|  |
| --- |
| Rendering Denoising  Path Tracing Denoising mit neuronalen Netzen  **Bachelorthesis**  [Kurztext einfüngen, falls gewünscht] |
| Studiengang: CPCVR  Autor: Pascal Cornu  Betreuer: Marcus Hudritsch  Experten: Harald Studer  Datum: 02.06.2025 |

Management Summary

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit.

Inhaltsverzeichnis

[Path Tracing Denoising mit neuronalen Netzen 1](#_Toc199777840)

[Management Summary 2](#_Toc199777841)

[Inhaltsverzeichnis 3](#_Toc199777842)

[1 Einleitung 4](#_Toc199777843)

[1.1 Path Tracing 5](#_Toc199777844)

[1.1.1 Rendering-Gleichung 5](#_Toc199777845)

[1.1.2 Monte-Carlo-Integration beim Path Tracing 5](#_Toc199777846)

[1.1.3 Path Tracing Algorithmus 6](#_Toc199777847)

[1.1.4 Path Tracing Rauschen 7](#_Toc199777848)

[1.2 Denoising 8](#_Toc199777849)

[1.2.1 Klassische Filtermethoden 8](#_Toc199777850)

[1.2.2 Statistische Modelle 8](#_Toc199777851)

[1.2.3 Lernbasierte Methoden 8](#_Toc199777852)

[2 Projektmanagement 9](#_Toc199777853)

[2.1 Projektziele 9](#_Toc199777854)

[2.2 Meilensteine 9](#_Toc199777855)

[2.2.1 Planung 9](#_Toc199777856)

[2.2.2 Forschung 9](#_Toc199777857)

[2.2.3 Datensatz 9](#_Toc199777858)

[2.2.4 Modell 9](#_Toc199777859)

[2.2.5 Präsentation 9](#_Toc199777860)

[2.3 Projektablauf 10](#_Toc199777861)

[3 Implementation 11](#_Toc199777862)

[3.1 Erstellung des Datensatzes 11](#_Toc199777863)

[3.1.1 Szenenvorbereitung in Blender 11](#_Toc199777864)

[3.1.2 Automatisierte Bildgenerierung per Python-Skript 11](#_Toc199777865)

[3.1.3 Bildausschnitte 12](#_Toc199777866)

[3.1.4 Organisation und Verwendung des Datensatzes 13](#_Toc199777867)

[3.2 Erstellung des Modells 13](#_Toc199777868)

[3.2.1 Modell Architektur 13](#_Toc199777869)

[3.2.2 Optimierung der Hyperparameter 13](#_Toc199777870)

[3.2.3 Schutz vor Overfitting 14](#_Toc199777871)

[3.3 Modell Training 14](#_Toc199777872)

[3.4 Grafische Anwendung 15](#_Toc199777873)

[4 Abbildungsverzeichnis 17](#_Toc199777874)

[5 Tabellenverzeichnis 17](#_Toc199777875)

[6 Glossar 17](#_Toc199777876)

[7 Literaturverzeichnis 17](#_Toc199777877)

[8 Anhang 18](#_Toc199777878)

[9 Selbständigkeitserklärung 19](#_Toc199777879)

# 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.

## 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.

### Rendering-Gleichung

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

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

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.

### 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

wird gegeben durch:

wobei:

* ist die Anzahl der Proben.
* sind zufällig ausgewählte Punkte in der Domäne .
* ist die Wahrscheinlichkeitsdichtefunktion (PDF), für die Stichprobe .

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

wobei die BRDF als Wichtungsfunktion verwendet wird, um die Varianz zu verringern und die Konvergenz zu verbessern.

### Path Tracing Algorithmus

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

1. Strahlenerzeugung: Für jedes Pixel wird ein Primärstrahl von der Kamera in die Szene geworfen.
2. Strahlenschnittpunkt: Der erste Schnittpunkt mit einer Oberfläche wird gefunden.
3. Schattierung und BRDF-Sampling: Die Oberflächen-BRDF bestimmt, wie das Licht reflektiert oder durchgelassen wird.
4. Rekursive Pfadverfolgung: Eine neue Richtung wird auf der Grundlage der BRDF abgetastet, und der Prozess wird fortgesetzt, bis eine Abbruchbedingung erfüllt ist. Zum Beispiel das Erreichen einer Lichtquelle oder maximaler Tiefe.
5. Monte-Carlo-Schätzung: Die akkumulierte Strahldichte entlang des Pfades wird gemittelt, um die Pixelfarbe zu schätzen.

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 unvoreingenommen, 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, die darauf abzielen, qualitativ hochwertige Bilder aus verrauschten Renderings mit geringer Abtastung zu rekonstruieren.

### 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 1: Verrauschtes Bücherregal mit Blender (Cycles)

## 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 pro 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.

### 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.
* **Gaussian-Blur**: Wendet eine gewichtete Glättung an, wobei näherliegende Pixel stärker berücksichtigt werden. Geeignet zur Rauschreduktion mit geringem Detailverlust.

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

### 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.

### 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.

# Projektmanagement

## Projektziele

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabellenkopf** |  |  |  |
| Path Tracing und Denoising erforschen | Erarbeitung eines tiefen Verständnisses von Path Tracing.  Untersuchung von Rauschquellen, bestehenden Rauschunterdrückungstechniken und neuronalen Netzen zur Rauschunterdrückung. | | |
| Implementierung eines einfachen Path Tracers | Mit dem erlernten Wissen aus der Forschung soll entweder ein eigener Path Tracer erstellt werden, oder den Path Tracer aus dem SLProject4 oder SmallPT übernehmen und auf die Bedürfnisse anpassen. | | |
| Implementierung eines Denoisers | Für den Path Tracer soll ein Denoiser für die Rauschunterdrückung erstellt werden. Dazu gehört die Auswahl oder Erstellung eines Datensatzes für das Trainieren vom Datensatz. | | |
| Denoiser Vergleich | Den Denoiser mit Nvidias Optix Denoiser oder den verschiedenen Methoden in Blender vergleichen. | | |

Tabelle 1: Projektziele

## Meilensteine

### Planung

* Meetings mit dem Betreuer sind festgelegt.
* Projektziele, so wie Meilensteine sind erstellt.
* Projektablauf ist festgelegt.

### Forschung

* Wissen über Path Tracing und Denoising ist erlangt und dokumentiert.
* Der Path Tracer im SLProject4 wurde analysiert und die Fehler so weit möglich behoben.

### Datensatz

* Öffentliche Datensätze für Denoising wurden erkundet.
* Ein passendes Datensatz wurde ausgewählt, falls möglich.
* Existiert kein passender öffentlicher Datensatz, wurde ein eigener Datensatz erstellt.

### Modell

* Ein Modell wurde erstellt. Es übt anhand des erstellten Datensatz. Das Modell kann neue Bilder akzeptieren und gibt als Resultat eine entrauschte Version desselben Bildes zurück.
* Falls mehrere Arten von Modellen sich als sinnvoll erweisen, wurden zwei ausgewählt und ein Modell pro Ansatz erstellt und trainiert.

### Präsentation

* Dokumentation, Video, Plakat und Präsentation wurden erstellt und finalisiert.
* Die Präsentation ist geübt und die Ausstellung ist vorbereitet.

## Projektablauf

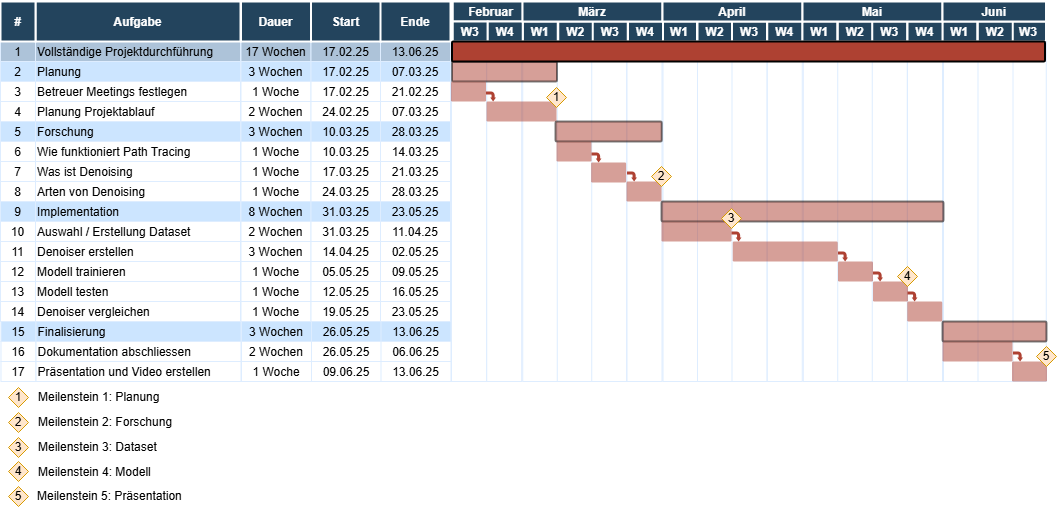


Abbildung 2: Projektablauf Gantt Diagramm

# Implementation

## 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.

### 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. Für jede Kamera wurde anschliessend eine Reihe von Bildern aus derselben Szene, aber mit unterschiedlichen Parametern erzeugt.

### 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 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 Kamera-Perspektive wird zusätzlich ein Ground-Truth-Bild generiert. Dieses verwendet eine hohe Sampleanzahl in Kombination mit dem OptiX-Denoiser von Blender, um ein visuell möglichst rauschfreies Referenzbild zu erhalten.

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

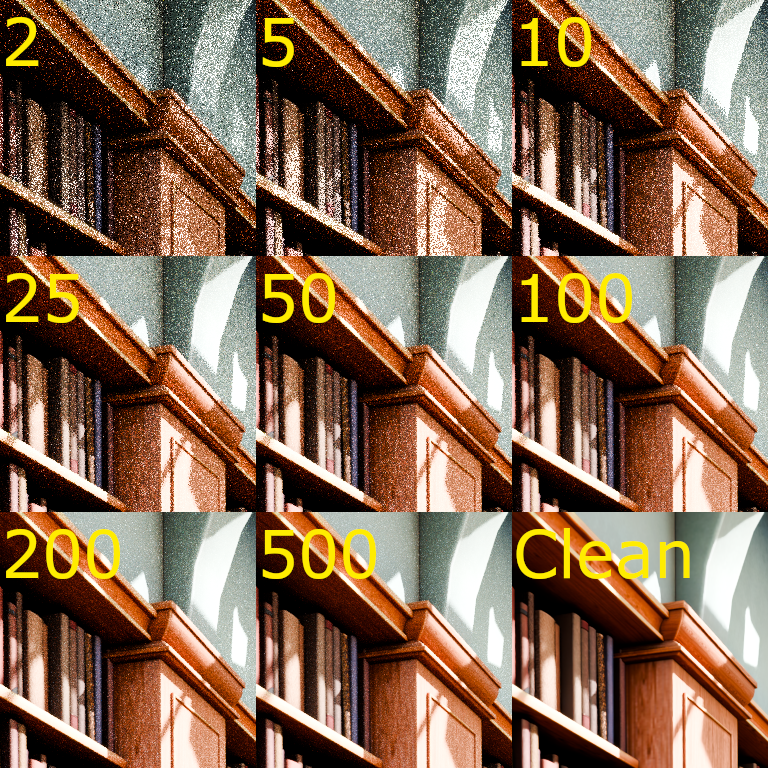


Abbildung 3: Visualisierung der verschiedenen Rauschstufen eines Patches

Das

### Bildausschnitte

Da neuronale Netze, insbesondere Autoencoder, typischerweise auf kleinere Eingabedimensionen trainiert werden, wurden die 1024 × 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 × 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.

### Organisation und Verwendung des Datensatzes

Der vollständige Datensatz besteht aus 13056 Patch-Paaren (verrauschtes Eingabebild + Ground Truth). Die Daten wurden in drei Teilmengen aufgeteilt:

* Training – 9856 Patch-Paare
* Validierung – 1792 Patch-Paare
* Test – 1408 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.

## 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.

### 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.

### Optimierung der Hyperparameter

Um das Modell möglichst effektiv zu trainieren, wurde die Optimierungsbibliothek Optuna eingesetzt. Sie hilft dabei, wichtige Einstellungswerte des Trainings automatisch zu testen und zu verbessern. Dazu gehören unter anderem die sogenannte 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 Einstellungen für den verwendeten Optimierungsalgorithmus.

Bei der automatisierten Optimierung mit Optuna wurden gezielt verschiedene sogenannte 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

}

### 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.
* **Frühzeitiges Abbrechen**: Wenn sich die Qualität auf den Validierungsdaten über längere Zeit nicht verbessert hat, wurde das Training automatisch gestoppt.
* **Regulierung der Gewichte**: Das Netzwerk wurde leicht dazu "bestraft", wenn es sehr extreme Werte für seine internen Verbindungen verwendet hat. So bleibt es stabiler und verallgemeinert besser.

## 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.

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

## 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.

Die 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 genau zu betrachten.

Um die visuelle Qualitätsverbesserung 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. 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.

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

## Test und Validierung

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.

Dabei werden zwei wichtige Metriken berechnet:

* 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 Entrauschung.
* SSIM (Structural Similarity Index Measure): SSIM quantifiziert die wahrgenommene strukturelle Ähnlichkeit zwischen zwei Bildern, wobei Werte näher bei 1 eine hohe Bildähnlichkeit anzeigen. Diese Metrik bewertet das visuelle Erscheinungsbild der Bilder besser als reine Pixelabstände.

Das Skript berechnet beide Metriken sowohl vor als auch nach der Entrauschung für jedes Bild im Testdatensatz. Zudem werden die Ergebnisse tabellarisch in einer CSV-Datei gespeichert, um eine spätere Analyse zu erleichtern.

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Parallel dazu erzeugt das Skript einen Plot, der die L1- und SSIM-Werte für alle Testbilder visualisiert. Dieser Plot zeigt anschaulich die Verbesserung durch das Modell: Die roten und orangen Kurven repräsentieren die Werte vor dem Denoising (verrauschtes Bild), während die grünen und blauen Kurven die Werte nach dem Denoising darstellen. Typischerweise ist zu beobachten, dass der L1-Loss nach der Entrauschung deutlich sinkt und der SSIM-Wert steigt, was auf eine erfolgreich verbesserte Bildqualität hinweist.

Zusätzlich werden für jedes Testbild Vergleichsbilder gespeichert, die das verrauschte, das entrauschte sowie das saubere Bild nebeneinander darstellen. Diese Bilder ermöglichen eine direkte visuelle Kontrolle der Entrauschungsqualität.

# Resultate

Schlussfolgerungen/Fazit

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit.

# Abbildungsverzeichnis

[Abbildung 1: Verrauschtes Bücherregal mit Blender (Cycles) 7](#_Toc199774399)

[Abbildung 2: Projektablauf Gantt Diagramm 10](#_Toc199774400)

[Abbildung 3: Visualisierung der verschiedenen Rauschstufen eines Patches 12](#_Toc199774401)

# Tabellenverzeichnis

[Tabelle 1: Projektziele 5](#_Toc194865739)

# Glossar

**Overfitting**

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

**bla**

Et ut aut isti repuditis qui ium

**Cowoll**

Et ut aut isti repuditis qui ium

# Literaturverzeichnis

**Literatureintrag**

*Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr* 7

**Literatureintrag**

*Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr* 9

**Literatureintrag**

*Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr* 11

# Anhang

Et ut aut isti repuditis qui ium nonsecturia quis incientiae laborem elliquis et quatur, sitiur aut od moluptatur aut ea conseque peri sim erro essequisit remporia dem et landi dest, cone poris quunt volecab ipidero quatur ad quibusamus.

# Selbständigkeitserklärung

Ich bestätige, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe. Sämtliche Textstellen, die nicht von mir stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum:

Unterschrift: