) ist ein spezieller Typ künstliches neuronales Netz, das besonders für die Verarbeitung von Bildern geeignet ist. Es nutzt Faltungsschichten, um Merkmale wie Kanten, Formen oder Texturen automatisch zu erkennen. CNNs werden häufig in der Bildklassifikation, Objekterkennung und Bildverarbeitung eingesetzt.

### **Denoising**

Englisch für Entrauschung oder Rauschunterdrückung. In dieser Arbeit ist besonders der Prozess gemeint, Rauschen aus Bildern zu entfernen.

### **Downsampling**

Die Reduktion der räumlichen Auflösung eines Bildes oder einer Feature Map. In neuronalen Netzen geschieht dies häufig durch Operationen wie Max-Pooling oder Strided Convolution. Ziel ist es, die Datenmenge zu verringern, semantisch wichtige Merkmale zu verdichten und die Rechenlast zu reduzieren.

### **Entrauschung**

Allgemeiner Begriff für Verfahren, die störendes Bildrauschen aus digitalen Bildern entfernen. In dieser Arbeit wird hauptsächlich Deep Learning zur Entrauschung von Path-Tracing-Renderings eingesetzt.

### **Feature Map**

Eine Feature Map ist das Ergebnis einer Faltungsschicht innerhalb eines neuronalen Netzes. Sie stellt eine zweidimensionale Repräsentation dar, in der bestimmte Merkmale (z. B. Kanten, Texturen oder Formen) erkannt wurden. Mehrere Feature Maps entstehen durch den Einsatz mehrerer Filter und bilden gemeinsam die Grundlage für die hierarchische Merkmalsextraktion.

### **Ground Truth**

Der Begriff Ground Truth bezeichnet in der Bildverarbeitung und im maschinellen Lernen die als korrekt angenommene Referenz oder den „Wahrheitswert“, an dem die Ergebnisse eines Modells oder einer Methode gemessen und bewertet werden. Im Kontext des Path Tracing bezeichnet die Ground Truth ein Bild, das mit einer sehr hohen Anzahl an Samples gerendert wurde und daher praktisch rauschfrei ist.

### **GUI**

Ein Graphical User Interface ermöglicht die Interaktion mit einem Computerprogramm über visuelle Elemente wie Schaltflächen, Menüs oder Fenster. Im Gegensatz zu textbasierten Schnittstellen erleichtert eine GUI die Bedienung durch intuitive, oft mit der Maus steuerbare Bedienelemente.

### **OpenCV**

Eine Open-Source-Bibliothek für Bildverarbeitung und Computer Vision, die häufig in Python oder C++ verwendet wird.

### **Overfitting**

Englisch für Überanpassung. Problem bei Neuronalen Netzen, die lediglich den Trainings-Datensatz auswendig lernen.

## **Renderer**

Ein Renderer ist ein Softwaremodul oder Programm, das den Renderprozess ausführt. Es nimmt eine 3D-Szene als Eingabe und erzeugt daraus ein Bild oder eine Animation. Renderer können auf unterschiedliche Methoden basieren, z. B. Rasterisierung (für Echtzeit-Grafik) oder Raytracing (für realistische Beleuchtung).

---

## **Rendern**

Rendern bezeichnet den Prozess der Bildsynthese aus einer 3D-Szene durch einen Computer. Dabei werden Geometrie, Materialien, Lichtquellen und Kameraeinstellungen verarbeitet, um ein zweidimensionales Bild zu erzeugen.

---

## **Renderpipeline**

Die Renderpipeline beschreibt die Abfolge von Verarbeitungsschritten, die nötig sind, um eine Szene in ein Bild zu überführen. Sie umfasst typischerweise die Szenenvorbereitung, Sichtbarkeitsberechnung, Beleuchtung, Schattierung und Nachbearbeitung (z. B. Entrauschung). In modernen Systemen kann die Renderpipeline hardware- oder softwareseitig implementiert sein und lässt sich je nach Anwendungsfall anpassen

---

## **SLProject4**

SLProject ist eine plattformunabhängige 3D-Computergrafik-Szenengraph-Bibliothek von der Berner Fachhochschule.

---

## **SmallPT**

Ein minimalistischer, nur 100 Zeilen langer Path Tracer, der von Kevin Beason entwickelt wurde. Er dient als Lernprojekt, um die grundlegenden Konzepte des physikalisch basierten Renderings (PBR) mittels Path Tracing zu demonstrieren. Trotz seiner Einfachheit erzeugt SmallPT realistisch beleuchtete Bilder durch Simulation globaler Lichttransportprozesse.

---

## **Upsampling**

Die Erhöhung der räumlichen Auflösung einer Feature Map. In Decoderstrukturen wird dies typischerweise durch transponierte Faltungen (ConvTranspose2d) oder Interpolationsverfahren erreicht, um das ursprüngliche Bildformat schrittweise wiederherzustellen.

---

## **U-Net**

Eine Architektur für neuronale Netze mit symmetrischem Encoder-Decoder-Aufbau und Skip-Connections, ursprünglich für medizinische Bildsegmentierung entwickelt, aber effektiv für Denoising.

---

## 9 Literaturverzeichnis

- [1] M. Pharr, W. Jakob und G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, Morgan Kaufmann, 2016.
- [2] H. Chen, Y. Zhang, W. Zhang, P. Liao, K. Li, J. Zhou und G. Wang, „Low-dose CT via convolutional neural network,“ *Biomedical Optics Express*, Bd. 8, Nr. 2, pp. 679-694, 2017.
- [3] J. T. Kajiya, „The rendering equation,“ *ACM SIGGRAPH Computer Graphics*, Bd. 20, Nr. 4, pp. 143-150, 1986.
- [4] MySimsWorld, „Library Hall Blender Scene free 3D model,“ cgtrader, [Online]. Available: <https://www.cgtrader.com/free-3d-models/interior/hall/library-hall-blender-scene>. [Zugriff am 08.06.2025].
- [5] NVIDIA, «NVIDIA OptiX Ray Tracing Engine,» [Online]. Available: <https://developer.nvidia.com/rtx/ray-tracing/optix>. [Zugriff am 05.06.2025].
- [6] Intel, «Intel Open Image Denoise,» [Online]. Available: <https://www.openimagedenoise.org/>. [Zugriff am 05.06.2025].
- [7] AMD, «AMD Radeon ProRender,» [Online]. Available: <https://www.amd.com/en/products/graphics/software/radeon-prorender.html>. [Zugriff am 05.06.2025].
- [8] C. R. Chaitanya, A. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai und T. Aila, «Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder,» in *ACM SIGGRAPH*, New York, 2017.
- [9] S. Bako, T. Vogels, B. McWilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, F. Rousselle und M. Zwicker, «Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings,» in *ACM Transactions on Graphics (TOG), SIGGRAPH*, New York, 2017.
- [10] K. Zhang, W. Zuo, Y. Chen, D. Meng und L. Zhang, „Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising,“ *IEEE Transactions on Image Processing*, Bd. 26, Nr. 7, pp. 3142-3155, 2017.
- [11] A. Buades, B. Coll und J. M. Morel, „A Non-Local Algorithm for Image Denoising,“ in *IEEE Computer Society Conference on Computer Vision and Pattern*, San Diego, 2005.
- [12] C. a. H. J. a. C. W. a. S. T. a. F. D. J. a. N. M. Saharia, „Image Super-Resolution via Iterative Refinement,“ *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, Bd. 45, Nr. 4, pp. 4713-4726, 2023.
- [13] P. L. Cornu, «GitHub,» 2025. [Online]. Available: <https://github.com/asyxui/PTDenoising>. [Zugriff am 06.06.2025].
- [14] PyTorch, „torch.nn module,“ [Online]. Available: <https://docs.pytorch.org/docs/stable/nn.html>. [Zugriff am 06.06.2025].
- [15] X. S. C. & Y. Y.-B. Mao, „Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections,“ in *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems*, New York, 2016.
- [16] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*, MIT Press, 2016.
- [17] S. Ioffe und C. Szegedy, „Batch normalization: Accelerating deep network training by reducing internal covariate shift,“ arXiv, 2015.
- [18] H. Zhao, O. Gallo, I. Frosio und J. Kautz, „Loss Functions for Image Restoration with Neural Networks,“ *IEEE Transactions on Computational Imaging*, Bd. 3, Nr. 1, pp. 47-57, 2017.
- [19] Blender Foundation, „Demo Files,“ [Online]. Available: <https://www.blender.org/download/demo-files/>. [Zugriff am 08.06.2025].
- [20] Universität Marburg, „Vulkan Ray Tracing Pipeline,“ 2024. [Online]. Available: [https://www.mathematik.uni-marburg.de/~thormae/lectures/graphics2/graphics\\_2\\_1\\_ger\\_web.html#1](https://www.mathematik.uni-marburg.de/~thormae/lectures/graphics2/graphics_2_1_ger_web.html#1). [Zugriff am 08.06.2025].

## 10 Anhang

### 10.1 Weitere Ergebnisse

Für alle Bilder gilt: Von links nach rechts: Verrauscht, Entrauscht, Referenz.



Abbildung 17: Cornell Box, Kamera 1, Patch 1, 10 Samples per Pixel

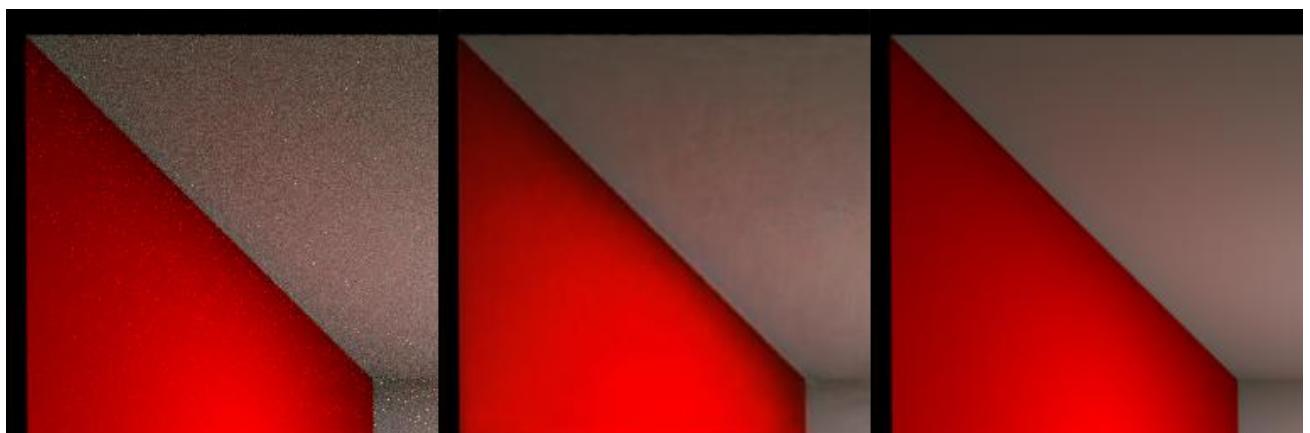


Abbildung 18: Cornell Box, Kamera 0, Patch 0, 50 Samples per Pixel

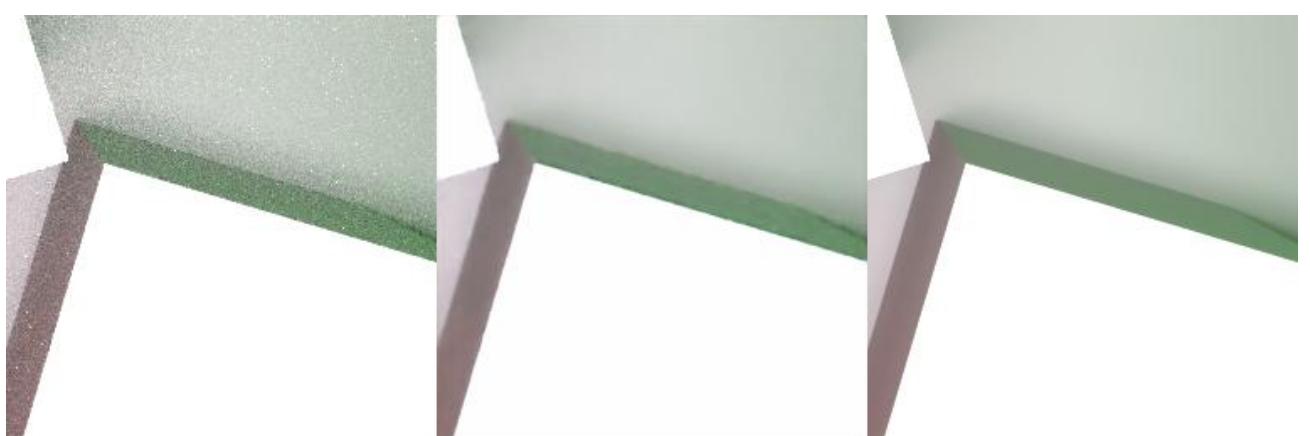


Abbildung 19: Cornell Box, Kamera 1, Patch 10, 10 Samples per Pixel



Abbildung 20: Mr Elephant, Kamera 1, Patch 10, 100 Samples per Pixel



Abbildung 21: Mr Elephant, Kamera 2, Patch 0, 10 Samples per Pixel



Abbildung 22: Mr Elephant, Kamera 2, Patch 7, 25 Samples per Pixel



Abbildung 23: Mr Elephant, Kamera 3, Patch 4, 200 Samples per Pixel

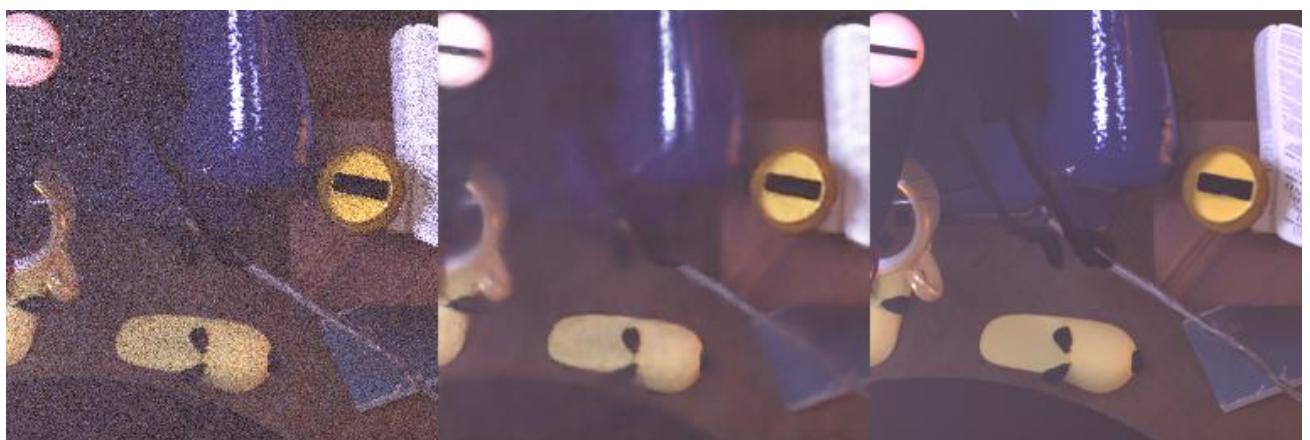


Abbildung 24: Mr Elephant, Kamera 6, Patch 10, 50 Samples per Pixel



Abbildung 25: RestRoom Interior, Kamera 0, Patch 6, 200 Samples per Pixel



Abbildung 26: RestRoom Interior, Kamera 0, Patch 3, 50 Samples per Pixel

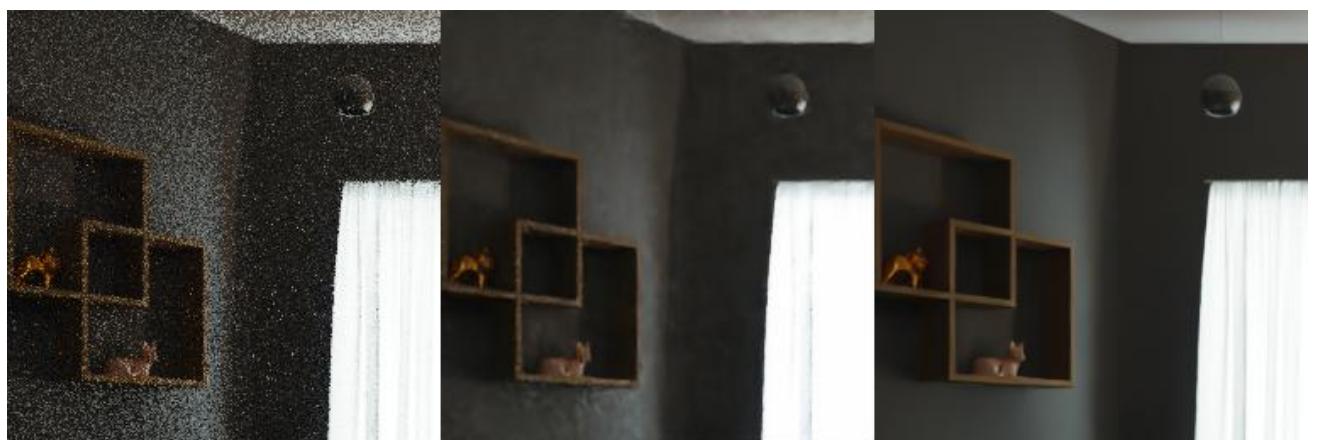


Abbildung 27: RestRoom Interior, Kamera 0, Patch 5, 100 Samples per Pixel



Abbildung 28: RestRoom Interior, Kamera 1, Patch 9, 500 Samples per Pixel

## 10.2 SLProject4 Resultat

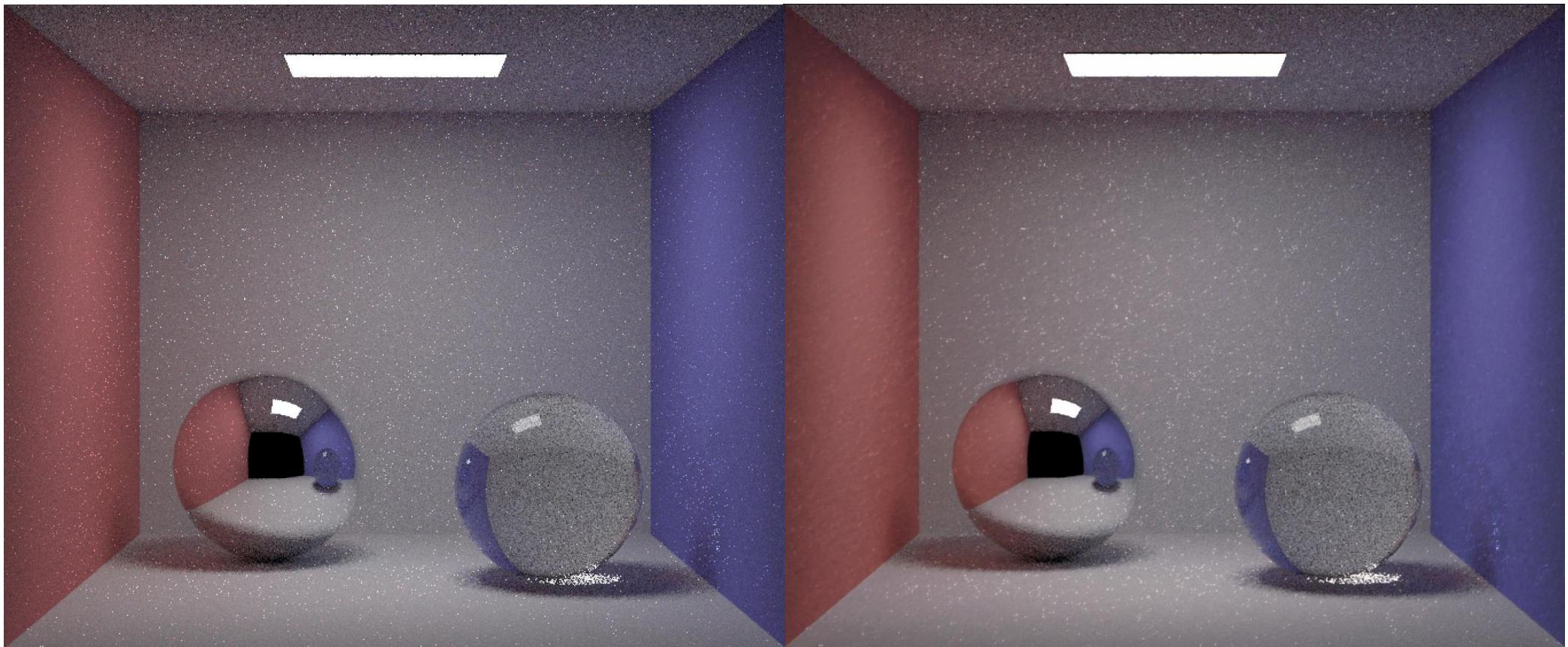


Abbildung 29: Anwendung des Denoisers am SLProject Path Tracer. 10 Samples per Pixel, 0.5-fache Auflösung. Links ist verrauscht, rechts ist entrauscht

Der Denoiser scheint die «Fireflies» – also die weissen Punkte, die der SLProject Path Tracer fälschlicherweise erzeugt – nicht als Rauschen zu erkennen und entfernt sie daher nicht (nur an den Seitenwänden teilweise). Wahrscheinlich liegt das daran, dass der Datensatz, mit dem das Modell trainiert wurde, keine solchen Fireflies enthält.

## 10.3 Meeting Protokoll

### 10.3.1 Kick-off Meeting (24.02.2025 09:00)

**Standort:** Microsoft Teams Online Meeting

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Pascal Cornu (Studierender)

**Meeting Protokoll:**

- Prof. Marcus Hudritsch gibt Rahmenbedingungen, so wie den Zugewiesenen Experten (Dr. Harald Studer) durch.
- Zukünftige Meetings für den Abgleich werden alle zwei Wochen für das gesamte Projekt festgesetzt.
- Pascal Cornu zeigte seine bisherige Recherche in das Thema Path Tracing.
- Die nächsten Schritte, insbesondere die Planung und Recherche wurden diskutiert: Das Projekt «SmallIPT» soll selbst ausgeführt werden und mit dem Path Tracer von SLProject4 verglichen werden. Dadurch sollen die Artefakte vom SLProject4 Path Tracer untersucht werden. Zusätzlich soll ein einfacher Autoencoder selbst implementiert werden.

### 10.3.2 Abgleich 1 (13.03.2025 11:00)

**Standort:** Microsoft Teams Online Meeting

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Pascal Cornu (Studierender)

**Meeting Protokoll:**

- Pascal Cornu zeigte seine weitere Recherche in die Themen Path Tracing und Denoising, sowie die das erlangte Wissen über die Unterschiede vom SmallIPT und dem SLProject4 Path Tracer. Die so genannten Fireflies, die Artefakte die im SLProject, nicht aber im SmallIPT vorhanden sind, wurden erklärt und ein bisschen eingedämmt.
- Der simple Autoencoder, der auf dem MNIST Datensatz basiert, wurde vorgestellt.
- Prof. Marcus Hudritsch gibt Feedback über die Projektplanung und teilt seine Kenntnisse über Denoising.
- Die nächsten Schritte werden geplant: Für den Denoising Autoencoder braucht es ein Datensatz. Öffentliche Datensätze sollen gesucht und analysiert werden. Prof. Marcus Hudritsch weist auf die Seite «Papers With Code» hin.

### 10.3.3 Abgleich 2 (24.03.2025 09:00)

**Standort:** Microsoft Teams Online Meeting

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Pascal Cornu (Studierender)

**Meeting Protokoll:**

- Pascal Cornu präsentiert zwei gefundene Datensätze, die für das Modell verwendet werden können.
- Prof. Hudritsch findet die Qualität der Datensätze nicht ausreichend. Ein eigener Datensatz soll synthetisch erstellt werden. Beispielsweise mit Blender in kostenlosen öffentlichen Szenen ein Bild generieren lassen nach 5 Sekunden rendern, 30 Sekunden, 1 Minute und so weiter.
- Für den Expertentreff soll eine PowerPoint Präsentation vorbereitet werden.

#### **10.3.4 Expertentreff (07.04.2025 09:00)**

**Standort:** Biel, Rolex Gebäude, Zimmer N553

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Dr. Harald Studer (Experte)
- Pascal Cornu (Studierender)
- Nick Denzler, Kilian Wampfler, Tim Schär (Zuschauer)

**Meeting Protokoll:**

- Pascal Cornu stellt in einer 10 Minuten PowerPoint Präsentation folgende Themen vor:
  - Projektziele und Projektablauf
  - Erkenntnisse der Recherche über Path Tracing und Denoising
  - Arbeit am SLProject Denoiser
  - Generierung des Datensatzes mit Blender
- Dr. Harald Studer stellt mehrere Fragen über das Projekt, insbesondere über vergleichbare Arbeiten und die Einschränkung der Projektziele.
- Prof. Marcus Hudritsch stellt mehrere Fragen über den bisher erstellten Datensatz.

#### **10.3.5 Abgleich 3 (22.04.2025 09:00)**

**Standort:** Microsoft Teams Online Meeting

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Pascal Cornu (Studierender)

**Meeting Protokoll:**

- Pascal Cornu präsentiert seine Erweiterung des Datensatzes mit folgenden Funktionen:
  - In den Blender Szenen können manuell weitere Kameras platziert werden, die mit dem Skript automatisch eingelesen werden, um die Grösse des Datensatzes zu erweitern.
  - Unterteilung der 1024 x 1024 Pixel Bilder in 16 nicht überlappende 256 x 256 Pixel Bilder. (Die verlangte Grösse des Modells)
- Pascal Cornu präsentiert einen simplen Autoencoder, der den erstellten Datensatz verwendet und beliebige 256 x 256 Pixel Bilder entrauschen kann. Die Qualität des Outputs ist jedoch noch nicht gut.
- Prof. Marcus Hudritsch gibt Feedback über den erstellten Datensatz. Er ist skeptisch über die Menge der Bilder, 10000 Bilder reichen womöglich nicht aus. Zusätzlich erläutert er, dass bisher jedes Projekt mit einem eigenen Datensatz gescheitert ist.
- Die nächsten Schritte werden geplant: Die Architektur des Modells soll verbessert werden. Generell soll mehr Arbeit in das Modell gesteckt werden.

#### **10.3.6 Abgleich 4 (05.05.2025 09:00)**

**Standort:** Microsoft Teams Online Meeting

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Pascal Cornu (Studierender)

**Meeting Protokoll:**

- Pascal Cornu präsentiert seine Fortschritte beim Modell:
  - Tiefere Modell Architektur mit Skip-Connections» und «double convolutions».
  - Optuna Hyperparameter Training wurde implementiert, aber noch nicht ausgeführt.
- Prof. Marcus Hudritsch gibt Feedback über den fehlenden Schutz gegen Overfitting.

- Die nächsten Schritte werden geplant: Zugriff bei der Management Platform for Deep Learning (MLMP) soll angefragt werden. Das Training des Modells soll auf dieser Platform stattfinden. Zuvor soll aber noch der Schutz gegen Overfitting erweitert werden.

#### **10.3.7 Abgleich 5 (19.05.2025 09:00)**

**Standort:** Microsoft Teams Online Meeting

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Pascal Cornu (Studierender)

**Meeting Protokoll:**

- Pascal Cornu präsentiert das trainierte Modell und die Verbesserungen am Modell gegen Overfitting. Zusätzlich wurde der Datensatz erweitert und aufgeteilt in Trainings-, Test- und Validationsdatensatz.
- Prof. Marcus Hudritsch ist grundsätzlich zufrieden mit dem Modell. Er weist darauf hin, die Bachelorthesis eine Woche früher abzugeben, damit er Zeit hat alles anzuschauen vor der Verteidigung. Weiter soll das Modell entweder ins SLProject eingebaut werden, oder mit Python und OpenCV eine grafische Anwendung erstellt werden.

#### **10.3.8 Abgleich 6 (02.06.2025 09:00)**

**Standort:** Microsoft Teams Online Meeting

**Teilnehmer:**

- Prof. Marcus Hudritsch (Betreuer)
- Pascal Cornu (Studierender)

**Meeting Protokoll:**

- Pascal Cornu präsentiert die grafische Anwendung, die es erlaubt ein beliebig grosses Bild zu entauschen. Zusätzlich wurde das Poster und das Book erstellt.
- Prof. Marcus Hudritsch gibt Feedback über das Book. Es muss noch erwähnt werden warum Path Tracing so wichtig ist. Zudem ist der Umfang eher knapp.
- Prof. Marcus Hudritsch weist auf einige Dinge hin, die in der Dokumentation nicht fehlen dürfen. Weiter soll ein Vergleich mit dem Optix Denoiser in Blender gemacht werden.

## 11 Selbständigkeitserklärung

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelor-Thesis selbstständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet.

Ich bestätige weiterhin, dass ich bei der Erstellung dieser Studienarbeit durchgehend steuernd gearbeitet habe und von einer KI erzeugte Inhalte nicht unreflektiert übernommen habe.

Pascal Cornu

Name, Vorname

12.06.2025

Datum

Unterschrift

A handwritten signature in black ink, appearing to read "P. Cornu".