

**Fachhochschule Südwestfalen - Hagen**

Fachbereich Elektrotechnik und Informationstechnik

## Masterarbeit

**Entwicklung eines Lehrsimulators für die  
Computertomographie durch Implementierung  
von physikalischen Modellen, Gerätekomponenten  
und Rekonstruktionsprozessen in C++**

**Jan Wolzenburg**



# **Masterarbeit**

## **Entwicklung eines Lehrsimulators für die Computertomographie durch Implementierung von physikalischen Modellen, Geräteteilkomponenten und Rekonstruktionsprozessen in C++**

**Development of an educational simulator for  
computed tomography by implementing physical  
models, device components and reconstruction  
processes in C++**

**Verfasser:** Jan Wolzenburg

**Wohnort:** Altena

**Matrikelnummer:** 10048688

**Studiengang:** Medizintechnik

**Erstprüfer:** Prof. Dr. Jens Gröbner

**Zweitprüfer:** Prof. Dr. Amir Moussavi-Biugui

**Abgabedatum:** 16. Mai 2024



# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>VII</b>
<b>Symbolverzeichnis</b>	<b>IX</b>
<b>Abstract</b>	<b>1</b>
<b>1 Einleitung</b>	<b>3</b>
1.1 Technische Entwicklung der Computertomographie . . . . .	4
1.2 Ziel und Aufbau der Arbeit . . . . .	6
<b>2 Grundlagen</b>	<b>7</b>
2.1 Erzeugung und Schwächung von Röntgenstrahlung . . . . .	7
2.2 Mathematische Grundlagen der Computertomographie . . . . .	16
2.3 Bestimmung der Schwächungskoeffizienten . . . . .	28
2.4 Modellannahmen in der Computertomographie . . . . .	30
2.5 Artefakte in der Computertomographie . . . . .	31
<b>3 Implementierung</b>	<b>39</b>
3.1 Konzeptionierung . . . . .	42
3.2 Darstellung geometrischer Objekte . . . . .	45
3.3 Abbilden der Gerätekomponenten . . . . .	50
3.4 Beschreibung von Körpermodellen . . . . .	57
3.5 Algorithmus zur Strahlverfolgung . . . . .	61
3.6 Aufnahme der Projektionen . . . . .	72
3.7 Rekonstruktion der Schnittbilder . . . . .	77
3.8 Grafische Benutzeroberfläche . . . . .	80
<b>4 Verifikation</b>	<b>85</b>
4.1 Gerätekomponenten . . . . .	85
4.2 Strahlverfolgung . . . . .	88
4.3 Physikalische Modelle . . . . .	88
4.4 Rückprojektion . . . . .	94
4.5 Gesamte Simulation . . . . .	96
<b>5 Diskussion und Fazit</b>	<b>103</b>
5.1 Bewertung der Implementierung . . . . .	103
5.2 Bewertung des Nutzens . . . . .	107
5.3 Ausblick . . . . .	107
<b>Literaturverzeichnis</b>	<b>A</b>
<b>A Anhang</b>	<b>E</b>
A.1 Kompilieren und Benutzung des Simulators . . . . .	E
A.2 Ausgewählter Quelltext . . . . .	J
<b>Eigenständigkeitserklärung</b>	<b>V</b>



# Abbildungsverzeichnis

1.1	Darstellung eines Körpers durch Aufnahme vieler Schnittbilder mit der CT . . . . .	3
1.2	Aufbau von Computertomographen erster und zweiter Generation . . . . .	4
1.3	Aufbau von Computertomographen dritter und vierter Generation . . . . .	5
2.1	Schematischer Aufbau einer Röntgenröhre . . . . .	7
2.2	Kontinuierliches Bremsspektrum einer Röntgenröhre . . . . .	9
2.3	Winkelverteilung der gestreuten Photonen bei Compton-Streuung . . . . .	14
2.4	Schwächung von Röntgenstrahlung in Wasser . . . . .	15
2.5	Ordnungsschema für Linienintegrale in der kartesischen Ebene . . . . .	17
2.6	Die Radon-Transformation . . . . .	17
2.7	Das Fourier-Scheiben Theorem . . . . .	19
2.8	Rückprojektion der gefilterten Projektion zur Bildrekonstruktion . . . . .	21
2.9	Rotation eines Pixels des Detektorarrays um den Ursprung . . . . .	23
2.10	Abtastung eines kontinuierlichen Signals mit einem Dirac-Kamm . . . . .	23
2.11	Überlappung der Fourier-transformierte bei Abtastung im Ortsraum . . . . .	24
2.12	Filterfunktionen nach Ram. und Lak. sowie Shepp und Logan . . . . .	25
2.13	Faltungskerne der Filter nach Ram. und Lak. sowie Shepp und Logan . . . . .	26
2.14	Interpolation bei der Rückprojektion . . . . .	27
2.15	Entstehung von Teilvolumenartefakten innerhalb eines Schnittbildes . . . . .	32
2.16	Erwartete und reale Projektionswerte bei Teilvolumen . . . . .	32
2.17	Entstehung von Streustrahlungartefakten . . . . .	33
2.18	Bleilamellen als Streustrahlenraster . . . . .	33
2.19	Ideale und energieabhängige Röntgenschwächung im Vergleich . . . . .	34
2.20	Cupping-Artefakte durch Strahllaufhärtung . . . . .	35
2.21	Energieabhängigkeit der Schwächung . . . . .	36
2.22	Metallartefakte durch Zahnimplantate im rekonstruierten Bild . . . . .	37
3.1	Funktionale Komponenten des Simulators . . . . .	44
3.2	Klassendiagramm für die analytische Geometrie . . . . .	46
3.3	Verschachtelte Koordinatensystem dargestellt als Baum . . . . .	47
3.4	Gerade und Ebene im kartesischen Koordinatensystem . . . . .	49
3.5	Klassen, die Geraden und Ebenen im dreidimensionalen Raum darstellen . . . . .	49
3.6	Röntgenröhre und Detektor als Komponenten der Gantry . . . . .	51
3.7	Nadelstrahlen und Röntgenröhre als Klassen in der Implementierung . . . . .	51
3.8	Erzeugung des diskreten Spektrums . . . . .	53
3.9	Vier Strahlen, die für einen Detektorpixel erzeugt werden . . . . .	53
3.10	Zum Detektor zugehörige Klassen . . . . .	54
3.11	Punkteraster im Sinogramm . . . . .	54

3.12 Konstruktion zweier Pixel des Pixelarrays . . . . .	55
3.13 Minimale Anzahl der Teilrotationen zur vollständigen Rasterabtastung . . . . .	56
3.14 Klassen zur Abbildung des Körpermodells . . . . .	57
3.15 Volumenelemente im Körpermodell . . . . .	58
3.16 Näherung der energieabhängigen Absorption verschiedener Gewebe . . . . .	60
3.17 Programmablaufplan der Strahlverfolgung durch das Modell . . . . .	62
3.18 Schnitt eines Strahls mit einem Voxel . . . . .	65
3.19 Programmablaufplan zur Bestimmung des Voxelausganges . . . . .	66
3.20 Programmablaufplan zum Streualgorithmus . . . . .	68
3.21 Programmablaufplan zum Sammeln der Projektionsdaten . . . . .	73
3.22 Gleichzeitige Strahlverfolgung mit durch Multithreading . . . . .	74
3.23 PAP zur Bestimmung der Radon-Koordinaten aus einer Geraden . . . . .	76
3.24 Hauptfenster der grafischen Benutzeroberfläche . . . . .	81
3.25 Verarbeitungsfenster der grafischen Benutzeroberfläche . . . . .	82
3.26 Fenster zur Erstellung einfacher Körpermodelle . . . . .	83
4.1 Aus gegebenen Parametern konstruierte Pixelanordnung . . . . .	85
4.2 Schrittweises Füllen des Sinogramms mit Projektionsdaten . . . . .	86
4.3 Aus Röhreneigenschaften berechnetes Spektrum der Röntgenröhre . . . . .	87
4.4 Erzeugung der Strahlen aus den Pixelpositionen . . . . .	88
4.5 Schnittpunkte eines Strahls mit den Voxeln im Körpermodell . . . . .	88
4.6 Schnitt durch ein einfacher Körpermodell . . . . .	89
4.7 Veränderung der mittleren Strahlungsenergie bei Transmission . . . . .	90
4.8 Streckenabhängigkeit der Koeffizienten für den idealen und reellen Fall . . . . .	91
4.9 Initiale und gestreute Strahlen im Wassermodell . . . . .	92
4.10 Erprobung der Erzeugung zufälliger Ereignisse . . . . .	92
4.11 Relative Häufigkeit diskreter Streuwinkel bei Streureignissen . . . . .	93
4.12 Schwächungskoeffizienten, die aus der Simulation resultieren . . . . .	94
4.13 Schnitt durch ein Körpermodell zur Erprobung der Algorithmen . . . . .	94
4.14 Aufgenommene Projektionsdaten eines Schnittbildes . . . . .	95
4.15 Gefilterte Projektionsdaten eines Schnittbildes . . . . .	95
4.16 Vergleich von ungefilterter und gefilterter Rückprojektion . . . . .	96
4.17 Vergleich von idealem und rekonstruiertem Schnittbild . . . . .	97
4.18 Auswirkung des Röntgenvorfilters auf das rekonstruiert Bild . . . . .	98
4.19 Streifenartefakte durch begrenzte Detektorsensitivität . . . . .	99
4.20 Simulation von Streuung und Effekt des Streustrahlenrasters . . . . .	100
4.21 Vergleich von geringer und hoher Simulationsqualität . . . . .	100
4.22 Aufnahme mit hoher Simulationsqualität und -auflösung . . . . .	101
4.23 Simulation mit einem Körpermodell mit vielen Elementen . . . . .	101

# Symbolverzeichnis

$\lambda$	Wellenlänge
$\nu$	Frequenz
$E$	Energie
$h$	Plancksches Wirkungsquantum
$c$	Lichtgeschwindigkeit
$U_A$	Beschleunigungsspannung in einer Röntgenröhre
$e$	Elementarladung
$E_{\text{kin}}$	Kinetische Energie beschleunigter Elektronen
$\nu_{\text{max}}$	Maximale Frequenz der Bremsstrahlung
$J_\nu$	Spektrale Leistungsdichte
$J$	Strahlungsintensität
$\eta$	Wirkungsgrad der Bremsstrahlung
$P_A$	Leistung der Elektronenstrahlung
$I_A$	Anodenstrom
$Z$	Ordnungszahl
$k$	Wirkungsgradkonstante
$n$	Photonenfluss
$\mu$	Schwächungskoeffizient
$\mu_m$	Massenschwächungskoeffizient
$\rho$	Dichte
$d$	Flächendichte
$\sigma$	Wirkungsquerschnitt
$N_A$	Avogadrokonstante
$M$	Molare Masse
$\mu_{\text{str}}$	Massenstreuukoeffizient
$\mu_{\text{tot}}$	Totaler Massenschwächungskoeffizient
$\mu_{\text{abs}}$	Massenschwächungskoeffizient durch Absorption
$E_{nQ}$	Bindungsenergie
$\sigma_{\text{pho}}$	Wirkungsquerschnitt des Photoeffektes
$n_Q$	Quantenzahl
$\alpha$	Feinstrukturkonstante
$r_0$	Klassischer Elektronenradius
$\omega_0$	Relativistische Grenzfrequenz
$M_{\text{At}}$	Molare Masse von Atomen
$m_H$	Masse eines Wasserstoffatoms
$\mu_{\text{pho}}$	Massenschwächungskoeffizient durch den Photoeffekt
$\sigma_K$	Wirkungsquerschnitt für einen Compton-Streuprozess
$\Omega$	Raumwinkel
$a$	Auf die Ruheenergie des Elektrons bezogene Photonenenergie
$m_0$	Ruhemasse des Elektrons
$\mu_C$	Massenschwächungskoeffizient durch Compton-Streuung
$M_{\text{El}}$	Molare Masse von Elektronen

$N_{\text{El}}$	Elektronenanzahl je Masse
$p_{\text{WW}}$	Wahrscheinlichkeit für eine Wechselwirkung
$\vartheta$	Streuwinkel
$w$	Massenanteil
$x$	Komponente in kartesischen Koordinatensystemen
$y$	Komponente in kartesischen Koordinatensystemen
$z$	Komponente in kartesischen Koordinatensystemen
$p$	Projektionswerte oder Linienintegral
$\ell$	Eine Gerade
$g$	Lot zu einer Geraden $\ell$ durch den Ursprung
$\theta$	Winkel zwischen Geradenlot und $x$ -Achse
$s$	Kürzester Abstand einer Gerade zum Ursprung
$t$	Parameter einer Gerade $\ell$
$P$	1D Fourier-Transformierte der Projektionen
$\omega$	Kreisfrequenz
$\mathcal{F}$	Operator für die 1D Fourier-Transformation
$u$	Kartesische Komponente für 2D Fourier-Transformierte
$v$	Kartesische Komponente für 2D Fourier-Transformierte
$\tilde{p}$	Gefilterte Projektionen
$H$	Filterfunktion im Frequenzbereich
$h$	Impulsantwort oder Filterfunktion im Raumbereich
$N_\theta$	Anzahl der Projektionswinkel
$N_s$	Anzahl der Projektionen je Winkel
$s_{\text{Mess}}$	Messfeldgröße
$J_0$	Initial Strahlungsintensität
$\text{HU}$	Hounsfield-Einheiten
$\varphi$	Akzeptanzwinkel des Streustrahlrasters
$\mathbb{Q}$	Menge rationaler Zahlen
$\vec{e}_i$	Basisvektor eines kartesischen Koordinatensystems
$\vec{o}$	Ursprung eines kartesischen Koordinatensystems
$\mathbb{R}$	Menge reeller Zahlen
$\vec{u}_\ell$	Ursprung einer Gerade
$\vec{u}_E$	Auflagepunkt einer Ebenen
$\vec{v}_\ell$	Richtung einer Gerade
$\vec{v}_1$	Richtungvektor einer Ebenen
$\vec{v}_2$	Richtungvektor einer Ebenen
$\vec{n}$	Normalenvektor einer Ebene

# Abstract

Die Computertomographie zählt zu den wichtigsten bildgebenden Verfahren in der Medizin. Sie ermöglicht die Aufnahme von Schnittbildern für diagnostische oder therapeutische Zwecke. Das Ziel dieser Arbeit ist die Beschreibung und Bewertung eines in C++ implementierten Simulators für die Computertomographie. Der Simulator hat den Zweck, die Computertomographie auf einer physikalischen und technischen Ebene abzubilden. Der Einsatz des Simulators ist dabei auf die Veranschaulichung besonderer physikalischer und technischer Effekte zu Lehrzwecken begrenzt.

Durch den Simulator werden die relevanten physikalischen Modelle, technischen Gerätekomponenten und Rekonstruktionsprozesse abgebildet. Zur Interaktion mit dem Simulator wurde eine grafische Benutzeroberfläche zur Eingabe von Simulationsparametern und Darstellung der Ergebnisse erstellt. Durch Implementierung der physikalischen Modelle und Gerätekomponenten ist der Simulator implizit in der Lage, Besonderheiten der Computertomographie, die sich durch Artefakte in rekonstruierten Bildern äußern, darzustellen. Unter anderem werden der Einfluss von Streustrahlung, Strahlaufl härtung und Metall durch den Simulator gezeigt. Weiterhin zeigt der Simulator den Einfluss verschiedener Geräteparameter auf die rekonstruierten Bilder.

In der Arbeit werden neben den physikalischen, mathematischen und technischen Grundlagen, die Besonderheiten der Computertomographie beschrieben. Dazu zählen insbesondere die Diskrepanz zwischen den Modellannahmen, die für die Computertomographie getroffen werden, und der Realität sowie den daraus resultierenden Folgen. Der Fokus liegt auf der Erläuterung implementierungsspezifischer Details und die Bewertung der entwickelten Lösungen anhand von Simulationsergebnissen.



# 1. Einleitung

Bei der Computertomographie (CT) handelt es sich um ein bildgebendes Verfahren in der Medizin. Die Computertomographie ermöglicht die Aufnahme überlagerungsfreier Schnittbilder eines menschlichen Körpers [1, S. 60]. Daher stammt der Name *Tomographie*, der sich aus den griechischen Worten *tomo* (Schnitt) und *graphein* (Schreiben) zusammensetzt. Im Gegensatz zum Projektionsröntgen, bei dem aus genau einer Richtung die Projektion von Röntgengenstrahlung gemessen wird, werden zur Aufnahme von Schnittbildern viele Projektionen aufgenommen. Durch einen Prozess, der als *Rückprojektion* bezeichnet wird, können aus den Projektionsdaten Schnittbilder rekonstruiert werden. Die gemessenen Projektionen entsprechen der Schwächung von Röntgenstrahlung. Daher ist die Gewebeeigenschaft, die in den Schnittbildern dargestellt wird, der Röntgenschwächungskoeffizient oder die zu Wasser und Luft relative Schwächung. Letztere wird in sogenannten *Hounsfield-Einheiten* (HU) angegeben. Durch die Aufnahme mehrerer übereinanderliegender Schnittbilder können dreidimensionale Abbildungen einer Körperregion oder des ganzen Körpers erstellt werden. So ist es möglich, Darstellungen wie in Abbildung 1.1 innerhalb von 25 Sekunden zu erzeugen. Besonders in der medizinischen Diagnostik hat die Computertomographie eine hohe Relevanz.



Quelle: [1, S. 61]

**Abbildung 1.1.:** Dreidimensionale Darstellung eines Körpers durch Aufnahme vieler Schnittbilder mit der Computertomographie

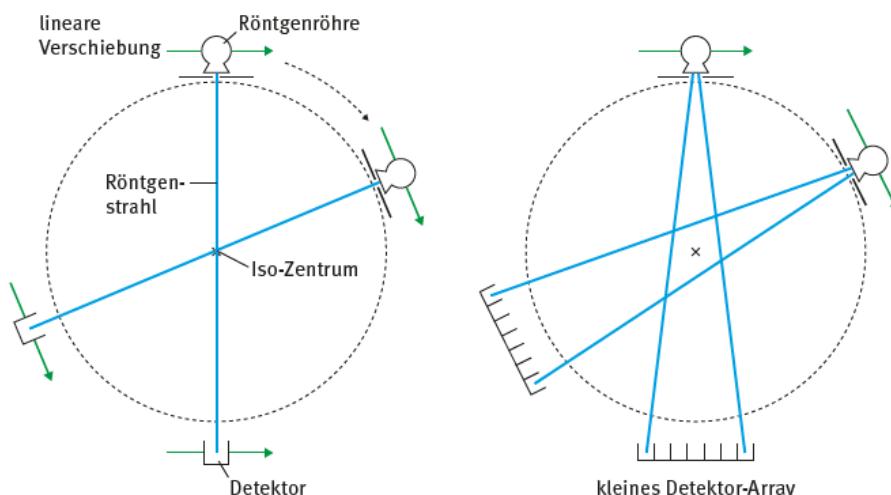
Konkrete Einsatzgebiete sind die Darstellung von Gefäßen oder die Bildgebung während der Erstversorgung von Traumapatienten [1, S. 60]. Weiterhin zählen Durchblutungsanalysen, gezielte Untersuchungen von Organen und Hilfsanwendung in der Bestrahlungsplanung sowie während Operationen zu den Einsatzgebieten [2, S. 484]. Die medizinische Relevanz der

Computertomographie wird deutlich, wenn die Anzahl der in Deutschland durchgeführten Untersuchungen betrachtet wird. Im Jahr 2018 zählten von insgesamt 130 Millionen durchgeführten Röntgenuntersuchungen 10 % zu CT-Untersuchungen [3, S. 43]. Das sind 0,16 CT-Untersuchungen pro Person und Jahr, was für eine Person im Mittel einer Häufigkeit von ungefähr einer Untersuchung pro sechs Jahren entspricht.

Abseits der Medizin spielt die Computertomographie in technischen, forensischen, anthropologischen, archäologischen oder paläontologischen Anwendungen eine Rolle. Besonders die zerstörungsfreie Materialprüfung und dreidimensionale Visualisierung stehen in diesen Anwendungen im Vordergrund.[1, S. 60]

## 1.1. Technische Entwicklung der Computertomographie

Computertomographen wurden seit den ersten Aufnahmen aus dem Jahr 1970 stetig weiterentwickelt. Die Geräte lassen sich in vier Generationen einteilen, die sich erheblich im Aufbau unterscheiden. In Abbildung 1.2 (Links) ist der Aufbau des ersten Computertomographen aus dem Jahr 1970 schematisch abgebildet. Eine Strahlungsquelle und ein Detektor sind gegenüberliegend angeordnet und werden verschoben sowie rotiert. Die Strahlungsquelle war ein radioaktives Isotop und die Aufnahme eines Schnittbildes dauerte 9 Tage [4, S. 137]. Ein

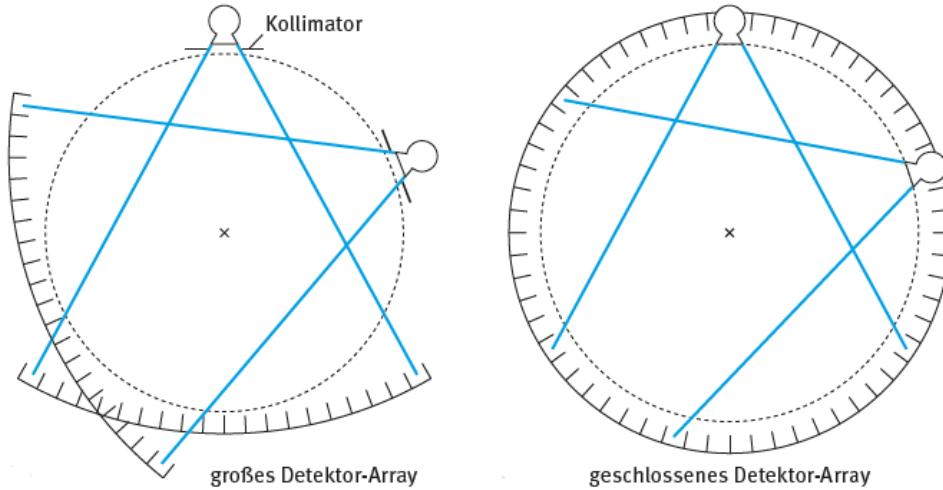


Quelle: [1, S. 64]

**Abbildung 1.2:** Prinzipielle Aufbau von Computertomographen erster und zweiter Generation. Links: Erste Generation mit Rotation und Translation eines Nadelstrahles. Rechts: Zweite Generation mit Rotation und Translation eines kleinen Fächerstrahles

Computertomograph der zweiten Generation, dessen Aufbau in Abbildung 1.2 (Rechts) dargestellt ist, verfügt über ein lineares Array aus Detektoren. Statt eines Strahlungsbündels, wird ein Strahlungsfächer von einer Röntgenröhre emittiert. Dadurch können für eine Position des Detektorarrays eine größere Anzahl von Projektionen gemessen werden. Die Aufnahmezeit liegt dadurch bei ungefähr 20 Sekunden [4, S. 138]. Wie bei Computertomographen der ersten Generation werden die Projektionen durch Kombination von Translation und Rotationen aufgenommen.

Computertomographen der dritten Generation (Abb. 1.3 Links) verfügen über ein ausreichend großes Detektorarray und großen Öffnungswinkel des Strahlungsfächers, sodass keine Translation notwendig ist. Die Röhre und das Detektorarray rotieren um den Patienten. Computertomographen dieser Generation sind jene, die am häufigsten zum Einsatz kommen [1, S. 66]. Bei Computertomographen der vierten Generation (Abb. 1.3 Rechts) sind die Detektoren stationär um den Patienten angeordnet. Ausschließlich die Röntgenröhre rotiert.



Quelle: [1, S. 64]

**Abbildung 1.3.:** Prinzipieller Aufbau von Computertomographen dritter und vierter Generation. Links: Dritte Generation mit Rotation eines großen Fächerstrahles. Rechts: Vierte Generation mit ortsfesten Detektoren und rotierender Röntgenröhre

Weiterentwicklungen der primitiven Generationen, wie sie im vorherigen Paragrafen beschrieben wurden, sind beispielsweise die Elektronenstrahl-CT, Spiral-CT und Mehrzeilen-CT. Die Elektronenstrahl-CT ermöglicht die Aufnahme eines Schnittbildes innerhalb von 50 ms [2, S. 90]. Das ist möglich, da sich keine Komponente für eine Aufnahme bewegt. Stattdessen wird ein Elektronenstrahl so gelenkt, dass an einer beliebigen Stelle auf dem Kreisumfang Röntgenstrahlung emittiert wird [2, S. 90]. Bei der Spiral-CT rotiert die *Gantry*, was die Röntgenquelle und die Detektoren als mechanische Einheit zusammenfasst, kontinuierlich. Simultan wird der Patient entlang der Longitudinalachse verschoben. Dadurch kann in kurzer Zeit eine 3D-Aufnahme einer Körperregion angefertigt werden [4, S. 170]. Mehrzeilen-CT ermöglichen die Aufnahme mehrerer Schnittbilder gleichzeitig. Dafür sind mehrere Detektorzeilen hintereinander angeordnet. Durch Nutzung von 128 und mehr Zeilen und Verkürzung der Rotationszeit sind Applikationen, die eine kurze Akquisitionszeit benötigen, möglich [1, S. 68–69].

Ein moderner Computertomograph aus dem Jahr 2016 ist zum Beispiel der Mehrzeilen-CT *Siemens SOMATOM Defintions AS*. Dieser kann über bis zu 128 Zeilen und einer Rotationszeit von 0,3 s verfügen. Die kleinste Ortsauflösung in jede Raumrichtung liegt bei 0,33 mm. Die zeitliche Auflösung liegt bei bis zu 150 ms. [5]

## 1.2. Ziel und Aufbau der Arbeit

Ziel der Arbeit ist die Entwicklung eines Simulators für die Computertomographie. Es existieren kommerzielle Simulatoren [6] [7], die den Zweck haben, klinisches Personal in einer virtuellen Umgebung auszubilden. Die Simulatoren dienen dem Training der Abläufe und dem Erwerb klinischer Erfahrung in einer rein virtuellen Umgebung. Neben Simulatoren zum Training klinischen Personals existiert ein Simulator [8], der die grundlegende technische Funktionsweise darstellt und als didaktisches Instrument dient. Dieser Simulator bildet nur das mathematische Grundprinzip dar und berücksichtigt keine physikalischen Modelle oder technische Aspekte.

Der Simulator, der in dieser Arbeit entwickelt wird, soll anhand physikalischer Modelle Röntgenstrahlung und deren Interaktion mit beliebigen Körpermodellen nachbilden. Außerdem sollen mit Computertomographen der dritten Generation als Basis Gerätekomponenten und ihre Eigenschaften im Simulator abgebildet sein. Zusätzlich sollen die notwendigen Prozesse zur Rekonstruktion von Schnittbildern Teil der Simulation sein. Der Simulator soll als didaktisches Instrument, das die Auswirkungen gewisser physikalischer und technischer Aspekte realitätsnah darstellt, dienen können. Von der Implementierung physikalischer Modelle und technischer Komponenten wird sich versprochen, dass implizit eine Vielzahl der Besonderheiten oder Artefakte Teil der Simulationsergebnisse sind. Dazu zählen zum Beispiel die Auswirkungen von Streustrahlung, Metallobjekten oder Strahlaufhärtung. Der Simulator soll dabei über eine grafische Benutzeroberfläche bedienbar sein und die Simulationsergebnisse anzeigen.

Diese Arbeit erläutert die physikalischen Grundlagen und Modelle zur Beschreibung von Röntgenstrahlung und deren Interaktion mit Materie. Auf Basis dieser Modelle und der notwendigen mathematischen Konzepte wird die grundlegende Idee der Computertomographie erklärt. Dabei wird der Unterschied zwischen der kontinuierlichen Beschreibung und der diskreten Beschreibung, welche durch technische Limitationen notwendig ist, erläutert. Es werden die Besonderheiten der physikalischen Modelle sowie deren Grenzen und die daraus resultierenden Artefakte dargestellt. Mit hohem Detailgrad wird die Implementierung des Simulators beschrieben. Konkret sind es die Beschreibung von Strahlung, deren Erzeugung sowie Detektion und Verfolgung durch Körpermodelle, die erklärt werden. Weiterhin werden die implementierten Rekonstruktionsprozesse und erstellte grafische Benutzeroberfläche gezeigt. Im weiteren Verlauf der Arbeit werden die relevanten Komponenten des Simulators durch Vergleichen mit den physikalischen Modellen oder erwarteten Ergebnissen verifiziert. Auf Grundlage der Verifikation findet eine Bewertung der Implementierung statt. Diese liefert eine Basis zur Beurteilung des Nutzens und Formulierung eines Ausblicks.

## 2. Grundlagen

In der Computertomographie wird Röntgenstrahlung genutzt, um eine spezifische Materieeigenschaften in Volumenelementen eines Objektes zu bestimmten. Bei dieser Materieeigenschaft handelt es sich um das Schwächungsverhalten von Röntgenstrahlung. Die Computertomographie nutzt physikalische Modelle und mathematische Verfahren, um durch Messung von Röntgenprojektionen Schnittbilder zu rekonstruieren.

### 2.1. Erzeugung und Schwächung von Röntgenstrahlung

Bei Röntgenstrahlung handelt es sich um elektromagnetische Strahlung mit Wellenlängen zwischen 10 nm und 0,05 nm. Im elektromagnetischen Spektrum befindet sich Röntgenstrahlung zwischen dem ultravioletten Licht und der Gammastrahlung.

Elektromagnetische Strahlung wird durch ihre Wellenlänge  $\lambda$ , Frequenz  $v$  oder Quantenenergie  $E$  der Photonen charakterisiert. Diese drei Größen stehen über das Plancksche Wirkungsquantum  $h$  und die Lichtgeschwindigkeit  $c$  in Beziehung. [4, S. 6]

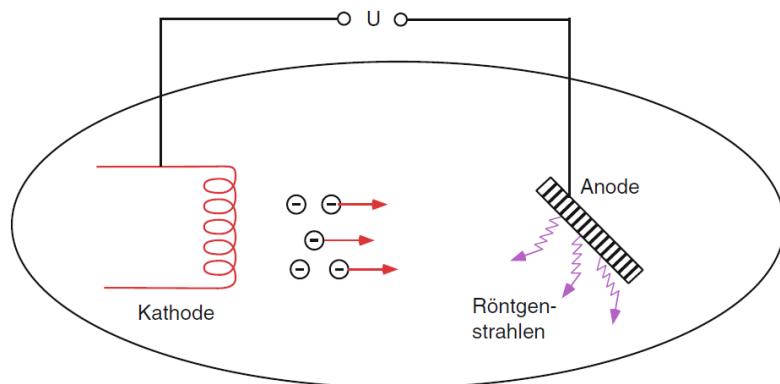
$$v = \frac{c}{\lambda} \quad (2.1)$$

$$E = h \cdot v \quad (2.2)$$

mit:  $h = 6,626\,196 \cdot 10^{-34} \text{ J} \cdot \text{s}$

$$c = 299\,792\,456 \frac{\text{m}}{\text{s}}$$

Zur technischen Erzeugung von Röntgenstrahlung werden Röntgenröhren eingesetzt. Eine Röntgenröhre ist ein evakuiertes Gefäß, in dem sich eine Glühkathode und eine Anode befinden. Wie in Abbildung 2.1 zu sehen, sorgt ein elektrischer Strom durch die Glühkathode über den thermoelektrischen Effekt für einen Austritt von Elektronen. Die Elektronen werden im elektrostatischen Feld, das zwischen Anode und Kathode anliegt, in Richtung Anode mit der Beschleunigungsspannung  $U_A$  beschleunigt.



Quelle: [4, S. 6]

**Abbildung 2.1.:** Schematischer Aufbau einer Röntgenröhre

Nach Eintritt der Elektronen in das Anodenmaterial werden diese durch Wechselwirkung mit den Atomen des Anodenmaterials abgebremst. Die Abbremsung im elektrischen Feld der

Atomkerne sorgt für eine Emission elektromagnetischer Strahlung, die als *Bremsstrahlung* bezeichnet wird [2, S. 19]. Die maximale Frequenz  $v_{\max}$  im Spektrum der Bremsstrahlung korrespondiert über Gleichung 2.2 mit der kinetischen Energie  $E_{\text{kin}}$ , die die Elektronen im elektrostatischen Feld aufnehmen [4, S. 5]. Strahlung mit dieser Frequenz wird emittiert, wenn ein Elektron ohne vorherigen Energieverlust mit einem Atomkern des Anodenmaterials kollidiert und vollständig absorbiert wird [2, S. 22].

$$E_{\text{kin}} = e \cdot U_A \quad (2.3)$$

$$\Rightarrow v_{\max} = \frac{e \cdot U_A}{h} \quad (2.4)$$

mit:  $e = 1,602\,191\,7 \text{ A} \cdot \text{s}$

Treffen die beschleunigten Elektronen auf eine infinitesimal dünne Schicht des Anodenmaterials, kann angenommen werden, dass für jedes Frequenzintervall  $dv$  von der Frequenz null bis  $v_{\max}$  die spektrale Leistungsdichte  $J_v$  konstant ist. Die spektrale Leistungsdichte ist eine Größe, die beschreibt welche Intensität  $dJ$  in einem Frequenzintervall  $dv$  abgestrahlt wird. [9, S. 59]

$$J_v = \frac{dJ}{dv} \quad (2.5)$$

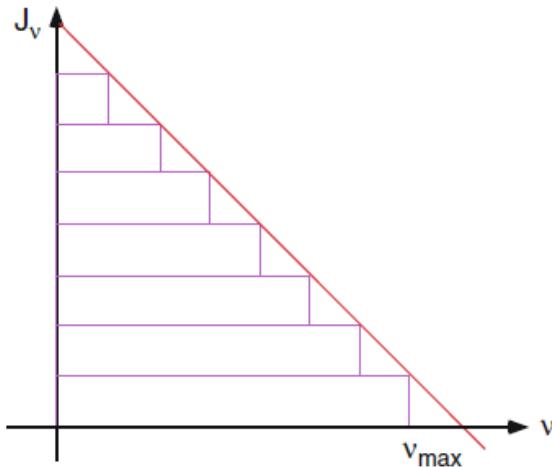
In Abbildung 2.2 ist die Wechselwirkung mit einer einzigen dünnen Schicht durch das unterste Rechteck dargestellt. In jeweils hintereinanderliegenden dünnen Schichten nimmt die maximale Frequenz der Bremsstrahlung ab, da die Elektronen in der vorherigen Schicht Energie verlieren [4, S. 7]. Daraus ergibt sich, dass die spektrale Leistungsdichte der Bremsstrahlung linear mit steigender Frequenz abnimmt. Die Intensität in einem Frequenzintervall  $dJ$  entspricht dem Produkt aus der Photonenergie  $E$  und dem Photonenfluss  $dn$  innerhalb des Intervalls [9, S. 59]. Über Gleichung 2.2 und 2.5 lässt sich der Photonenfluss beschreiben.

$$dJ = E \cdot dn = h \cdot v \cdot dn = J_v \cdot dv \quad (2.6)$$

$$\Leftrightarrow dn = \frac{J_v}{h \cdot v} \cdot dv \quad (2.7)$$

Neben der Bremsstrahlung wird auch charakteristische Strahlung emittiert. Diese entsteht, wenn eines der beschleunigten Elektronen mit einem Elektron aus den inneren Schalen der Atomhülle interagiert. Bei einer solchen Interaktion wird das innere Elektron herausgeschlagen. Die vakante Position wird durch ein Elektron aus einer der äußeren Schale aufgefüllt und Strahlung, deren Energie der Energiedifferenz zwischen den Schalen entspricht, emittiert. Da der energetische Abstand zwischen den Schalen diskrete Werte hat, entstehen scharfe Karten im Spektrum. [2, S. 20]

Neben den Prozessen, die für die Bremsstrahlung und charakteristische Strahlung verantwortlich sind, kommt es zum Auger-Effekt. Bei diesem Effekt wird ein Photon der cha-



Quelle: [4, S. 8]

**Abbildung 2.2.:** Kontinuierliches Bremsspektrum einer Röntgenröhre. Spektrale Leistungsdichte  $J_v$  über der Frequenz  $v$

rakteristischen Strahlung unmittelbar nach seiner Emission durch Emission eines anderen Elektrons absorbiert. Für Elemente mit großer Ordnungszahl ist die Wahrscheinlichkeit für diese Wechselwirkung gering. [2, S. 22].

Die insgesamt abgegebene Bremsstrahlungsintensität  $J_{\text{ges}}$  ergibt sich durch Integration über die spektrale Leistungsdichte [9, S. 60].

$$J_{\text{ges}} = \int_0^{v_{\text{max}}} J_v dv \quad (2.8)$$

Der Wirkungsgrad  $\eta$  der Bremsstrahlung ist definiert als das Verhältnis aus abgegebener Bremsstrahlungsintensität zur Leistung  $P_A$  der Elektronenstrahlung [9, S. 60]. Letztere entspricht dem Produkt aus Beschleunigungsspannung  $U_A$  und Anodenstrom  $I_A$  [4, S. 12].

$$\eta = \frac{J_{\text{ges}}}{P_A} \quad (2.9)$$

$$\text{mit: } P_A = U_A \cdot I_A \quad (2.10)$$

Der Wirkungsgrad ist proportional zur Beschleunigungsspannung  $U_A$ , der Ordnungszahl  $Z$  des Anodenmaterials und einer Konstanten  $k$  [9, S. 61]. Mit den Gleichungen 2.9 und 2.10 lässt sich abgestrahlte Strahlungsleistung  $J_{\text{ges}}$  aus den Röntgenröhreneigenschaften ermitteln.

$$\eta = k \cdot U_A \cdot Z \quad (2.11)$$

$$\Rightarrow J_{\text{ges}} = k \cdot U_A^2 \cdot I_A \cdot Z \quad (2.12)$$

$$\text{mit: } k = 1,1 \cdot 10^{-9}$$

### 2.1.1. Schwächung von Röntgenstrahlung

Photonen, die Materie durchdringen, werden absorbiert oder gestreut. Betrachtet man ein enges Bündel von Photonen mit identischer Energie und einem Photonenfluss  $n$  durch eine

dünne Materialschicht der Dicke  $\Delta x$ , wird der Photonenfluss um  $\Delta n$  verringert. Diese Verringerung ist proportional zum Photonenfluss, der Materialdicke und einer materialspezifischen Konstanten  $\mu$  [9, S. 88].

$$-\Delta n = \mu \cdot n \cdot \Delta x \quad (2.13)$$

Die Proportionalitätskonstante  $\mu$  wird als *Schwächungskoeffizient* bezeichnet. Für aufeinanderfolgende dünne Schichten ergibt sich die Differentialgleichung 2.14, welche für den Anfangswert  $n(x = 0\text{cm}) = n_0$  die Lösung 2.15, die als Schwächungsgesetz bezeichnet wird, hat.

$$\frac{dn}{dx} = -\mu \cdot n \quad (2.14)$$

$$n(x) = n_0 \cdot e^{-\mu \cdot x} \quad (2.15)$$

Der Schwächungskoeffizient  $\mu$  hängt von den Eigenschaften der Photonen und des Materials ab. Insbesondere beeinflussen die Photonenergie  $E$  und Ordnungszahl  $Z$  sowie Dichte  $\rho$  des schwächenden Materials die Größe des Schwächungskoeffizienten. Durch Einführung der Flächendichte  $d = \rho \cdot x$  sowie Normierung des Schwächungskoeffizienten  $\mu$  auf die Materialdichte  $\rho$ , ergibt sich ein Schwächungsgesetz, das unabhängig von Temperatur und Dichte des Materials ist [9, S. 88]. Über den Zusammenhang zwischen der Strahlungsintensität  $J$  und der Photonenzahl  $n$  aus Gleichung 2.6 (S. 8), lässt sich das Schwächungsgesetz auch für die Strahlungsintensität angeben.

$$\mu_m = \frac{\mu}{\rho} \quad (2.16)$$

$$n(d) = n_0 \cdot e^{-\mu_m \cdot d} \Leftrightarrow J(d) = J_0 \cdot e^{-\mu_m \cdot d} \quad (2.17)$$

Der neu definierte Koeffizient  $\mu_m$  wird *Massenschwächungskoeffizient* genannt.

Um Schwächung durch Wechselwirkungen mit einzelnen Teilchen zu beschreiben, wird der *Wirkungsquerschnitt*  $\sigma$  definiert. Er stellt anschaulich eine Fläche um ein Teilchen dar. Trifft ein Photon diese Fläche, kommt es zu einer Wechselwirkung als Streuung oder Absorption. Der Wirkungsquerschnitt ist der Quotient aus Massenschwächungskoeffizient und Teilchenanzahl pro Masse. Letztere kann über die Avogadrokonstante  $N_A$  und molare Masse  $M$  beschrieben werden. [9, S. 89]

$$\sigma = \mu_m \cdot \frac{M}{N_A} \quad (2.18)$$

$$\text{mit: } N_A = 6,022\,140\,76 \cdot 10^{23} \frac{1}{\text{mol}}$$

### Absorptions- und Streuprozesse

Bei Durchgang von Photonen durch Materie kommt es zur Wechselwirkung. Mehrere Wechselwirkungsprozesse tragen zur Schwächung von Röntgenstrahlung bei. Bei Absorptionsprozessen geben Photonen ihre ganze oder einen Teil ihrer Energie an ein Elektron ab, sodass

es zur Ionisation kommt. Die Schwächung durch Absorption wird mit dem Absorptionskoeffizienten  $\mu_{\text{abs}}$  analog zum Massenschwächungskoeffizient  $\mu_m$  charakterisiert. Neben Absorptionsprozessen kommt es zu Streuprozessen. Bei diesen Wechselwirkungen kommt es nicht zur Absorption der Photonen, sondern zu einem Stoßvorgang mit einem Elektron. Dabei werden Photonen aus dem primären Bündel gelenkt. Der Streukoeffizient  $\mu_{\text{str}}$  beschreibt dabei den Anteil der Photonenergie, die aus dem Bündel herausläuft. Die gesamte Schwächung durch Absorption und Streuung  $\mu_{\text{tot}}$  ergibt sich als Summe des Absorptionskoeffizienten  $\mu_{\text{abs}}$  und Streuquerschnittes  $\mu_{\text{str}}$ . [9, S. 89–91]

$$\mu_{\text{tot}} = \mu_{\text{abs}} + \mu_{\text{str}} \quad (2.19)$$

**Die Rayleigh-Streuung** kommt zustande, da die Röntgenstrahlung als elektromagnetische Welle die Elektronen in Schwingung versetzt. Da beschleunigte Ladungen selbst elektromagnetische Wellen aussenden, haben die gestreuten Photonen dieselbe Frequenz wie die einfallenden Photonen. [4, S. 16]

**Die Compton-Streuung** beschreibt einen Vorgang, bei dem ein Photon mit einem Elektron stößt. Dabei kommt es zu einem Energie- und Impulsübertrag vom Photon auf das Elektron, sodass sich die Richtung und Energie des Photons verändern. Der Energieübertrag ist dabei abhängig vom Streuwinkel. [4, S. 16]

**Der Photoeffekt** liegt vor, wenn ein Photon seine Energie vollständig auf ein Elektron überträgt. Dabei kommt es zur Ionisation der Atome und zu einem Transport des Elektrons in das Leitungsband oder aus dem Festkörper heraus. [4, S. 16]

**Die Paarbildung** ist ein Prozess bei dem ein Photon mit ausreichend großer Energie ein Elektron-Positron-Paar erzeugt. [4, S. 16]

Der Photoeffekt und die Compton-Streuung sowie -Absorption tragen bei diagnostischen Röntgenenergien wesentlich zum Schwächungsverhalten von Wasser bei. Da menschliches Gewebe primär aus Wasser besteht, werden diese Schwächungsprozesse genauer beschrieben.

### Photoeffekt

Der Photoeffekt, der auch als photoelektrische Absorption [9, S. 102] bezeichnet werden kann, beschreibt einen vollständigen Absorptionsprozess von Photonen. Ein Photon gibt seine Energie  $h \cdot v$  vollständig an ein mit der Energie  $E_{nQ}$  gebundenes Elektron ab und schlägt dieses aus dem Atom heraus. Die kinetische Energie des Elektrons beträgt danach  $E_{\text{kin}} = h \cdot v - E_{nQ}$ . Mit steigender Quantenenergie können auch Elektronen aus weiter innen liegenden Schalen, die mit größerer Energie gebunden sind, herausgeschlagen werden. Da die Energiedifferenzen zwischen den Schalen diskrete Werte aufweisen, gibt es Energien, bei denen die Absorption sprunghaft ansteigt. [10, S. 838]

Nach [10, S. 839] kann der Wechselwirkungsquerschnitt der photoelektrischen Absorption  $\sigma_{\text{pho}}$  in Abhängigkeit von der Ordnungszahl  $Z$  und Photonfrequenz  $v$  angegeben werden. Ebenso spielen die Feinstrukturkonstante  $\alpha$ , der klassische Elektronenradius  $r_0$ , die relativis-

tische Grenzfrequenz  $\omega_0$  und die Hauptquantenzahl  $n_Q$  eines Elektrons eine Rolle.

$$\sigma_{\text{pho}} \approx \frac{\pi}{12} \cdot \frac{\alpha^3 \cdot Z^4}{n_Q^2} \cdot r_0^2 \cdot \frac{\omega_0^3}{v^3} \quad (2.20)$$

$$\alpha = \frac{1}{137}$$

$$r_0 = 2,8 \cdot 10^{-15} \text{ m}$$

$$\omega_0 = 7,763\,393\,9 \cdot 10^{20} \frac{1}{\text{s}}$$

$$n_Q = 1,2,3,\dots$$

Über Gleichung 2.18 lässt sich durch die Abhängigkeit der molaren Masse eines Atoms  $M_{\text{At}}$  von seiner Ordnungszahl  $Z$  ableiten, dass der Massenabsorptionskoeffizient des Photoeffektes  $\mu_{\text{pho}}$  entsprechend Gleichung 2.22 von  $v$  und  $Z$  abhängt [9, S. 103]. Die Masse eines Atoms wird dabei über sein Ordnungszahl  $Z$  und die Masse des Wasserstoffatoms  $m_H$  abgeschätzt.

$$M_{\text{At}} = m_{\text{At}} \cdot N_A \approx 2 \cdot Z \cdot m_H \cdot N_A \quad (2.21)$$

$$\boxed{\mu_{\text{pho}} \propto \frac{Z^3}{v^3}} \quad (2.22)$$

Die photoelektrische Absorption steigt zur dritten Potenz mit der Ordnungszahl und sinkt zur dritten Potenz mit der Photonenenergie. Materialien, die aus Elementen mit hoher Ordnungszahl bestehen, sind starke Absorber. Photonen mit geringer Energie werden am stärksten absorbiert.

### Compton-Streuung

Bei der Compton-Streuung stößt ein Photon mit einem Elektron. Dabei gibt das Photon einen Teil seiner Energie an das Elektron ab und verändert seine Trajektorie. Ein Elektron, das bei einer Kollision ein Photon in das Raumwinkelintervall  $d\Omega$  streut, verfügt für diese Kollision über einen differentiellen Wirkungsquerschnitt  $d\sigma_K$ . Dieser kann als explizite Funktion des Streuwinkels  $\vartheta$  und der initialen Photonenenergie  $h \cdot v_0$  angegeben werden [11, S. 250].

$$d\sigma_K = r_0^2 \cdot d\Omega \cdot \left( \frac{1 + \cos^2 \vartheta}{2 \cdot (1 + a)^2} \right) \cdot \left( 1 + \frac{a^2 \cdot (1 - \cos \vartheta)^2}{(1 + \cos^2 \vartheta) \cdot [1 + a \cdot (1 - \cos \vartheta)]} \right) \quad (2.23)$$

$$\text{mit: } a = \frac{h \cdot v_0}{m_0 \cdot c^2} \quad \text{und} \quad m_0 = 9,109\,383\,701\,5 \cdot 10^{-31} \text{ kg}$$

Der totale Wirkungsquerschnitt für eine Kollision ist unabhängig vom Streuwinkel  $\vartheta$  und lässt sich durch Integration über alle Raumwinkel berechnen [11, S. 263].

$$\begin{aligned} \sigma_K = \int_0^\pi d\sigma_K d\Omega &= 2 \cdot \pi \cdot r_0^2 \cdot \left[ \frac{1+a}{a^2} \cdot \left( \frac{2 \cdot (1+a)}{1+2 \cdot a} - \frac{\log(1+2 \cdot a)}{a} \right) \right. \\ &\quad \left. + \frac{\log(1+2 \cdot a)}{2 \cdot a} - \frac{1+3 \cdot a}{(1+2 \cdot a)^2} \right] \end{aligned} \quad (2.24)$$

Um die Schwächung eines Photonenbündels durch Compton-Streuung zu beschreiben, lässt sich analog zu Gleichung 2.18 ein Massenschwächungskoeffizient  $\mu_C$  definieren. Dazu wird die molare Masse der Elektronen  $M_{El}$  aus der molaren Masse der Atome  $M_{At}$  dividiert durch die Anzahl der Elektronen je Atom, was der Ordnungszahl  $Z$  entspricht, berechnet [11, S. 271]. Die Größe  $N_{El}$  ist die Elektronenanzahl je Masse.

$$\mu_C = \sigma_K \cdot \frac{N_A}{M_{El}} = \sigma_K \cdot N_A \cdot \frac{Z}{M_{At}} \quad (2.25)$$

$$\Rightarrow \boxed{\mu_C = \sigma_K \cdot N_{El}} \quad (2.26)$$

Der Quotient  $\frac{Z}{M_{At}}$  ist für alle Elemente außer Wasserstoff nahezu konstant [11, S. 271]. Grund dafür ist die Abhängigkeit der molaren Masse von der Ordnungszahl  $M_{At} \approx 2 \cdot Z \frac{g}{mol}$ . Somit ist auch die Elektronenanzahl je Masse  $N_{El} \approx 2,72 \cdot 10^{23} \frac{1}{g}$  konstant. Da elementarer Wasserstoff aus nur einem Proton besteht, weicht dieser Wert ungefähr um den Faktor 2 ab. Somit hängt der Massenschwächungskoeffizient für Elemente nur von der Dichte des Materials und über der Wechselwirkungsquerschnitt auch von der Photonenergie ab.

Für ein Bündel aus Photonen, das durch ein Material mit dem Massenschwächungskoeffizienten für die Compton-Streuung  $\mu_C$  und der Flächendichte  $d = \rho \cdot x$  durchdringt, lässt sich eine Wahrscheinlichkeit  $p_{WW}$  definieren. Diese Größe zeigt für jedes Photon die Wahrscheinlichkeit an, dass es zu einer Wechselwirkung kommt. Sie entspricht dem Verhältnis aus der Anzahl der wechselwirkenden Photonen  $n_0 - n(d)$  und der Gesamtanzahl  $n_0$ . Mit Gleichung 2.17 kann die Wahrscheinlichkeit als Funktion des Massenschwächungskoeffizienten und der Flächendichte dargestellt werden.

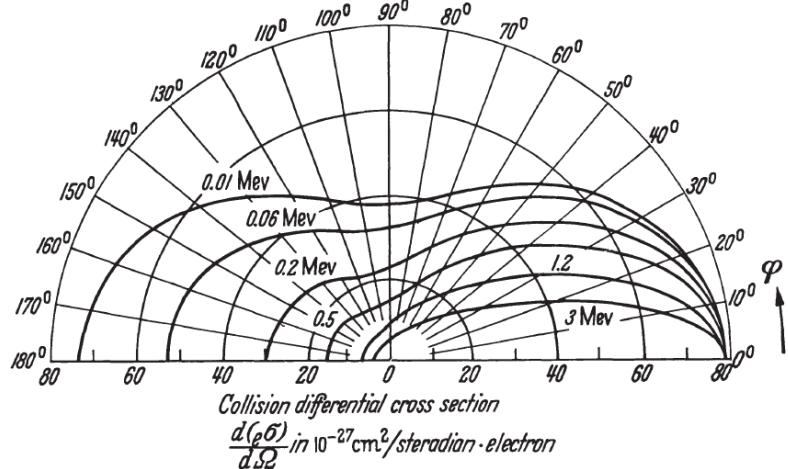
$$p_{WW}(d) = \frac{n_0 - n(d)}{n_0} = 1 - \frac{n(d)}{n_0} \quad (2.27)$$

$$\boxed{p_{WW}(d) = 1 - e^{-\mu_C \cdot d}} \quad (2.28)$$

Wenn es zu einer Kollision kommt, ist die Streuwinkelverteilung nach Gleichung 2.23 abhängig von der Photonenergie. In Abbildung 2.3 sind für alle Winkel zwischen  $0^\circ$  und  $180^\circ$  sowie verschiedene Quantenenergien die jeweiligen Wirkungsquerschnitte pro Raumwinkelintervall  $\frac{d\sigma_K}{d\Omega}$  in Polarkoordinaten dargestellt. Der Wechselwirkungsquerschnitt für ein Streuereignis, bei dem das Photon mit einem Winkel zwischen  $\theta_0$  und  $\theta_1$  gestreut wird, lässt sich durch Integration über den differentiellen Wirkungsquerschnitt  $d\sigma_K$  bestimmen [11, S. 255].

$$\sigma_K(\phi_0 \leq \vartheta \leq \phi_1) = \int_{\phi_0}^{\phi_1} \frac{d\sigma_K}{d\Omega} \cdot 2 \cdot \pi \cdot \sin \vartheta d\vartheta \quad (2.29)$$

Bei einem Streuprozess kommt es zu einer Teilabsorption der Photonenergie. Die Wellenlänge des Photons  $\lambda_0$ , die über Gleichung 2.1 und 2.2 (S. 7) mit der Energie verknüpft ist, ändert sich bei einer Streuung um einen vom Streuwinkel  $\vartheta$  abhängigen Betrag  $\Delta\lambda$  [11,



Quelle: [11, S. 251]

**Abbildung 2.3.:** Winkelverteilung der gestreuten Photonen bei Compton-Streuung S. 237].

$$\lambda = h \cdot \frac{c}{E} \quad (2.30)$$

$$\Delta\lambda = \lambda' - \lambda_0 = \frac{h}{m_0 \cdot c} \cdot (1 - \cos \theta) \quad (2.31)$$

Die Energie des Photons nach dem Streuereignis  $E'$  lässt sich über Gleichung 2.30 in Abhängigkeit der initialen Energie  $E_0$  und dem Streuwinkel berechnen.

$$\begin{aligned} \Delta\lambda &= \frac{h \cdot c}{E'} - \frac{h \cdot c}{E_0} = \frac{h}{m_0 \cdot c} \cdot (1 - \cos \theta) \\ \Leftrightarrow E' &= \left( \frac{1 - \cos \vartheta}{m_0 \cdot c} + \frac{1}{E_0} \right)^{-1} \end{aligned} \quad (2.32)$$

### Röntgenschwächung in Wasser

Die Schwächungseigenschaften von Wasser spielen bezüglich medizinischer Diagnostik mit Röntgenstrahlung die primäre Rolle, da menschliches Gewebe zu einem Großteil aus Wasser besteht. Es handelt sich bei Wasser um ein Molekül bestehend aus zwei Wasserstoff- und einem Sauerstoffatom. Aus diesem Grund können die Schwächungskoeffizienten des Moleküls aus den Koeffizienten der Bestandteile durch Gewichtung mit den Massenanteilen  $w$  ermittelt werden [11, S. 274]. Für Wasser ergibt sich die Gewichtung nach Gleichung 2.33.

$$\begin{aligned} \mu_{m, H_2O} &= \mu_{m, H} \cdot w_H + \mu_{m, O} \cdot w_O \\ &= \mu_{m, H} \cdot \frac{2 \cdot m_H}{2 \cdot m_H + m_O} + \mu_{m, O} \cdot \frac{m_O}{2 \cdot m_H + m_O} \\ &= \mu_{m, H} \cdot \frac{1}{9} + \mu_{m, O} \cdot \frac{8}{9} \\ \text{mit: } \frac{m_O}{m_H} &= 16 \end{aligned} \quad (2.33)$$

Aus den Wechselwirkungsprozessen ergeben sich verschiedene Koeffizienten, die jeweils zum totalen Massenschwächungskoeffizienten  $\mu_{tot}$  beitragen. Dieser setzt sich aus den Beiträgen

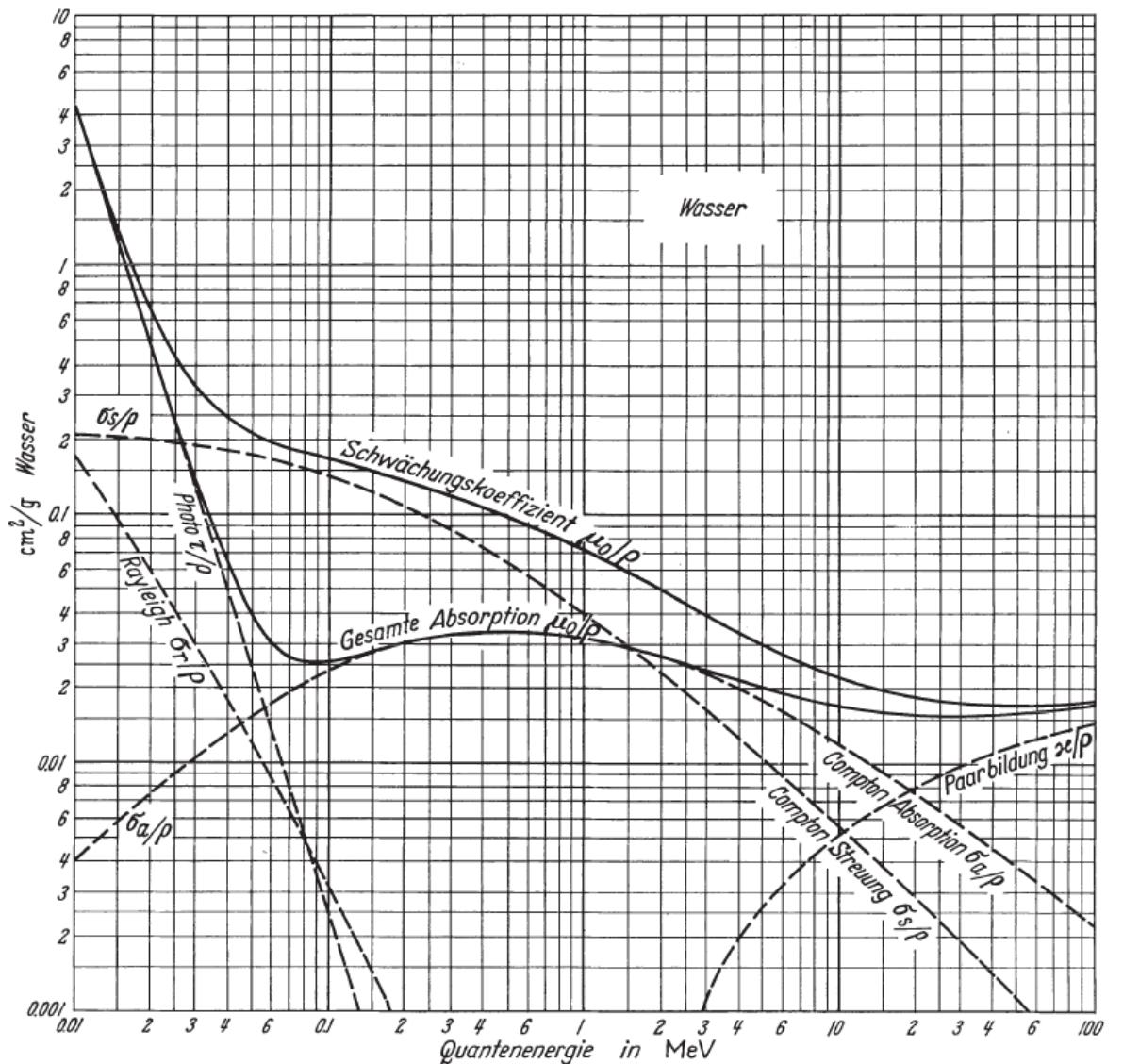
durch Absorption  $\mu_{\text{abs}}$  und Streuung  $\mu_{\text{str}}$  zusammen. Ersterer enthält Beiträge des Photoeffektes  $\mu_{\text{pho}}$ , der Compton-Absorption  $\mu_{C, \text{abs}}$  und der Paarbildung  $\mu_{\chi}$ . Zur Schwächung durch Streuung tragen die Compton-Streuung  $\mu_{C, \text{str}}$  und die Rayleigh-Streuung  $\mu_{\tau}$  bei.

$$\mu_{\text{tot}} = \mu_{\text{abs}} + \mu_{\text{str}} = \mu_{\text{pho}} + \mu_{C, \text{abs}} + \mu_{\chi} + \mu_{C, \text{str}} + \mu_{\tau} \quad (2.34)$$

$$\text{mit: } \mu_{\text{abs}} = \mu_{\text{pho}} + \mu_{C, \text{abs}} + \mu_{\chi} \quad (2.35)$$

$$\text{und } \mu_{\text{str}} = \mu_{C, \text{str}} + \mu_{\tau}$$

In Abbildung 2.4 sind die Massenschwächungskoeffizienten der verschiedenen Wechselwirkungsprozesse und deren Summen in einem Energiebereich von 10 keV bis 100 MeV dargestellt. Beide Achsen sind im logarithmischen Maßstab dargestellt. In der Abbildung weichen die Bezeichnungen für die einzelnen Koeffizienten von den hier gewählten Bezeichnungen ab.



Quelle: [11, S. 274]

**Abbildung 2.4.:** Massenschwächungskoeffizienten und Massenabsorptionskoeffizienten von Röntgenstrahlung in Wasser

Bei Energien bis 54 keV dominiert der Photoeffekt. Die Gesamtabsorption  $\mu_{\text{abs}}$  ist in diesem Bereich proportional zu  $v^{-3}$ . Zwischen 54 keV und 20 MeV dominiert die Compton-Absorption. In diesem Bereich ist die Schwächung durch Absorption nahezu konstant. Bei noch größeren Energien gibt die Paarbildung den Verlauf vor. Die Compton-Streuung  $\mu_{\text{str}}$  leistet für Energien bis 100 keV einen fast konstanten Beitrag zur Gesamtschwächung. Für größere Energien verläuft der Koeffizient mit  $v^{-1}$  [11, S. 263].

## 2.2. Mathematische Grundlagen der Computertomographie

Ziel der Computertomographie ist die Bestimmung der Funktionswerte einer unbekannten zweidimensionalen Funktion  $f(x, y)$ . Diese Funktion beschreibt die Röntgenabsorption für eine dünne Schicht, die parallel zu den transmittierten Röntgenstrahlen liegt.

Johann Radon veröffentlichte 1917 einen Artikel „[ü]ber die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten“ [12]. In diesem Artikel wird beschrieben, dass es möglich ist, eine Funktion  $f(x, y)$  allein durch Kenntnis der geradlinigen Integralwerte  $p$  zu bestimmen. Die Funktion muss dafür stetig sein und außerhalb eines Gebietes um den Ursprung zu null konvergieren [12, S. 263]. In der Computertomographie entspricht die Funktion  $f(x, y)$  dem totalen Schwächungskoeffizienten  $\mu_{\text{tot}}$ . Die geradlinigen Integralwerte, die auch als *Linienintegrale* bezeichnet werden, sind ein Maß für die Schwächung eines Röntgenstrahls entlang einer geradlinigen Trajektorie.

Als *Radon-Transformation* wird die Transformation bezeichnet, bei dem der Funktion  $f(x, y)$  alle möglichen Linienintegrale  $p(\theta, s)$  entlang der Geraden  $\ell$  zugeordnet werden. Diese Geraden sind in einem bestimmten Ordnungsschema, das in den Abbildungen 2.5 dargestellt ist, definiert. Die Geraden werden durch den Winkel  $\theta$  und die Strecke  $s$  eindeutig charakterisiert [13, S. 49].

$$\ell_{\theta, s}: x \cdot \cos \theta + y \cdot \sin \theta = s \quad (2.36)$$

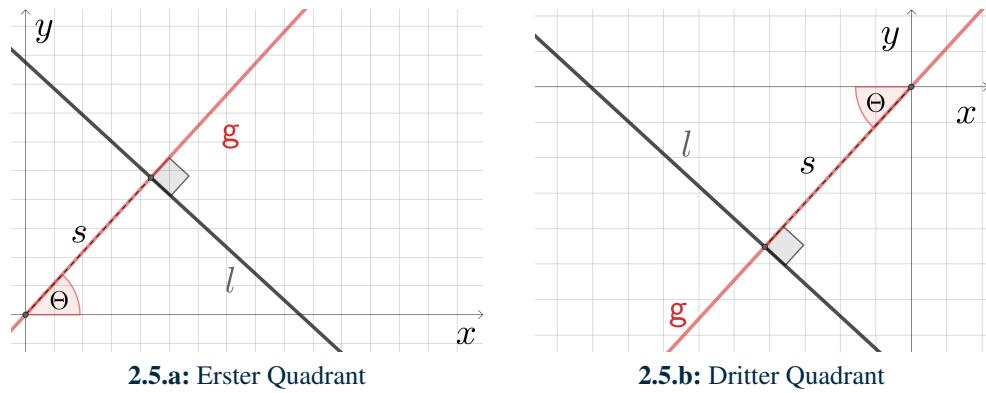
Der Winkel  $\theta$  entspricht dem Winkel zwischen der  $x$ -Achse und dem Lot  $g$  der Geraden, das durch den Ursprung verläuft. Die Strecke  $s$  ist der kürzeste Abstand der Geraden  $\ell$  zum Ursprung, was der Länge des Lotes  $g$  gleich ist. Um Redundanzen in der Zuordnung zu vermeiden, wird der Winkel  $\theta$  auf das Intervall  $[0, \pi]$  und die Strecke  $s$  auf das Intervall  $[-s_{\max}, s_{\max}]$  mit  $s_{\max} > 0$  beschränkt. Die Redundanz kommt zustande, da die Gerade  $\ell_{\theta, s}$  gleich der Geraden  $\ell_{\theta+\pi, -s}$  ist.

$$x \cdot \cos(\theta + \pi) + y \cdot \sin(\theta + \pi) = -s \Leftrightarrow -x \cdot \cos \theta - y \cdot \sin \theta = -s \quad (2.37)$$

$$\Leftrightarrow x \cdot \cos \theta + y \cdot \sin \theta = s \quad (2.38)$$

Liegt das Lot im ersten oder zweiten Quadranten wird festgelegt, dass  $s > 0$  ist (Abb. 2.5.a). Für den dritten und vierten Quadranten gilt  $s < 0$  (Abb. 2.5.b).

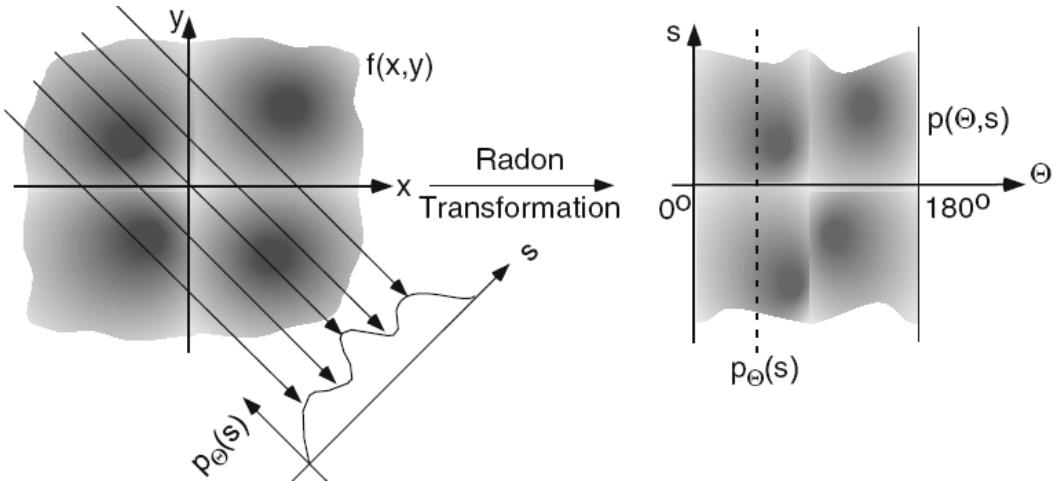
Das Linienintegral entspricht der Integration entlang der Geraden  $\ell$  mit dem Parameter  $t$ , welcher den vorzeichenbehafteten Abstand eines Punktes auf der Geraden  $\ell$  zum Schnittpunkt

**Abbildung 2.5.:** Ordnungsschema für Linienintegrale in der kartesischen Ebene

(Fußpunkt) mit dem Lot  $g$  darstellt [13, S. 50].

$$p(\theta, s) = \int_{\ell_{\theta,s}} f(x, y) dt \quad (2.39)$$

Alle Linienintegrale, die einem konstanten Winkel  $\theta$  zugeordnet sind, werden als *Projektion*  $p_\theta(s)$  bezeichnet. In Abbildung 2.6 sind zu der Funktion  $f(x, y)$  einige Linienintegrale entlang der Geraden  $\ell$  und die dazugehörige Projektion  $p_\theta(s)$  dargestellt. Das Ergebnis der Radon-Transformation ist wiederum eine zweidimensionale Funktion, die die Projektionen für alle Winkel enthält. Die Darstellung der Projektionen als Bild wird *Sinogramm* genannt.



Quelle: [4, S. 133]

**Abbildung 2.6.:** Die Radon-Transformation

### 2.2.1. Fourier-Scheiben-Theorem

Durch Rotation um den Winkel  $\theta$  im Uhrzeigersinn können mithilfe der Rotationsmatrix die auf der Geraden  $\ell$  liegenden Punkte  $(x, y)$  in das kartesische Koordinatensystem aus  $s$  und  $t$  überführt werden [13, S. 58]. Dabei ist  $s$  die Länge des Lotes und  $t$  der Abstand eines Punktes auf der Geraden zum Fußpunkt. Anschaulich werden die Gerade  $\ell$  und ihr Lot  $g$  im Uhrzeigersinn gekippt, bis das Lot  $g$  auf der  $x$ -Achse liegt. Beschrieben wird die Rotation durch jene Rotationsmatrix, die eine Rotation gegen den Uhrzeigersinn beschreibt. Daher

wird der Winkel mit  $-1$  multipliziert.

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} \cdot \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad (2.40)$$

Damit kann die Projektion  $p_\theta(s)$  zu einem Winkel unter Berücksichtigung von Gleichung 2.39 beschrieben werden. Die Fourier-Transformierte  $P_\theta(\omega)$  der Projektionen bezüglich des Abstandes  $s$  ist über Gleichung 2.42 gegeben. Durch Einsetzen von Gleichung 2.41 in Gleichung 2.42 ergibt sich Gleichung 2.43.

$$p_\theta(s) = \int_{-\infty}^{\infty} f(s, t) dt \quad (2.41)$$

$$P_\theta(\omega) = \mathcal{F}\{p_\theta(s)\} = \int_{-\infty}^{\infty} p_\theta(s) \cdot e^{-j \cdot 2 \cdot \pi \cdot \omega \cdot s} ds \quad (2.42)$$

$$\begin{aligned} P_\theta(\omega) &= \int_{-\infty}^{\infty} \left( \int_{-\infty}^{\infty} f(s, t) dt \right) \cdot e^{-j \cdot 2 \cdot \pi \cdot \omega \cdot s} ds \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(s, t) \cdot e^{-j \cdot 2 \cdot \pi \cdot \omega \cdot s} dt ds \end{aligned} \quad (2.43)$$

Für eine Transformation in das ursprüngliche Koordinatensystem  $(x, y)$  wird für den Abstand  $s$  der Ausdruck aus Gleichung 2.36 eingesetzt. Für die Differentiale  $dt ds$  können direkt die Differentiale  $dxdy$  eingesetzt werden, da es sich bei beiden Koordinatensystemen um kartesische handelt, die zueinander nicht skaliert, sondern nur rotiert sind [2, S. 166].

$$P_\theta(\omega) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \cdot e^{-j \cdot 2 \cdot \pi \cdot \omega \cdot (x \cdot \cos \theta + y \cdot \sin \theta)} dxdy \quad (2.44)$$

$$\begin{aligned} &= \mathcal{F}\{f(x, y)\} \\ &= F(\omega \cdot \cos \theta, \omega \cdot \sin \theta) \end{aligned} \quad (2.45)$$

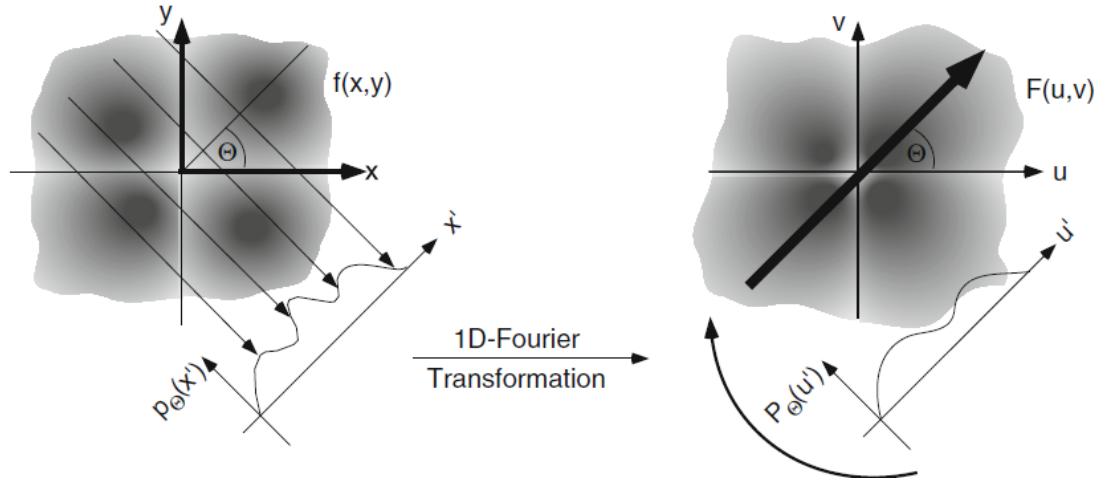
Mit  $u = \omega \cdot \cos \theta$  und  $v = \omega \cdot \sin \theta$  entspricht  $P_\theta(\omega)$  aus Gleichung 2.44 der zweidimensional Fourier-transformierte  $F(u, v)$  von  $f(x, y)$ . Durch inverse Fourier-Transformation ließe sich die gesuchte Funktion ermitteln.

Das Fourier-Scheiben-Theorem besagt, dass eine unbekannte Funktion  $f(x, y)$  durch Messung der Projektionen  $p_\theta(s)$  bestimmt werden kann. Die eindimensional Fourier-transformierte  $P_\theta(\omega)$  der Projektionen beschreiben die Werte von  $F(u, v)$  auf Radialstrahlen. Durch inverse Fourier-Transformation von  $F(u, v)$  ergibt sich die gesuchte Funktion  $f(x, y)$ . [4, S. 135]

In Abbildung 2.7 ist das Fourier-Scheiben-Theorem für eine Projektion mit beliebigem Winkel grafisch abgebildet.

## 2.2.2. Bildrekonstruktion

Ziel der Bildrekonstruktion ist die Bestimmung der Funktion  $f(x, y)$  aus den gemessenen Projektionen  $p_\theta(s)$ . Dazu stehen die analytische und die iterative Bildrekonstruktion zur Auswahl. Die *analytische Bildrekonstruktion* basiert auf der gefilterten Rückprojektion [14, S. 166]. Diese Methode nutzt vereinfachte Systemmodelle, sodass statistische und geometrische Eigenschaften der Röntgenstrahlung nicht berücksichtigt werden [14, S. 172]. Würden diese



Quelle: [4, S. 135]

**Abbildung 2.7.: Das Fourier-Scheiben Theorem**

Eigenschaften berücksichtigt, existiert keine analytische Lösung. Im Gegensatz dazu steht die *iterative Bildrekonstruktion*, die eine bessere Bildqualität liefert und Artefakte reduziert [14, S. 172].

Im Simulator ist die analytische Bildrekonstruktion in Parallelstrahlgeometrie implementiert. Der Ausgangspunkt der analytischen Bildrekonstruktion sind Gleichung 2.44 und 2.45. Nach diesen besteht über die zweidimensionale Fourier-Transformierte  $F(u, v)$  ein Zusammenhang zwischen der gesuchten Funktion und den gemessenen Projektionen. Zur Herleitung der Gleichungen für die analytische Bildrekonstruktion in Parallelstrahlgeometrie wird sich an [13, S. 63–64] orientiert und Details ergänzt.

Durch inverse Fourier-Transformation lässt sich nach Gleichung 2.46 die gesuchte Funktion bestimmen.

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) \cdot e^{j \cdot 2 \cdot \pi \cdot (u \cdot x + v \cdot y)} du dv \quad (2.46)$$

Die kartesischen Koordinaten  $u$  und  $v$  werden in die Polarkoordinaten  $\theta$  und  $\omega$  überführt. Der Faktor  $\omega$  bei der Transformation der Differentiale in Gleichung 2.49 ist notwendig, da das Flächenelement  $du dv$  dem Flächenelement eines Ringsegmentes entspricht. Die Fläche dieses Ringsegmentes ist durch die Seitenlängen  $d\omega$  und  $\omega \cdot d\theta$  gegeben [2, S. 130].

$$u = \omega \cdot \cos \theta \quad (2.47)$$

$$v = \omega \cdot \sin \theta \quad (2.48)$$

$$du dv = \omega d\omega d\theta \quad (2.49)$$

Die inverse Fourier-Transformation in Polarkoordinaten ist damit durch Gleichung 2.50 gegeben. Das Integral lässt sich in das Intervall zwischen  $[0, \pi]$  und  $[\pi, 2 \cdot \pi]$  zerlegen. Die Integrationsgrenzen sind in beiden Teilen identisch, da im zweiten Teil der Winkel  $\theta$  um  $\pi$

verschoben wird.

$$f(x, y) = \int_0^{2\pi} \int_0^{\infty} F(\omega, \theta) \cdot e^{j \cdot 2 \cdot \pi \cdot \omega \cdot (x \cdot \cos \theta + y \cdot \sin \theta)} \cdot \omega d\omega d\theta \quad (2.50)$$

$$\begin{aligned} &= \int_0^{\pi} \int_0^{\infty} F(\omega, \theta) \cdot e^{j \cdot 2 \cdot \pi \cdot \omega \cdot (x \cdot \cos \theta + y \cdot \sin \theta)} \cdot \omega d\omega d\theta \\ &+ \int_0^{\pi} \int_0^{\infty} F(\omega, \theta + \pi) \cdot e^{j \cdot 2 \cdot \pi \cdot \omega \cdot [x \cdot \cos(\theta + \pi) + y \cdot \sin(\theta + \pi)]} \cdot \omega d\omega d\theta \end{aligned} \quad (2.51)$$

Für Polarkoordinaten gilt allgemein, dass die Rotation eines Punktes um  $\pi$  identisch zu einer Multiplikation des Betrages mit  $-1$  ist. Daher gilt die Beziehung in Gleichung 2.52. Eine Phasenverschiebung um  $\pi$  des Argumentes im Sinus oder Kosinus ist ebenso äquivalent zu einer Multiplikation mit  $-1$  (Gl. 2.53).

$$F(\omega, \theta + \pi) = F(-\omega, \theta) \quad (2.52)$$

$$x \cdot \cos(\theta + \pi) + y \cdot \sin(\theta + \pi) = -(x \cdot \cos \theta + y \cdot \sin \theta) \quad (2.53)$$

Durch Einsetzen von 2.52 und 2.53 in den zweiten Summanden der Gleichung 2.51, ergibt sich Gleichung 2.54. Durch Multiplikation der Integrationsvariablen  $\omega$  mit  $-1$  und Vertauschen der Integrationsgrenzen folgt Gleichung 2.55.

$$f(x, y) = \dots + \int_0^{\pi} \int_0^{\infty} F(-\omega, \theta) \cdot e^{-j \cdot 2 \cdot \pi \cdot \omega \cdot (x \cdot \cos \theta + y \cdot \sin \theta)} \cdot \omega d\omega d\theta \quad (2.54)$$

$$= \dots + \int_0^{\pi} \int_{-\infty}^0 F(\omega, \theta) \cdot e^{j \cdot 2 \cdot \pi \cdot \omega \cdot (x \cdot \cos \theta + y \cdot \sin \theta)} \cdot (-\omega) d\omega d\theta \quad (2.55)$$

In Gleichung 2.55, in der das innere Integral nur für negative  $\omega$  ausgewertet wird, gilt für den zweiten Summanden  $-\omega = |\omega|$ . Für den ersten Summanden (Gl. 2.51) gilt  $\omega = |\omega|$ , da dort nur positive  $\omega$  ausgewertet werden. Somit unterscheiden sich die inneren Integrale nur durch ihre Integrationsgrenzen, die aneinander liegen. Wegen der Intervalladditivität von Integralen lassen sich die Integrale so addieren, dass Gleichung 2.56 resultiert.

$$f(x, y) = \int_0^{\pi} \left( \int_{-\infty}^{\infty} F(\omega, \theta) \cdot |\omega| \cdot e^{j \cdot 2 \cdot \pi \cdot \omega \cdot (x \cdot \cos \theta + y \cdot \sin \theta)} d\omega \right) d\theta \quad (2.56)$$

### Filterung

Das innere Integral lässt sich mit Gleichung 2.45 (S. 18) und 2.36 (S. 16) als  $\tilde{p}_{\theta}(s)$  nach Gleichung 2.57 ausdrücken. Der Ausdruck  $\tilde{p}_{\theta}(s)$  wird als *gefilterte Projektionen* bezeichnet, da die Multiplikation von  $P_{\theta}(s)$  mit der Funktion  $|\omega|$  im Frequenzraum eine Filteroperation darstellt. Der Faktor  $|\omega|$  kann allgemeiner *Filterfunktion*  $H(\omega)$  genannt werden.

$$\tilde{p}_{\theta}(s) = \int_{-\infty}^{\infty} P_{\theta}(\omega) \cdot |\omega| \cdot e^{j \cdot 2 \cdot \pi \cdot \omega \cdot s} d\omega \quad (2.57)$$

$$= \mathcal{F}^{-1}\{P_{\theta}(\omega) \cdot |\omega|\} = \mathcal{F}^{-1}\{P_{\theta}(\omega) \cdot H(\omega)\} \quad (2.58)$$

Nach dem *Faltungstheorem* ist das Produkt zweier Fourier-transformierte identisch mit der Fourier-transformierte der gefalteten Funktionen im Ortsraum [2, S. 124]. Die gefilterten

Projektionen  $\tilde{p}_\theta(s)$  lassen sich nach Gleichung 2.61 durch eine Faltung im Zeitbereich der gemessenen Projektionen  $p_\theta(s)$  und der Impulsantwort  $h(s)$  einer *Filterfunktion*  $H(\omega)$  bestimmen.

$$\mathcal{F}\{\tilde{p}_\theta(s)\} = P_\theta(\omega) \cdot H(\omega) = \mathcal{F}\left\{\mathcal{F}^{-1}\{P_\theta(\omega)\} * \mathcal{F}^{-1}\{H(\omega)\}\right\} \quad (2.59)$$

$$= \mathcal{F}\{p_\theta(s) * h(s)\} \quad (2.60)$$

$$\Leftrightarrow \boxed{\tilde{p}_\theta(s) = p_\theta(s) * h(s) = \int_{-\infty}^{\infty} p_\theta(s') \cdot h(s - s') ds'} \quad (2.61)$$

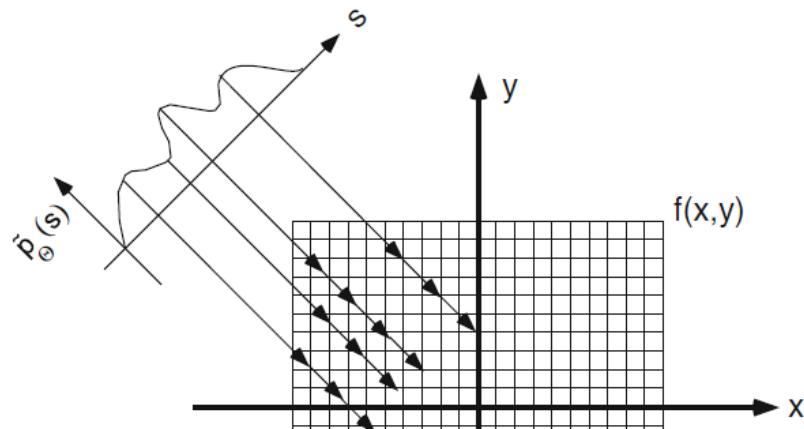
### Rückprojektion

Der letzte Schritt der Rekonstruktion ist die *Rückprojektion*. Aus Gleichung 2.56 und 2.57 ist die zu rekonstruierende Funktion  $f(x, y)$  durch Gleichung 2.62 gegeben. Aus der impliziten Definition der Geraden  $\ell_{\theta,s}: s = x \cdot \cos \theta + y \cdot \sin \theta$  (Gl. 2.36, S.16), die die Länge  $s$  und Winkel  $\theta$  des Lotes mit den Koordinaten des Punktes  $(x, y)$  verknüpft, folgt Gleichung 2.63.

$$f(x, y) = \int_0^\pi \tilde{p}_\theta(s) d\theta \quad (2.62)$$

$$\Leftrightarrow \boxed{f(x, y) = \int_0^\pi \tilde{p}_\theta(x \cdot \cos \theta + y \cdot \sin \theta) d\theta} \quad (2.63)$$

Der Funktionswert von  $f(x, y)$  an einem Punkt  $(x, y)$  ist die Summe aller gefilterten Projektionen  $\tilde{p}_\theta(x \cdot \cos \theta + y \cdot \sin \theta)$ . In Abbildung 2.8 ist zu sehen, wie für einen Winkel den Punkten, die jeweils auf der Geraden  $\ell$  liegen, der Wert einer gefilterten Projektion zugeordnet wird. Durch Aufsummierung der Beiträge jeder gefilterten Projektion werden alle Funktionswerte  $f(x, y)$  bestimmt.



Quelle: [4, S. 151]

**Abbildung 2.8.:** Rückprojektion der gefilterten Projektion zur Bildrekonstruktion

### Zusammenfassung

Zur Rekonstruktion eines Schnittbildes  $f(x, y)$  werden Projektionen  $p_\theta(s)$  gemessen. Eine Projektion entspricht der Summe aller Funktionswert  $f(x, y)$  entlang einer Geraden  $\ell$  mit dem Abstand  $s$  zum Ursprung und dem Lotwinkel  $\theta$ . Das Fourier-Scheiben-Theorem besagt, dass alle eindimensional Fourier-transformierte  $P_\theta(\omega)$  der Projektionen der zweidimensional

Fourier-transformierte  $F(u, v) = \mathcal{F}\{f(x, y)\}$  der gesuchten Funktion entspricht.

Zur Bestimmung der gesuchten Funktion  $f(x, y)$  werden die Projektionen im Orts- oder Frequenzraum gefiltert und rückprojiziert. Bei der Rückprojektion werden für jeden Punkt in  $f(x, y)$  alle Werte der gefilterten Projektionen, deren ursprünglichen Geraden  $\ell$  durch den Punkt laufen, aufsummiert.

### 2.2.3. Diskretisierung

In der Realität ist es nicht möglich, unendlich viele Projektionen mit unendlich vielen Linienintegralen zu messen. Die Anzahl der Projektionen  $N_\theta$  und die Anzahl der Linienintegral pro Projektion  $N_s$  sind endlich. Die Anzahl der Linienintegrale wird als gerade festgelegt. Die Winkelauflösung  $\Delta\theta$  und Ortsauflösung  $\Delta s$  hängen mit der jeweiligen Anzahl der diskreten Werte zusammen. Die Ortsauflösung hängt gleichzeitig von der Größe des Messfeldes  $s_{\text{Mess}} = 2 \cdot s_{\text{max}}$  ab.

$$\theta = k \cdot \Delta\theta \quad \text{mit: } k = 0, 1, 2, \dots, N_\theta - 1 \quad (2.64)$$

$$\Delta\theta = \frac{\pi}{N_\theta} \quad (2.65)$$

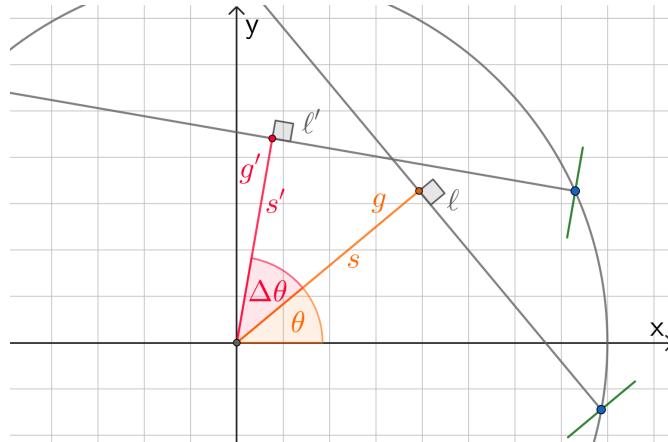
$$s = l \cdot \Delta s \quad \text{mit: } l = \left(-\frac{N_s}{2} + 0,5\right), \left(-\frac{N_s}{2} + 1,5\right), \dots, \left(\frac{N_s}{2} - 0,5\right) \quad (2.66)$$

$$\Delta s = 2 \cdot \frac{s_{\text{max}}}{N_s - 1} \quad (2.67)$$

Die Projektionen sind daher bezogen auf ihre Winkel  $\theta$  und Linienabstände  $s$  diskret. Die diskreten Projektionen können unter Nutzung der Koordinatentransformation in Gleichung 2.40 (S. 18) beschrieben werden.

$$p_{k \cdot \Delta\theta}(l \cdot \Delta s) = \int_{-\infty}^{\infty} f(l \cdot \Delta s \cdot \cos(k \cdot \Delta\theta) - t \cdot \sin(k \cdot \Delta\theta), \\ l \cdot \Delta s \cdot \sin(k \cdot \Delta\theta) + t \cdot \cos(k \cdot \Delta\theta)) dt \quad (2.68)$$

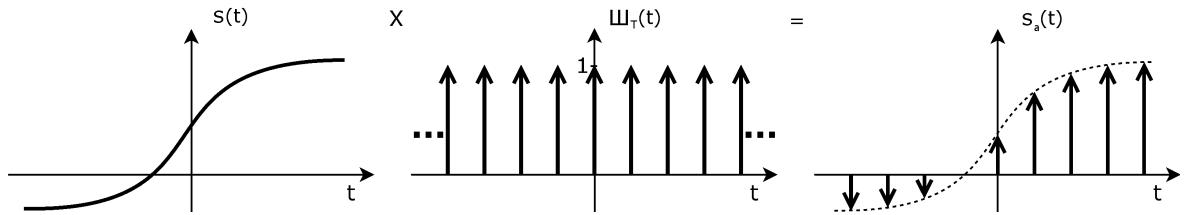
Die Anzahl sowie Auflösung der Projektionswinkel und Linienintegrale korrespondieren in der Computertomographie mit gerätespezifischen Eigenschaften. Genauer ist es das Detektorarray, das jeweils die Anzahl und Auflösung vorgibt. Für jeden Pixel im Detektorarray gibt es eine Ebenennormale, die einer der Geraden  $\ell$  entspricht. Jedem Pixel an einer bestimmten Position lässt sich daher genau ein Linienintegral zuordnen. Da das Lot auf der Ebenennormalen den Ursprung schneidet, ist die Rotation des Detektorarrays um den Ursprung durch die Rotation des Lotes beschreibbar. Bei einer Rotation mit  $\Delta\theta$  bleiben für jeden Pixel die Abstände  $s$  der Normalen zum Ursprung konstant. Lediglich der Winkel  $\theta$  des Lotes wird um  $\Delta\theta$  geändert. Die Anzahl der Werte  $N_s$  in einer Projektion entspricht daher genau der Pixelanzahl im Detektorarray, wenn dieses nur um den Ursprung rotiert und nicht translatiert wird. Die Winkelauflösung ist vorgegeben durch den Rotationswinkel  $\Delta\theta$ . In Abbildung 2.9 ist die Rotation eines Pixels um den Ursprung mit dem Winkel  $\Delta\theta$  dargestellt. Es ist zu sehen, dass die Länge  $s$  des Lotes dabei konstant ist.



**Abbildung 2.9.:** Rotation eines Pixels des Detektorarrays um den Ursprung

### Abtastung

Durch die Abtastung der Projektionen im Ortsraum ergeben sich zu beachtende Konsequenzen im Frequenzraum. Die abgetasteten Projektionen  $p_{A, \theta}(s)$  entsprechen dem Produkt aus den kontinuierlichen Projektionen  $p_\theta(s)$  und einem *Dirac-Kamm*, der aus regelmäßig aufeinanderfolgenden Dirac-Stößen besteht (Gl. 2.69) [2, S. 136].



Quelle: [15]

**Abbildung 2.10.:** Abtastung eines kontinuierlichen Signals mit einem Dirac-Kamm

Die Multiplikation im Ortsraum entspricht dabei einer Faltung im Frequenzraum. Die Fourier-Transformierte eines Dirac-Kamms ist wiederum ein Dirac-Kamm (Gl. 2.70) [2, S. 138]. Damit ist die Fourier-Transformierte  $P_{A, \theta}(\omega)$  der abgetasteten Projektionen  $p_{A, \theta}(s)$  durch Gleichung 2.71 gegeben.

$$p_{A, \theta}(s) = p_\theta(s) \cdot \sum_{l=-\infty}^{\infty} \delta(s - l \cdot \Delta s) \quad (2.69)$$

$$\text{mit: } \mathcal{F} \left\{ \sum_{l=-\infty}^{\infty} \delta(s - l \cdot \Delta s) \right\} = \frac{1}{\Delta s} \cdot \sum_{l=-\infty}^{\infty} \delta \left( \omega - \frac{l}{\Delta s} \right) \quad (2.70)$$

$$\Rightarrow P_{A, \theta}(\omega) = P_\theta(\omega) * \left[ \frac{1}{\Delta s} \cdot \sum_{l=-\infty}^{\infty} \delta \left( \omega - \frac{l}{\Delta s} \right) \right] \quad (2.71)$$

Die Faltung einer Funktion mit einem einzelnen Dirac-Stoß ist identisch zur ursprünglichen Funktion [2, S. 110]. Ist der Dirac-Stoß verschoben, so ist auch die Faltung um denselben Betrag verschoben  $f(x) * \delta(x - x_0) = f(x - x_0)$  [16, S. 40]. In Gleichung 2.71 wird durch die Summe zuerst der Dirac-Kamm gebildet und anschließend gefaltet. Die Reihenfolge lässt sich tauschen, sodass  $P_{A, \theta}(\omega)$  erst mit je einem Dirac-Stoß gefaltet und danach die Summe

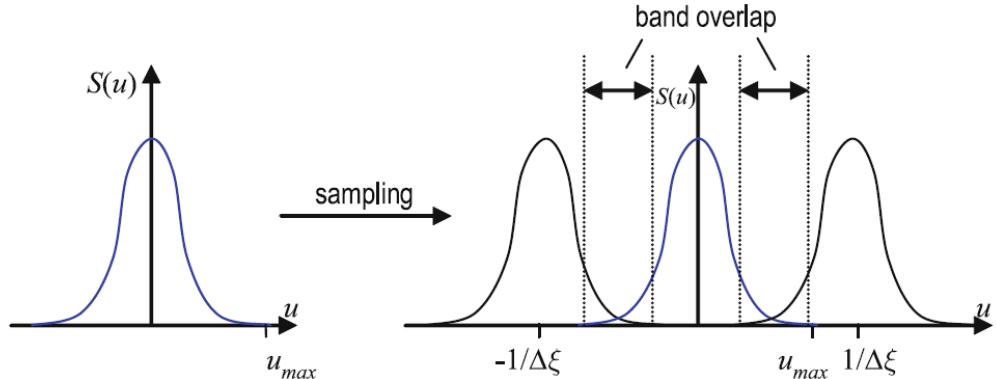
aller Faltungen gebildet wird. Da jeder der Dirac-Stöße um  $l/\Delta s$  verschoben ist, ergibt sich die Summe der Faltungen als Summe der jeweils verschobenen  $P_\theta(\omega)$ . In Gleichung 2.72 ist dieser Zusammenhang abgebildet.

$$P_{A,\theta}(\omega) = \frac{1}{\Delta s} \cdot \sum_{l=-\infty}^{\infty} P_\theta \left( \omega - \frac{l}{\Delta s} \right) \quad (2.72)$$

Damit ist die Fourier-Transformierte einer abgetasteten Projektion  $P_{A,\theta}(\omega)$  jeweils in einem Intervall von  $1/\Delta s$  periodisch [2, S. 139], da dieses dem Abstand der Dirac-Stöße entspricht. Betrachtet man nur die drei Dirac-Stöße mit  $l = -1, 0, 1$ , lässt sich  $P_{A,\theta}(\omega)$  als Summe der drei verschobenen  $P_\theta(\omega)$  darstellen.

$$P_{A,\theta}(\omega) = \frac{1}{\Delta s} \cdot \left[ P_\theta \left( \omega + \frac{1}{\Delta s} \right) + P_\theta(\omega) + P_\theta \left( \omega - \frac{1}{\Delta s} \right) \right] \quad (2.73)$$

In Abbildung 2.11 ist zu erkennen, dass es zur Überlappung kommt, wenn  $P_\theta(\omega)$  zu großen Frequenzen bei zu geringem Abstand der Dirac-Stöße  $l/\Delta s$  enthält.



Quelle: [2, S. 139]

**Abbildung 2.11.:** Überlappung der Fourier-transformierte bei Abtastung im Ortsraum.  $u = \omega$ ,  $S = P_\theta$ ,  $\Delta \xi = \Delta s$

Eine Überlappung der zueinander verschobenen Fourier-transformierte Projektionen wird vermieden, wenn für zwei benachbarte Fourier-transformierte Projektionen  $\omega_{\max} < l/\Delta s - \omega_{\max}$  gilt. Die Fourier-transformierten Projektionen dürfen für Frequenzen größer als  $\omega_{\max}$  keine Frequenzanteile besitzen  $P_\theta(\omega \leq \omega_{\max} \vee \omega \geq \omega_{\max}) = 0$  [2, S. 135]. Im Grenzfall, dass  $\omega_{\max}$  genau zwischen zwei der F.-T. Projektionen liegt, kommt es gerade nicht zur Überlappung. In Gleichung 2.74 ist das aus diesen Überlegungen resultierende *Nyquist-Kriterium* formuliert. Dieses besagt, dass eine korrekte Rekonstruktion eines Signales im Ortsbereich nur möglich ist, wenn die Abtastung mit  $\Delta s$  die Gleichung erfüllt [4, S. 106]. Gleichung 2.75 ist gültig, wenn das Nyquist-Kriterium erfüllt ist.

$$\omega_{\max} < \frac{1}{\Delta s} - \omega_{\max} \Leftrightarrow \boxed{\omega_{\max} < \frac{1}{2 \cdot \Delta s}} \quad (2.74)$$

$$P_{A,\theta}(-\omega_{\max} < \omega < \omega_{\max}) = \frac{1}{\Delta s} \cdot P_\theta(\omega) \quad (2.75)$$

## Filterung

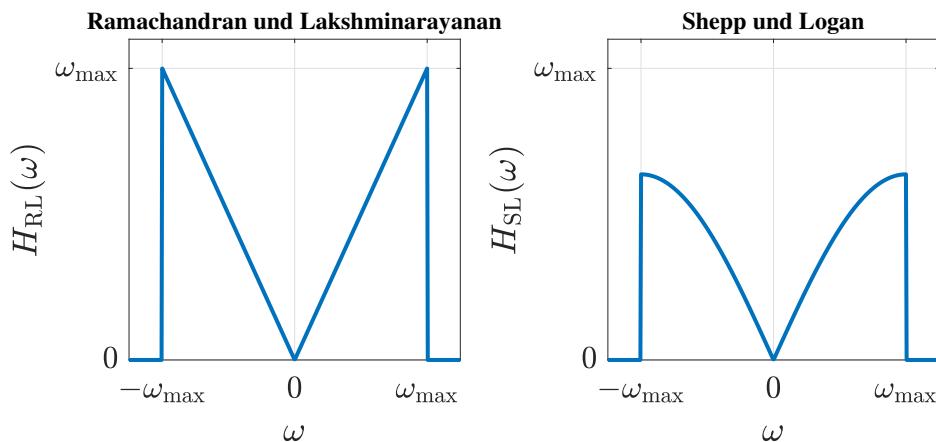
In Abschnitt 2.2.2 auf Seite 20 wurde gezeigt, dass vor der Rückprojektion die Fourier-Transformierten  $P_\theta(\omega)$  der Projektionen im Frequenzraum mit  $|\omega|$  multipliziert werden müssen. Diese Filteroperation kann mit einer praktischen Funktion  $H(\omega)$  durchgeführt werden, wenn sie der mathematisch korrekten Funktion  $|\omega|$  ähnlich ist [4, S. 153]. Weiterhin kann die Filterfunktion durch eine Bandbegrenzung das Nyquist-Kriterium erzwingen. Die einfachste Möglichkeit das Frequenzband zu begrenzen, ist die Fensterung der Filterfunktion mit der Rechteckfunktion. Diese setzt die Funktion oberhalb von  $\omega_{\max}$  auf null [4, S. 154]. Die gefensterte Betragsfunktion  $H_{RL}(\omega)$  in Gleichung 2.77 wird als Filterfunktion nach *Ramachandran und Lakshminarayanan* bezeichnet. Ein alternativer Filter nach *Shepp und Logan*, der mit einem zusätzlichen Faktor die höheren Frequenzen dämpft, ist in Gleichung 2.78 beschrieben.

$$\text{rect}(\omega) = \begin{cases} 1 & , \text{ wenn } |\omega| \leq 1 \\ 0 & , \text{ wenn } |\omega| > 1 \end{cases} \quad (2.76)$$

$$H_{RL}(\omega) = |\omega| \cdot \text{rect}\left(\frac{\omega}{\omega_{\max}}\right) \quad (2.77)$$

$$H_{SL}(\omega) = |\omega| \cdot \frac{\sin(\pi \cdot \omega \cdot (2 \cdot \omega_{\max})^{-1})}{\pi \cdot \omega \cdot (2 \cdot \omega_{\max})^{-1}} \cdot \text{rect}\left(\frac{\omega}{\omega_{\max}}\right) \quad (2.78)$$

In Abbildung 2.12 sind die Filter  $H_{RL}(\omega)$  und  $H_{SL}(\omega)$  dargestellt. In der Praxis bedeutet die Dämpfung im Filter nach Shepp und Logan, dass die räumliche Auflösung sinkt, das Rauschen jedoch verringert wird [4, S. 155]. Dieser Filter gilt unter vielen möglichen Filtern [2, S. 250–251] als Standard, da dieser dem idealen Filter nach Ramachandran und Lakshminarayanan ähnlich ist und dabei Artefakte durch Rauschen verringert [2, S. 252].



**Abbildung 2.12.:** Filterfunktionen nach Ramachandran und Lakshminarayanan sowie Shepp und Logan

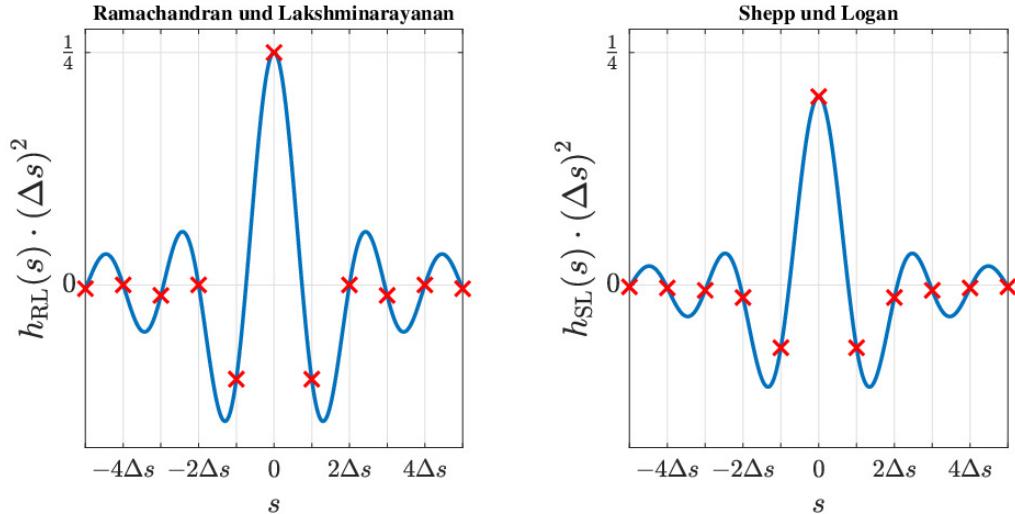
Wie durch Gleichung 2.61 auf Seite 21 beschrieben, lässt sich die Multiplikation mit der Filterfunktion im Frequenzraum als eine Faltung im Ortsraum beschreiben. Die invers Fourier-Transformierte der Filterfunktion  $H(\omega)$  wird als *Faltungskern*  $h(s)$  [4, S. 155] bezeichnet. Die kontinuierlichen Filterkerne  $h_{RL}(s)$  und  $h_{SL}(s)$  sind gegeben durch Gleichung 2.79 und

2.80 gegeben [4, S. 154].

$$h_{RL}(s) = \mathcal{F}^{-1}\{H_{RL}(\omega)\} = \frac{1}{2 \cdot \Delta s^2} \cdot \left[ \frac{\sin(\pi \cdot s \cdot \Delta s^{-1})}{\pi \cdot s \cdot \Delta s^{-1}} + \frac{\cos(\pi \cdot s \cdot \Delta s^{-1}) - 1}{(\pi \cdot s \cdot \Delta s^{-1})^2} \right] \quad (2.79)$$

$$h_{SL}(s) = \mathcal{F}^{-1}\{H_{SL}(\omega)\} = -\frac{2}{\pi^2 \cdot \Delta s^2} \cdot \frac{1 - 2 \cdot s \cdot \Delta s^{-1} \cdot \sin(\pi \cdot s \cdot \Delta s^{-1})}{4 \cdot (s \cdot \Delta s^{-1})^2 - 1} \quad (2.80)$$

In Abbildung 2.13 sind die Faltungskerne zu sehen. Die  $x$ -Achse ist in Vielfache der Ortsauflösung  $\Delta s$  aufgeteilt, da jedes  $s$  in den Faltungskernen auf diese normiert ist. Es sind ebenso bei den ganzzahligen Vielfachen der Ortsauflösung die Werte der Faltungskerne markiert, da nur diese bei der diskreten Faltung notwendig sind.



**Abbildung 2.13.:** Faltungskerne im Ortsraum der Filter nach Ramachandran und Lakshminarayanan sowie Shepp und Logan

Da es sich bei allen Signalen um abgetastete Signale bezüglich des Ortes handelt, muss die kontinuierliche Faltung in eine diskrete Faltung überführt werden. Die Projektionen  $p_\theta(s)$  sind nur an den Stellen  $s = l \cdot \Delta s$  bekannt. Daher kann ein Wert der gefilterten Projektion  $\tilde{p}_\theta(l \cdot \Delta s)$  durch Gleichung 2.81 angegeben werden.

$$\begin{aligned} \tilde{p}_\theta(s) &= p_\theta(s) * h(s) \\ \tilde{p}_\theta(l \cdot \Delta s) &= p_\theta(l \cdot \Delta s) * h(l \cdot \Delta s) = \int_{-\infty}^{\infty} p_\theta(l \cdot \Delta s - s') \cdot h(s') ds' \end{aligned} \quad (2.81)$$

Das Integral in Gleichung 2.81 muss approximiert werden, da die Projektionen nur bei Vielfachen von  $\Delta s$  bekannt sind. Dadurch geht das Integral in eine Summe mit  $s' = m \cdot \Delta s$  über.

$$\begin{aligned} \tilde{p}_\theta(l \cdot \Delta s) &\approx \sum_{m=-\infty}^{\infty} p_\theta((l-m) \cdot \Delta s) \cdot h(m \cdot \Delta s) \cdot \Delta s \\ \text{mit: } p_\theta((l-m) \cdot \Delta s) &= 0 \text{ für } l-m \notin \left[ \left( -\frac{N_s}{2} + 0,5 \right), \left( \frac{N_s}{2} - 0,5 \right) \right] \end{aligned} \quad (2.82)$$

Gleichung 2.82 drückt die Annahme aus, dass für alle  $s = (l-m) \cdot \Delta s$  außerhalb des Messfeldes

die Projektionen gleich null sind. Daher können die Grenzen der Summe angepasst werden.

$$\tilde{p}_\theta(l \cdot \Delta s) \approx \sum_{m=l-\frac{N_s}{2}}^{l+\frac{N_s}{2}} p_\theta((l-m) \cdot \Delta s) \cdot h(m \cdot \Delta s) \cdot \Delta s \quad (2.83)$$

mit:  $l = \left(-\frac{N_s}{2} + 0,5\right), \left(-\frac{N_s}{2} + 1,5\right), \dots, \left(\frac{N_s}{2} - 0,5\right)$

Aus Gleichung 2.83 geht hervor, dass der Filterkern  $h(s)$  für die diskrete Faltung nur an den Stellen  $s = m \cdot \Delta s$  bekannt sein muss. Da die Argumente  $s$  Filterkerne aus Gleichung 2.79 und 2.80 jeweils auf  $\Delta s$  normiert sind, lassen sich für ganzzahlige Vielfache  $m$  von  $\Delta s$  vereinfachte Gleichungen für  $h(m \cdot \Delta s)$  formulieren [4, S. 154].

$$h_{RL}(m \cdot \Delta s) = \begin{cases} \frac{1}{4 \cdot \Delta s^2} & \text{für } m = 0 \\ 0 & \text{für } m \text{ gerade, } \neq 0 \\ -\frac{1}{\pi^2 \cdot \Delta s^2 \cdot m^2} & \text{für } m \text{ ungerade} \end{cases} \quad (2.84)$$

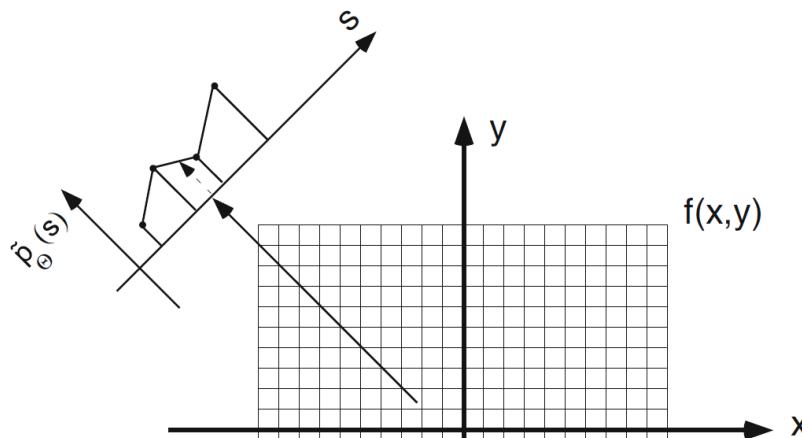
$$h_{SL}(m \cdot \Delta s) = -\frac{2}{\pi^2 \cdot \Delta s^2} \cdot \frac{1}{4 \cdot m^2 - 1} \quad (2.85)$$

### Interpolation bei der Rückprojektion

Mit Gleichung 2.63 ist die Rechenvorschrift zur Berechnung der Funktionswerte durch Integration der gefilterten Projektionen über alle Projektionswinkel gegeben. Das Argument  $s$  der gefilterten Projektionen  $\tilde{p}_\theta(s)$  kann dabei abhängig von den Koordinaten  $(x, y)$  und dem Projektionswinkel  $\theta$  jeden Wert annehmen.

$$f(x, y) = \int_0^\pi \tilde{p}_\theta(s) d\theta \quad \text{mit: } s = x \cdot \cos \theta + y \cdot \sin \theta$$

Wie im vorherigen Abschnitt beschrieben, sind die gefilterten Projektionen nur bei ganzzahligen Vielfachen der Ortsauflösung  $s = l \cdot \Delta s$  bekannt. Daher muss  $s$  wie in Abbildung 2.14 interpoliert werden.



Quelle: [4, S. 153]

**Abbildung 2.14.:** Interpolation bei der Rückprojektion

Als Methode zur Interpolation eignet sich die *lineare Interpolation*. Für ein gegebenes  $s$

werden die beidseitigen bekannten Nachbarn  $s_0 \leq s \leq s_1$  bestimmt. Über Gleichung 2.86 kann an beliebiger Stelle  $s$  der Wert der gefilterten Projektionen berechnet werden. Die Ausnahme bilden Werte für  $s < -s_{\max}$  und  $s > s_{\max}$  für die  $\tilde{p}_{\theta}(s) = 0$  gilt.

$$\begin{aligned} s_0 &= l_0 \cdot \Delta s \text{ und } s_1 = l_1 \cdot \Delta s \\ \tilde{p}_{\theta}(s) &= \tilde{p}_{\theta}(s_0) + \frac{\tilde{p}_{\theta}(s_1) - \tilde{p}_{\theta}(s_0)}{s_1 - s_0} \cdot (s - s_0) \\ &= \tilde{p}_{\theta}(l_0 \cdot \Delta s) + \frac{\tilde{p}_{\theta}(l_1 \cdot \Delta s) - \tilde{p}_{\theta}(l_0 \cdot \Delta s)}{\Delta s} \cdot (s - l_0 \cdot \Delta s) \end{aligned} \quad (2.86)$$

Das Integral über alle Projektionswinkel geht über in eine Summe, da auch die Anzahl der Projektionen endlich ist. Der Funktionswert an der Stelle  $(x, y)$  ergibt sich als Summe der gefilterten Projektionen über alle Projektionswinkel skaliert mit der Winkelauflösung.

$$\begin{aligned} f(x, y) &= \int_0^{\pi} \tilde{p}_{\theta}(x \cdot \cos \theta + y \cdot \sin \theta) d\theta \\ &\approx \Delta\theta \cdot \sum_{k=0}^{N_{\theta}-1} \tilde{p}_{k \cdot \Delta\theta}(x \cdot \cos(k \cdot \Delta\theta) + y \cdot \sin(k \cdot \Delta\theta)) \end{aligned} \quad (2.87)$$

### 2.3. Bestimmung der Schwächungskoeffizienten

In den vorherigen Abschnitten wurde die gesuchte Funktion mit  $f(x, y)$  bezeichnet. Die Größe, die bei der Computertomographie bestimmt werden soll, ist die Schwächung von Röntgenstrahlen, die durch die Schwächungskoeffizienten  $\mu(x, y)$  abgebildet ist. Mit Gleichung 2.39 (S. 17) und 2.41 (S. 18) lässt sich ein Linienintegral als Integral über  $\mu$ , der abhängig von der Distanz  $t$  entlang der Geraden ist, beschreiben.

$$p_{\theta}(s) = \int_{\ell_{\theta, s}} \mu(x, y) dt = \int_{-\infty}^{\infty} \mu(s, t) dt \quad (2.88)$$

Die Grundlage der Rekonstruktion ist die Kenntnis über die Linienintegrale beziehungsweise Projektionen  $p_{\theta}(s)$ . Die einzige messbare Größe ist jedoch die Strahlungsintensität  $J$  nach Austritt aus dem Material. Aus dem Schwächungsgesetz nach Gleichung 2.17 (S. 10) folgt für hintereinanderliegende Materialschichten mit unterschiedlichen Schwächungskoeffizienten Gleichung 2.89.

$$\begin{aligned} J(t) &= J_0 \cdot e^{-\mu \cdot t} \\ \Rightarrow J &= J_0 \cdot e^{-\mu_0 \cdot t_0} \cdot e^{-\mu_1 \cdot t_1} \cdot e^{-\mu_2 \cdot t_2} \dots = J_0 \cdot e^{-\mu_0 \cdot t_0 - \mu_1 \cdot t_1 - \mu_2 \cdot t_2 - \dots} \\ &= J_0 \cdot e^{-\sum \mu_i \cdot t_i} = J_0 \cdot e^{-\int \mu(t) dt} \end{aligned} \quad (2.89)$$

Das Integral im Exponent von Gleichung 2.89 entspricht dem Integral aus Gleichung 2.88. Die am Ausgang mit einem Detektorpixel, dessen Ebenennormale der Geraden  $\ell_{\theta, s}$  entspricht, gemessene Intensität wird mit  $J_{\theta, s}$  bezeichnet. Die Intensität am Eingang in das Material wird  $J_0$  genannt. Es wird angenommen, dass es für  $t < t_{\text{Ein}}$  und  $t > t_{\text{Aus}}$ <sup>1</sup> zu keiner Schwächung

---

<sup>1</sup>Das ist jene Region, die nur aus Luft besteht.

$\mu = 0$  kommt. Für das Linienintegral lässt sich damit Gleichung 2.91 aufstellen.

$$\frac{J_{\theta, s}}{J_0} = e^{- \int \mu(s, t) dt} \quad (2.90)$$

$$\Leftrightarrow \ln \left( \frac{J_0}{J_{\theta, s}} \right) = \int_{t_{\text{Ein}}}^{t_{\text{Aus}}} \mu(s, t) dt = p_{\theta}(s) \quad (2.91)$$

Damit kann durch Messung der Strahlungsintensitäten an den richtigen Stellen und einiger mathematischer Operationen das Schnittbild rekonstruiert werden.

### 2.3.1. Zusammenfassung des Vorgehens

Ziel der Computertomographie ist die Bestimmung des Schwächungskoeffizienten  $\mu$  in einer dünnen Ebene innerhalb eines Materials beziehungsweise Körpers. Dazu sind verschiedene Schritte notwendig.

1. Messen aller Projektionen
2. Projektionen filtern
3. Gefilterte Projektionen interpolieren
4. Rückprojizieren

Zur Bestimmung der Projektionen  $p_{\theta}(s)$  wird die Strahlungsintensität nach Durchgang des Körpers gemessen. Die Projektionen werden jeweils für diskrete Werte von  $s = l \cdot \Delta s$  und  $\theta = k \cdot \Delta \theta$  bestimmt. Die Gesamtzahl der Projektionswinkel ist mit  $N_{\theta} = \pi / \Delta \theta$  und die Anzahl der Linienintegrale pro Projektion mit  $N_s = 2 \cdot s_{\max} / \Delta s + 1$  gegeben.

$$p_{k \cdot \Delta \theta}(l \cdot \Delta s) = \ln \left( \frac{J_0}{J_{k \cdot \Delta \theta, l \cdot \Delta s}} \right) \quad (2.92)$$

$$\text{mit: } l = \left( -\frac{N_s}{2} + 0,5 \right), \left( -\frac{N_s}{2} + 1,5 \right), \dots, \left( \frac{N_s}{2} - 0,5 \right)$$

$$\text{und: } k = 0, 1, 2, \dots, N_{\theta} - 1$$

Die Projektionen werden im Ortsbereich mit einem diskreten Filterkern  $h(l \cdot \Delta s)$  gefaltet. Der Filterkern muss mit  $\omega_{\max} = (2 \cdot \Delta s)^{-1}$  bandbegrenzt sein und seine Fourier-Transformierte ähnlich zu  $H(\omega) = |\omega|$  sein.

$$\tilde{p}_{k \cdot \Delta \theta}(l \cdot \Delta s) = \sum_{m=l-\frac{N_s}{2}}^{l+\frac{N_s}{2}} p_{k \cdot \Delta \theta}((l-m) \cdot \Delta s) \cdot h(m \cdot \Delta s) \cdot \Delta s \quad (2.93)$$

Danach werden die gefilterten Projektionen interpoliert, da bei der Rückprojektion unbekannte Werte, die sich zwischen bekannten Werten befinden, benötigt werden. Für alle  $|l| > N_s / 2$  wird  $\tilde{p}_{k \cdot \Delta \theta}(l \cdot \Delta s) = 0$  angenommen.

$$\tilde{p}_{k \cdot \Delta \theta}(s) = \tilde{p}_{k \cdot \Delta \theta}(l_0 \cdot \Delta s) + \frac{\tilde{p}_{k \cdot \Delta \theta}(l_1 \cdot \Delta s) - \tilde{p}_{k \cdot \Delta \theta}(l_0 \cdot \Delta s)}{\Delta s} \cdot (s - l_0 \cdot \Delta s) \quad (2.94)$$

Durch Rückprojektion aller gefilterten Projektionen werden die gesuchten Schwächungskoeffizienten bestimmt. Das geschieht durch Aufsummierung der gefilterten Projektionen an den

Ebenenkoordinaten über alle Projektionswinkel.

$$\mu(x, y) = \Delta\theta \cdot \sum_{k=0}^{N_\theta-1} \tilde{p}_{k \cdot \Delta\theta}(x \cdot \cos(k \cdot \Delta\theta) + y \cdot \sin(k \cdot \Delta\theta)) \quad (2.95)$$

### 2.3.2. Hounsfield-Einheiten

Bisher wurde die Bestimmung der Schwächungskoeffizienten  $\mu$  als Ziel der Computertomographie dargestellt. In der praktischen Umsetzung sind es jedoch *Hounsfield-Einheiten* (HU), die den Bildpunkten zugeordnet und einem Benutzer angezeigt werden. Bei den Hounsfield-Einheiten handelt es sich um eine Skala, die die Schwächungskoeffizienten auf jene von Wasser und Luft bezieht. Wenn die Schwächung durch Luft mit  $\mu_{\text{Luft}} = 0 \frac{\text{cm}^2}{\text{g}}$  angenommen wird, lassen sich die Hounsfield-Einheiten durch Gleichung 2.96 berechnen [17, S. 119]

$$\text{HU} = \frac{\mu - \mu_{\text{Wasser}}}{\mu_{\text{Wasser}}} \cdot 1000 \text{HU} \quad (2.96)$$

Die Hounsfield-Einheit von Luft ist mit  $\text{HU}_{\text{Luft}} = -1000 \text{HU}$  und die von Wasser mit  $\text{HU}_{\text{Wasser}} = 0 \text{HU}$  gegeben. Das gilt nur, wenn der Computertomograph perfekt kalibriert ist, was selten der Fall ist. Zur Korrektur können durch Messung von  $\text{HU}_{\text{Luft}}$  und  $\text{HU}_{\text{Wasser}}$  im ausgegebenen Bild die idealen Werte  $\text{HU}_{\text{Ideal}}$  aus den gemessenen  $\text{HU}_{\text{Mess}}$  berechnet werden. [17, S. 119]

$$\text{HU}_{\text{Ideal}} = \frac{\text{HU}_{\text{Mess}} - \text{HU}_{\text{Wasser}}}{\text{HU}_{\text{Wasser}} - \text{HU}_{\text{Luft}}} \cdot 1000 \text{HU} \quad (2.97)$$

## 2.4. Modellannahmen in der Computertomographie

In Abschnitt 2.1.1 wurden physikalische Modelle zur Beschreibung von Strahlung und deren Schwächung eingeführt. In den vorangegangenen Abschnitten wurde gezeigt, dass durch die Annahme dieser Modelle eine analytische Formulierung der Computertomographie möglich ist. Da es in der Natur eines Modelles liegt, die Realität vereinfacht abzubilden, entsprechen die Ergebnisse in der Praxis nicht den erwarteten Ergebnissen. Um die Diskrepanz zwischen Modell und Realität beschreiben zu können, müssen die Modellannahmen genauer betrachtet werden.

### 2.4.1. Annahmen über Strahlung

Das Schwächungsgesetz aus Gleichung 2.17 (S. 10) beschreibt die Schwächung von Röntgenstrahlung in Abhängigkeit des Massenschwächungskoeffizienten  $\mu_m$  und der Flächendichte  $d$ . Dieses Schwächungsgesetz gilt nur für sehr dünne Schichten [18, S. 95] oder monoenergetische Strahlung, da der Massenschwächungskoeffizient  $\mu_m$ , wie in Abschnitt 2.1.1 (S. 14) beschrieben, von der Photonenenergie  $E$  abhängt.

Im Photonenmodell wird angenommen, dass sich die Trajektorie eines Photons bei der Transmission nicht verändert oder Sekundärphotonen erzeugt werden. Dass ein Detektorpixel Strahlung, die nicht geradlinig durch den gesamten Körper ging, detektiert, wird im Modell nicht berücksichtigt.

### 2.4.2. Annahmen über Detektion

Es wird angenommen, dass die Position und Ausrichtung jedes Detektorpixels genau bekannt ist, um die Intensität  $J_{\theta, s}$  für den Winkel  $\theta$  und Abstand  $s$  zu messen. Außerdem liegt im Modell die Rotationsebene exakt in der Projektionsebene [18, S. 94]. Des Weiteren wird bezüglich der Pixel angenommen, dass deren Größe genau der Ortsauflösung entspricht und die Pixel direkt aneinander liegen.

Bei der Aufnahme aller Projektionen ist eine vollständige und äquidistante Abtastung angenommen [18, S. 95]. Das bedeutet, dass die Projektionen für alle  $s = l \cdot \Delta s$  und  $\theta = k \cdot \Delta \theta$  bekannt sind. Was die Abtastung im Ort für Konsequenzen hat, wurde bereits in Abschnitt 2.2.3 (S. 22) erläutert.

Bezüglich der Intensitätsmessung durch einzelne Pixel wird angenommen, dass die Photonenanzahl keinem Rauschen unterliegt. Außerdem gilt die Annahme, dass die Strahlungsintensitäten von den Pixeln und der Signalverarbeitungskette kontinuierlich aufgelöst werden können. [18, S. 96]

## 2.5. Artefakte in der Computertomographie

Bei der Implementierung des Simulators werden die Modellannahmen entweder implizit berücksichtigt oder explizit verworfen, um die Realität genauer abzubilden. Die Modellannahmen bezüglich der Intensitätsmessungen und Detektorpixel werden im Simulator implementiert. Genauer handelt es sich um die exakte Kenntnis über die Pixelposition sowie -ausrichtung, die Rotation der Pixel in der Projektionsebene, das zwischenraumfreie Berühren der Pixel, die vollständige Abtastung im Raster und die rauschfreie sowie unendlich hoch aufgelöste Messung der Intensitäten.

Die Annahmen über Sekundärstrahlung und energieunabhängige Schwächung werden im Simulator explizit durch realitätsnähere Modelle ersetzt. Dadurch werden Artefakte durch Streustrahlung und Strahlaufhärtung simuliert. Ebenso werden Teilvolumenartefakte, die durch die diskrete Abtastung der Projektionen und scharfe Objektkanten entstehen, berücksichtigt. Metallartefakte, die durch eine jeweils sehr starke Ausprägung der anderen Artefakte entstehen, werden daher implizit im Simulator abgebildet.

### 2.5.1. Teilvolumenartefakte

Durchläuft Strahlung zwei nebeneinanderliegende Regionen mit den Schwächungskoeffizienten  $\mu_1$  und  $\mu_2$  und trifft auf einen Detektorpixel, würde erwartet, dass die gemessene Intensität einer Schwächung dem mittleren Schwächungskoeffizienten  $\bar{\mu}$  entspricht.

$$\text{erwartet: } J = J_0 \cdot e^{-\bar{\mu} \cdot \Delta x} \quad (2.98)$$

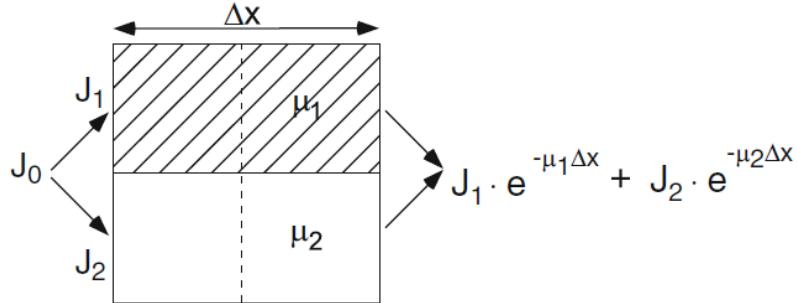
$$p = -\ln \left( \frac{J}{J_0} \right) = \bar{\mu} \cdot \Delta x \quad (2.99)$$

In Abbildung 2.15 jedoch ist zu sehen, dass sich die gemessene Intensität aus der Summe der einzelnen Intensitäten zusammensetzt. Sind die Ursprungsintensitäten  $J_1 = J_2$  identisch, ist

die gemessene Intensität gegeben durch Gleichung 2.101.

$$\text{tatsächlich: } J = \frac{J_0}{2} \cdot e^{-\mu_1 \cdot \Delta x} + \frac{J_0}{2} \cdot e^{-\mu_2 \cdot \Delta x} \quad (2.100)$$

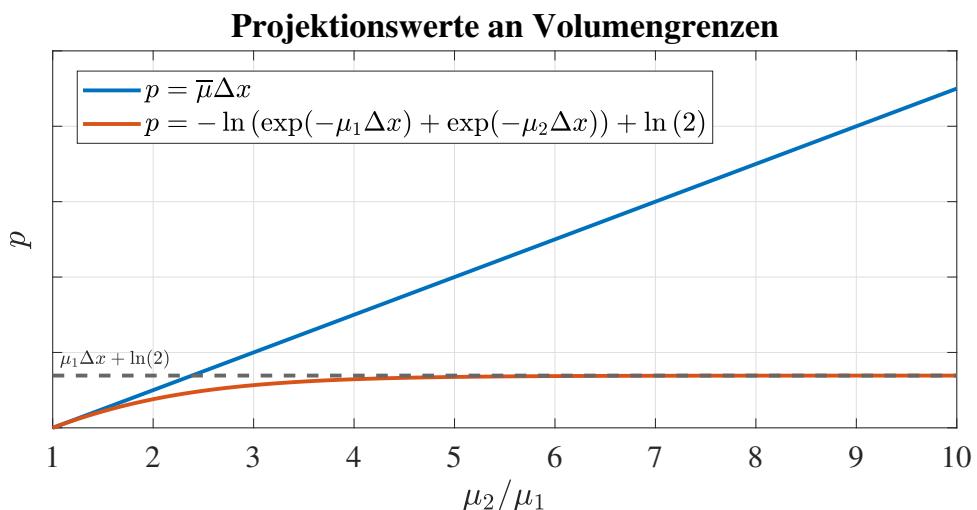
$$p = -\ln\left(\frac{J}{J_0}\right) = -\ln\left(e^{-\mu_1 \cdot \Delta x} + e^{-\mu_2 \cdot \Delta x}\right) + \ln(2) \neq \bar{\mu} \cdot \Delta x \quad (2.101)$$



Quelle: [4, S. 166]

**Abbildung 2.15.:** Entstehung von Teilvolumenartefakten innerhalb eines Schnittbildes

Ist die Schwächung durch eines der Teilvolumen im Vergleich zum zweiten Teilvolumen sehr groß  $\mu_2 \gg \mu_1$ , dominiert in Gleichung 2.101 der erste Summand, sodass sich die Projektion mit  $p = \mu_1 \cdot \Delta x + \ln(2)$  ergibt. In Abbildung 2.16 ist das ebenfalls erkennbar, da für große Quotienten  $\mu_2/\mu_1$  die Projektion konstant wird. Die gemessene Schwächung für diese Projektion wird also unterschätzt.

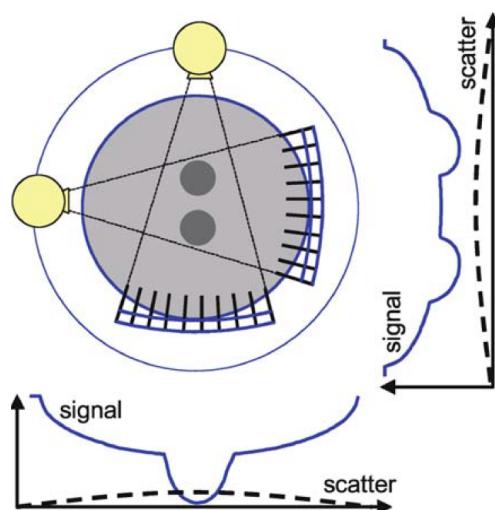


**Abbildung 2.16.:** Erwartete und reale Projektionswerte bei Teilvolumen

Im rekonstruierten Bild äußern sich Teilvolumenartefakte als Streifen, die entlang der Fortsetzung besonders gerader Grenzen zweier Volumen verlaufen. Das ist darin begründet, dass bei der Rückprojektion der falsche Wert des einen Projektionswertes entlang dieser Grenze nicht durch die Projektionen anderer Richtungen korrigiert werden kann. [2, S. 424–425]

### 2.5.2. Streustrahlungsartefakte

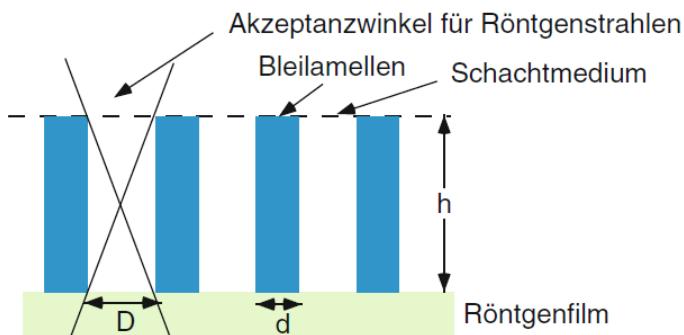
Gestreute Photonen sorgen dafür, dass Detektorpixel Strahlung, die nicht geradlinig durch den Körper hindurchgeht, detektieren. Die Intensitäten werden größer gemessen als bei einer geradlinigen Transmission eines Photonenzündels. Das resultiert in einer Anhebung der gemessenen Intensitäten über alle Pixel. Artefakte treten auf, wenn die Streustrahlungsintensität größer ist als jene der Primärstrahlung. In diesem Fall würde die Absorption bei der Messung unterschätzt. In Abbildung 2.17 sind zwei Projektionen dargestellt. Die zwei hintereinanderliegenden Absorber sorgen für eine geringe Primärstrahlungsintensität, sodass die Streustrahlungsintensität diese übersteigt. Im rekonstruierten Bild äußern sich solche Gegebenheiten als Streifenartefakte [2, S. 443].



Quelle: [2, S. 444]

**Abbildung 2.17.:** Entstehung von Streustrahlungsartefakten

Um den Einfluss der Sekundärstrahlung durch Streustrahlung auf die Messungen zu verringern, kann ein Streustrahlenraster genutzt werden. Dieses besteht aus Bleilamellen, die Sekundärstrahlung mit einem Auftreffwinkel Winkel größer als der Akzeptanzwinkel absorbieren. Ein solches Raster ist schematisch in Abbildung 2.18 abgebildet. Der Akzeptanzwinkel  $\varphi$



Quelle: [4, S. 54]

**Abbildung 2.18.:** Bleilamellen als Streustrahlenraster

ergibt sich aus der Geometrie der Lamellen. Genauer sind die Lamellenhöhe  $h$  und der Lamellenabstand  $D$  entscheidend. Der Akzeptanzwinkel ist der Winkel zwischen dem eintreffenden

Strahl und der Pixelnormalen beziehungsweise dem Detektor.

$$\varphi = \arctan \frac{D}{h} \quad (2.102)$$

### 2.5.3. Artefakte durch Strahlaufhärtung

Die Energieabhängigkeit der Röntgenschwächung hat zur Folge, dass die gemessene Intensität nicht nur von den Schwächungskoeffizienten entlang des geraden Weges abhängt. Die Schwächung und damit die gemessene Intensität hängen auch von der spektralen Zusammensetzung der Strahlung ab. Die gemessene Intensität ergibt sich nach Gleichung 2.103 als Integral der einzelnen Intensitäten über alle Energien im Spektrum [2, S. 425]. Mit diesem Ausdruck für die gemessenen Intensitäten, lässt sich mit Gleichung 2.91 (S. 29) für die gemessenen Projektionen Gleichung 2.104 aufstellen.

$$J = \int_0^{E_{\max}} J_0(E) \cdot e^{-\int \mu_m(t, E) dt} dE \quad (2.103)$$

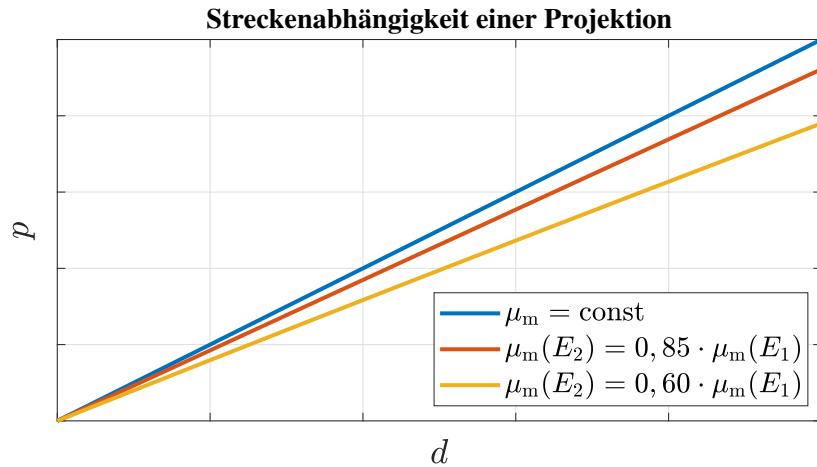
$$p_{\text{real}, \theta}(s) = -\ln \left( \frac{1}{J_0} \cdot \int_0^{E_{\max}} J_0(E) \cdot e^{-\int \mu(s, t, E) dt} dE \right) \quad (2.104)$$

In Gleichung 2.106 ist für den Fall einer Strahlung mit zwei Energien die Berechnung einer Projektion gegeben. Für gleiche Anfangsintensitäten  $J_0(E_1) = J_0(E_2)$  gilt Gleichung 2.107. Letztere ist in Abbildung 2.19 für zwei Verhältnisse der energieabhängigen Schwächungskoeffizienten dargestellt. Es ist klar zu erkennen, dass mit weiterem Weg durch den Körper der Unterschied zwischen der idealen und tatsächlichen Projektion größer wird.

$$\text{ideal: } p_i(d) = -\ln \left( \frac{J}{J_0} \right) = \mu_m \cdot d \quad (2.105)$$

$$\text{real: } p_r(d) = -\ln \left( \frac{1}{J_0(E_1) + J_0(E_2)} \cdot \left[ J_0(E_1) \cdot e^{-\mu_m(E_1) \cdot d} + J_0(E_2) \cdot e^{-\mu_m(E_2) \cdot d} \right] \right) \quad (2.106)$$

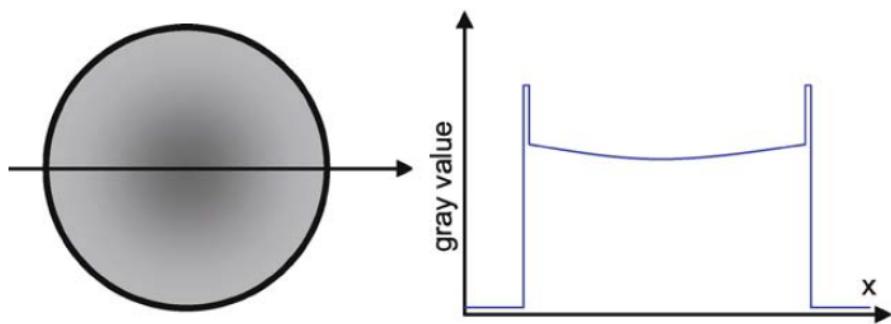
$$= -\ln \left( \frac{1}{2} \cdot \left[ e^{-\mu_m(E_1) \cdot d} + e^{-\mu_m(E_2) \cdot d} \right] \right) \quad \text{für: } J_0(E_1) = J_0(E_2) \quad (2.107)$$



**Abbildung 2.19.:** Ideale und energieabhängige Röntgenschwächung im Vergleich

Die Abhängigkeit der gemessenen Projektionen von der durchstrahlten Weglänge hat einen Effekt, der als *Cupping* bezeichnet wird, zur Folge. Für geringe Weglängen entsprechen die real gemessenen Intensitäten und damit die gemessenen Projektionen dem idealen Modell. Werden die Weglängen größer, steigt auch der Unterschied zwischen dem Modell und den gemessenen Projektionen (Abb. 2.19). Grund für dieses Verhalten ist der sinkende Schwächungskoeffizient mit steigender Energie (Abb. 2.4, S. 15). Dadurch kommt es zu einer stärkeren Verschiebung des Spektrums in den höherenergetischen Bereich, je länger der durchstrahlte Weg ist. Dieser Effekt wird als *Strahlaufhärtung* bezeichnet. Durch die Verschiebung des Spektrums zu höheren Energien und geringere Schwächung bei höherer Energie nimmt die effektive Schwächung mit steigender Strecke ab. Die Schwächung wird für längere Strecken unterschätzt, sodass den Stellen im durchstrahlten Körper mit größerer Entfernung zum Rand geringere Schwächungskoeffizienten zugeordnet werden als tatsächlich vorliegen [14, S. 191].

In Abbildung 2.20 ist das rekonstruierte Bild eines Körpers mit homogenem Schwächungskoeffizienten abgebildet. Im rekonstruierten Bild ist zu erkennen, dass trotz Homogenität die Grauwerte am Rand größer als die Innenliegenden sind.



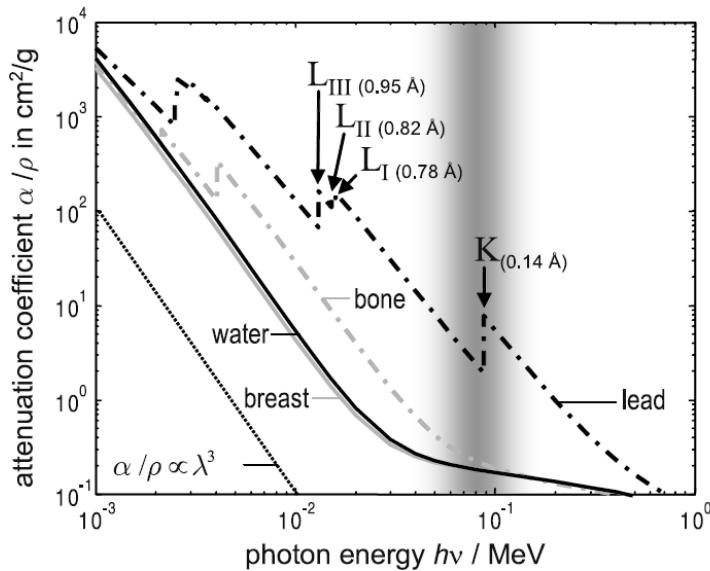
Quelle: [2, S. 429]

**Abbildung 2.20.:** Cupping-Artefakte durch Strahlaufhärtung

### Gegenmaßnahmen

Um die Stärke von Artefakten, die durch die Energieabhängigkeit der Schwächung auftreten, zu verringern, lassen sich verschiedene Maßnahmen ergreifen. Ideal wäre die Nutzung monoenergetischer Strahlung, was wegen der Erzeugung der Strahlung durch Röntgenröhren nicht möglich ist. Eine Verschiebung des Strahlenspektrum zu höheren Energien jedoch ist möglich und würde die Artefaktstärke verringern. Der Grund dafür ist in Abbildung 2.21 zu sehen. Im Bereich von ungefähr 100 keV ist die Abhängigkeit des Schwächungskoeffizienten von der Energie weitaus geringer als bei niedrigeren Energien, da der Gradient niedriger ist. Befände sich die Strahlung primär in diesem Bereich wäre die Artefaktstärke geringer.

Eine Verschiebung des Spektrums hin zu höheren Energien lässt sich durch zwei Maßnahmen erreichen. Eine größere Beschleunigungsspannung in der Röntgenröhre würde, wie in Abschnitt 2.1 beschrieben, für eine höhere maximale Energie im Spektrum sorgen. Der niederenergetische Teil des Spektrums lässt sich verringern, indem die Röntgenstrahlung vor Eintritt in den Körper gefiltert wird. Das wird durch Filter erreicht, die direkt nach der Rönt-



Quelle: [2, S. 444]

**Abbildung 2.21.:** Energieabhängigkeit der Schwächung. Bei höheren Energien ist die Abhängigkeit geringer.

genröhre Strahlung mit Energien unterhalb von ungefähr 60 keV herausfiltern [14, S. 160]. Als Filter kommen Materialien wie zum Beispiel Aluminium, Titan [14, S. 161] oder Kupfer [4, S. 167] in wenigen Millimetern Dicke zum Einsatz. Ein positiver Nebeneffekt der Filterung ist die Dosisreduktion für einen Patienten, da der niederenergetische Teil des Spektrums fast vollständig absorbiert würde und somit nur zur Patientendosis aber nicht zum Bild beitragen würde [14, S. 160].

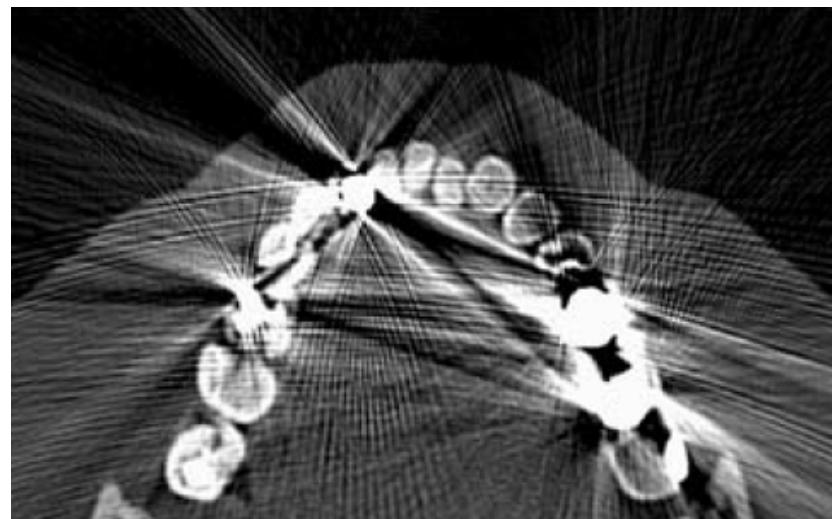
Insbesondere Cupping-Artefakte können durch eine Wasservorkorrektur verringert werden. Eine im klinischen Bereich häufig eingesetzte Korrektur ist eine empirische Kalibration. Diese stellt einen Zusammenhang zwischen der durchstrahlten Weglänge und der gemessenen Projektion her. Durch Kenntnis der Körpergeometrie und des empirischen Zusammenhangs von Schnittlänge und Messwert lassen sich unterschätzte Schwächungskoeffizienten in erster Näherung korrigieren. [14, S. 191]

## 2.5.4. Metallartefakte

Metalle verfügen im Allgemeinen über eine höhere Dichte als biologisches Gewebe. Daher ist grundsätzlich nach Gleichung 2.16 (S. 10) die Schwächung von Röntgenstrahlung größer, wenn die Dichte größer ist. So hat zum Beispiel Eisen mit einer Dichte von  $7,87 \frac{\text{g}}{\text{cm}^3}$  [19, S. 1049] eine zu Wasser achtfache Dichte. Außerdem ist nach Gleichung 2.22 die Schwächung durch den Photoeffekt abhängig von der Ordnungszahl  $\mu_{\text{pho}} \propto Z^3$ , die in Metallen ebenso größer ist als in Gewebe. Sowohl Absorption als auch Streuung sind daher in Metallen deutlich ausgeprägter.

Wegen der stark unterschiedlichen Schwächungskoeffizienten und den häufig geradlinigen Kanten metallener Implantate kommt es zu starken Teilvolumenartefakten [2, S. 438]. Zusätzlich ist die Streuung durch Metalle wegen der höheren Elektronendichte deutlich größer. Dadurch ist insbesondere im Schatten des metallenen Objektes der Sekundärstrahlungsanteil

hoch, was zu Streustrahlungartefakten führt [2, S. 438]. Neben Teilvolumenartefakten und Streustrahlungartefakten tragen Strahlaufhärtungsartefakte wesentlich zu Metallartefakten bei. Wegen der starken Absorption ist auch bei kleinen Objekten die Strahlaufhärtung relevant. Wird Strahlung zu so großen Teilen absorbiert, dass am Detektor keine Strahlung detektiert würde, wäre der Wert der Projektion wegen des Logarithmus unendlich hoch [2, S. 438]. Aus Sicht der Rückprojektion ist daher ununterscheidbar, ob an einer Stelle des durchstrahlten Weges oder an allen Stellen die Schwächung hoch ist. In Abbildung 2.22 sind stark ausgeprägte Metallartefakte durch Zahnimplantate zu sehen. Besonders auffällig sind die radial verlaufenden Streifenartefakte, die von jedem metallenen Objekt ausgehen.



Quelle: [2, S. 439]

**Abbildung 2.22.:** Metallartefakte durch Zahnimplantate im rekonstruierten Bild



# 3. Implementierung

An den Simulator werden bestimmte Anforderungen gestellt. Diese lassen sich in mehrere Kategorien unterteilen. *Allgemeine* Anforderungen beziehen sich auf die grundlegenden Funktionen und den Zweck des Simulators. Des Weiteren sind Anforderungen an die Implementierung der *physikalischen Modelle* bezüglich der Körpermodelle und deren Interaktion mit Röntgenstrahlung gestellt. Weiterhin gibt es Ansprüche an die zu implementierenden *Gerätekomponenten* und *Rekonstruktionsprozesse*. Der geschriebene *Quelltext* und die *Benutzeroberfläche* sollen ebenfalls bestimmten Kriterien entsprechen.

## Allgemeine Anforderungen

**ALG-000** *Musskriterium:* Es sollen einzelne Schnittbilder aus dreidimensionalen Körpermodellen, die Informationen über die Röntgenschwächungseigenschaften haben, erzeugt werden. Das soll über die Messung und Verarbeitung von Projektionen geschehen.

**ALG-001** *Musskriterium:* Die Röntgenstrahlung und deren Interaktion mit Materie soll nahe an den physikalischen Modellen liegen.

**ALG-002** *Musskriterium:* Der Simulator soll geeignet sein, die Funktionsweise eines Computertomographen und besondere Eigenschaften sowie Artefakte zu Lehrzwecken darzustellen.

**ALG-003** *Abgrenzungskriterium:* Der Simulator ist nicht zur klinischen Anwendung gedacht.

**ALG-004** *Wunschkriterium:* Zur Verbesserung der Wartbarkeit sollen alle Laufzeitfehler protokolliert und dem Benutzer angezeigt werden.

**ALG-005** *Wunschkriterium:* Parallele Berechnungen zeitintensiver Algorithmen.

## Anforderungen an die physikalischen Modelle

**PHY-010** *Musskriterium:* Das dreidimensionale Körpermodell soll sich aus quaderförmigen Volumenelementen (Voxeln) zusammensetzen.

**PHY-011** *Musskriterium:* Jedes Voxel soll Informationen über die Röntgenschwächung in Form eines Schwächungskoeffizienten tragen.

**PHY-012** *Musskriterium:* Jedes Voxel soll über spezielle Eigenschaften verfügen können.

**PHY-013** *Musskriterium:* Eine der speziellen Eigenschaften soll Metall darstellen.

**PHY-014** *Musskriterium:* Körpermodelle sollen aus geometrischen Körpern wie Sphären und Würfel erstellt werden können.

**PHY-015** *Wunschkriterium:* Körpermodelle sollen aus DICOM-Datensätzen erstellt werden können.

**PHY-020** *Musskriterium:* Röntgenstrahlung soll als Menge von Nadelstrahlen, die durch das Körpermodell verfolgt werden, simuliert werden.

**PHY-021** *Musskriterium:* Die Interaktion von Röntgenstrahlung mit Materie soll innerhalb des Körpermodells für jedes Voxel, das von einem Nadelstrahl getroffen wird, einzeln berechnet werden.

**PHY-022** *Musskriterium:* Strahlaufhärtungsartefakte sollen simuliert werden.

**PHY-023** *Musskriterium:* Streustrahlungsartefakte sollen simuliert werden.

**PHY-024** *Musskriterium:* Teilvolumenartefakte sollen simuliert werden.

**PHY-025** *Wunschkriterium:* Strahlaufhärtungs- und Streustrahlungsartefakte sollen auschaltbar sein.

**PHY-026** *Wunschkriterium:* Die Streustrahlungsartefakte sollen in ihrer Stärke einstellbar sein.

## Anforderungen an die Gerätekomponenten

**DEV-000** *Musskriterium:* Der Simulator soll, entsprechend eines Computertomographen der dritten Generation durch eine Röntgenröhre und einen Detektor, die Strahlungserzeugung sowie -detektion simulieren.

**DEV-010** *Musskriterium:* Die Strahlungseigenschaften sollen von der Röhrenspannung und dem Röhrenstrom abhängen.

**DEV-011** *Musskriterium:* Es sollen optionale Vorfilter, die zur Verringerung von Strahlaufhärtungsartefakten führen, implementiert werden.

**DEV-020** *Musskriterium:* Das Detektorarray soll aus einzelnen Pixeln konstruiert werden.

**DEV-021** *Musskriterium:* Die Geometrie des Detektorarrays soll einstellbar sein. Die Anordnung der einzelnen Pixel soll dabei entsprechend vorgegebenen Eigenschaften berechnet werden. Zu diesen Eigenschaften zählen:

- Anzahl sowie Auflösung der Projektionswinkel und Projektionsabstände
- Messfeld
- Detektor-Röhren-Abstand

**DEV-022** *Musskriterium:* Die Pixelanordnung muss so sein, dass bei einer Rotation der Detektorarrays eine äquidistante Abtastung der Winkel und Abstände gegeben ist.

**DEV-023** *Musskriterium:* Es soll ein optionales Streustrahlungsraster mit einstellbarem Akzeptanzwinkel implementiert werden.

**DEV-024** *Wunschkriterium:* Es sollen Pixelrauschen und Pixelempfindlichkeit im Simulator abgebildet sein.

## Anforderungen an die Rekonstruktionsprozesse

**SIG-000** *Musskriterium:* Die gefilterte Rückprojektion soll als Rekonstruktionsmethode genutzt werden.

**SIG-001** *Musskriterium:* Der Filter für die gefilterte Rückprojektion soll änderbar sein.

**SIG-002** *Musskriterium:* Das rekonstruierte Bild soll kalibrierte Hounsfield-Einheiten darstellen.

### Anforderungen an den Quelltext

**COD-000** *Musskriterium:* Der Quelltext soll einfach erweiterbar und modular sein.

**COD-001** *Musskriterium:* Der Quelltext muss gut dokumentiert sein.

**COD-002** *Musskriterium:* Der Quelltext soll verständlich sein.

### Anforderungen an die Benutzeroberfläche

**GUI-000** *Musskriterium:* Die Benutzeroberfläche soll grafisch sein. Die Eingabe soll über die Eingabegeräte Maus und Tastatur erfolgen.

**GUI-001** *Musskriterium:* Alle Eingaben sollen auf Plausibilität geprüft werden.

**GUI-002** *Musskriterium:* Dem Benutzer sollen zu Oberflächenelementen Informationen über Hilfstexte gegeben werden.

**GUI-003** *Wunschkriterium:* Der Programmstatus soll automatisch gespeichert und wiederhergestellt werden

**GUI-010** *Musskriterium:* Körpermodelle sollen geladen, gespeichert und erstellt werden können.

**GUI-011** *Musskriterium:* Es soll das ideale Schnittbild aus den reinen Modelldaten angezeigt werden.

**GUI-012** *Musskriterium:* Körpermodelle sollen bewegt werden können.

**GUI-013** *Wunschkriterium:* Die Stärke der Artefakte, die durch Voxel mit der Eigenschaft Metall verursacht werden, soll einstellbar sein.

**GUI-020** *Musskriterium:* Die Röhreneigenschaften aus DEV-010 und DEV-011 sollen einstellbar sein.

**GUI-021** *Musskriterium:* Die Projektionseigenschaften aus DEV-021 sollen eingegeben werden können.

**GUI-022** *Musskriterium:* Die Strahlungseigenschaften sollen angezeigt werden.

**GUI-023** *Wunschkriterium:* Die Pixelanordnung soll angezeigt werden.

**GUI-030** *Wunschkriterium:* Die Streustrahlungsartefakte sollen wie aus PHY-025 und PHY-026 einstellbar sein.

**GUI-031** *Wunschkriterium:* Die Strahlaufhärtungsartefakte sollen wie aus PHY-025 ausschaltbar sein.

**GUI-040** *Musskriterium:* Projektionen, die zu einem Schnittbild gehören, sollen gespeichert und geladen werden können.

**GUI-041** *Musskriterium:* Die Projektionen sollen als Sinogramm angezeigt werden.

**GUI-042** *Musskriterium:* Die gefilterten Projektionen sollen angezeigt werden.

**GUI-043** *Musskriterium:* Das rekonstruierte Bild soll angezeigt werden.

**GUI-044 Wunschkriterium:** Der Kontrast des rekonstruierten Bildes soll einstellbar sein.

### 3.1. Konzeptionierung

Der Simulator lässt sich in einzelne abstrakte Module mit jeweils einzelnen Komponenten, die untereinander und mit Komponenten anderer Module interagieren, unterteilen. Die Komponenten ergeben sich aus den gestellten Anforderungen. In Abbildung 3.1 sind die Komponenten und deren Beziehungen zueinander dargestellt. Über die Benutzeroberfläche kann ein Körpermodell erstellt oder geladen werden (GUI-010). Die Oberfläche stellt das Modell dar (GUI-011). Durch den Benutzer erfolgt eine Eingabe der Detektoreigenschaften (DEV-021, GUI-021) und Röhreneigenschaften (GUI-020) sowie der Eigenschaften für die Streuung (GUI-030) und Absorption (GUI-031). Startet der Benutzer die Aufnahme eines Schnittbildes, werden von der Röntgenröhre Nadelstrahlen erzeugt, die durch das Körpermodell verfolgt werden. Jedes getroffene Voxel verändert durch seine Eigenschaften die Strahlungseigenschaften (PHY-021) durch Absorption (PHY-022, PHY-024) und Streuung (PHY-023). Nachdem die Nadelstrahlen das Körpermodell verlassen, werden diese vom Detektor detektiert. Das Durchstrahlen des Körpermodells wird für mehrere Rotationswinkel der Gantry wiederholt. Die Detektionsergebnisse aller Durchstrahlungen werden als Projektionen (Sinogramm) gespeichert und verarbeitet. Durch Filterung entstehen die gefilterten Projektionen (SIG-000), die durch Rückprojektion das rekonstruierte Bild liefern. Dieses und die gefilterten Projektionen werden durch die Benutzeroberfläche dargestellt (GUI-043).

Der Simulator wird in der Programmiersprache C++ implementiert. C++ ermöglicht die Nutzung eines objektorientierten Paradigmas. In der objektorientierten Programmierung können Repräsentationen realer Gegenstände oder Konzepte als *Objekte* dargestellt werden [20, S. 719]. Außerdem wird eine Kombination von Daten und Verhalten dieser Objekte angestrebt. Das erhöht die Wartbarkeit und Erweiterbarkeit (COD-000) des Systems [20, S. 718], da bei Änderungen der Objekteigenschaften oder des Objektverhaltens nur diese angepasst werden müssen. Ein Objekt wird durch seine Beschreibung, die als *Klasse* bezeichnet wird, charakterisiert. Eine Klasse verfügt über Membervariablen und -funktionen, deren Zugriff beschränkt werden kann. Diese Komponenten einer Klasse werden als *Klassenmember* bezeichnet. Klassen können die Daten und das Verhalten anderer Klassen über *Vererbung* kopieren. Ein einzelnes Objekt einer Klasse nennt sich *Instanz*. Neben den Vorteilen des objektorientierten Paradigmas ist C++ eine der zeitlich effizientesten Programmiersprachen [21, S. 8+24] und verfügt über eine gewisse Popularität in der Open-Source Entwicklung [22]. Erstere Eigenschaft ist notwendig, um die Rechenzeit zu verringern. Letztere, um auf Bibliotheken zurückgreifen zu können.

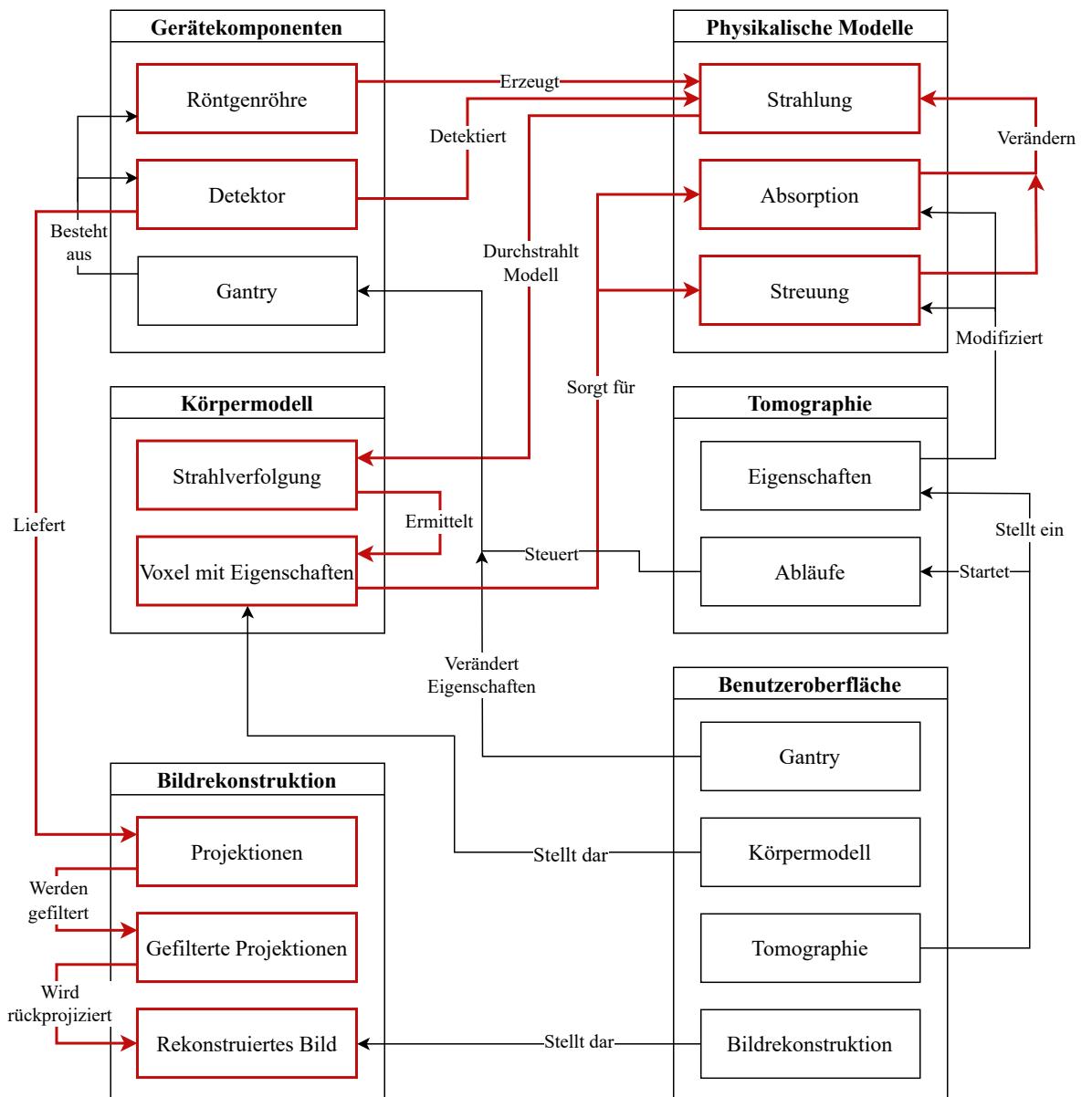
Entwickelt wird auf einer Microsoft Windows-Plattform mit der integrierten Entwicklungsumgebung (IDE) *Visual Studio Community 2022*. Zur Quelltextverwaltung wird ein Repository auf GitHub genutzt, in dem die Änderungen über den Entwicklungszeitraum verfolgt werden. Das Repository ist unter <https://github.com/janwolzenburg/ct-simulator> aufrufbar. Für den Quelltext werden zur Benennung von Klassen, Variablen und

Funktionen Konventionen getroffen (COD-002):

<b>Variablen:</b>	variable_mit_mehreren_worten
<b>Private Membervariablen:</b>	private_membervariable_
<b>Klassen:</b>	EineKlasse
<b>Funktion:</b>	EineFunktion()
<b>Zugriff auf private Membervariable:</b>	private_membervariable()

Die Definition und Implementierung von Klassen ist jeweils in .h und .cpp Dateien aufgeteilt. Der Dateiname entspricht dabei dem Klassennamen mit kleinem Anfangsbuchstaben. Weiterhin werden Klassen und Funktionen im *Doxxygen*-Stil kommentiert, sodass die Schnittstellen dokumentiert sind (COD-001). Der gesamte Quelltext ist in englischer Sprache verfasst. Dazu zählen Bezeichner und Kommentare.

Klassendiagramme, die in den folgenden Abschnitten genutzt werden, geben eine Übersicht über die Beziehung zwischen verschiedenen Klassen. Ein Element im Klassendiagramm ist mit dem Klassennamen betitelt und einer Liste der Klassen, von denen geerbt wird, versehen. Graue Pfeile mit den Stichworten *public*, *protected* und *private* zeigen an, ob von einer anderen Klasse geerbt wird. Grüne Pfeile tragen die Bezeichnung einer Membervariablen und zeigen auf die Klasse dieser Membervariablen. Andere Membervariablen und Memberfunktionen sind in den Diagrammen innerhalb dieses Kapitels ausgeblendet.



**Abbildung 3.1.:** Funktionale Komponenten des Simulators. Die Komponenten sind zu Modulen gegliedert und deren Beziehung zueinander durch Pfeile angezeigt. In Rot sind die Kernelemente des Simulators und deren Beziehung hervorgehoben.

## 3.2. Darstellung geometrischer Objekte

Zur Implementierung des Simulators ist es notwendig, geometrische Objekte definieren zu können und deren räumliche Beziehung zu ermitteln. So müssen Nadelstrahlen (PHY-020), Detektorpixel (DEV-020), Voxel des Körpermodells (PHY-010) sowie Translation und Rotation der Gerätekomponenten (DEV-000) im Simulator abgebildet werden.

Um physikalische Größen<sup>1</sup> quantitativ abbilden zu können, müssen der rationale Zahlenraum  $\mathbb{Q}$  sowie bestimmte mathematische Konstanten wie  $\pi$  oder  $e$  im Computerprogramm repräsentiert werden. Ganze Zahlen werden durch Datentypen wie zum Beispiel `int` (integer) repräsentiert. Rechen- und Vergleichsoperationen verhalten sich bei ganzen Zahlen erwartungsgemäß. Einzig bei Division wird der fraktionelle Anteil abgeschnitten. Fließkommazahlen können für die Repräsentation des kontinuierlichen Zahlenraumes genutzt werden. Bei deren Nutzung können jedoch aufgrund ihrer Definition unerwartete Effekte hervorrufen. Fließkommazahlen mit doppelter Präzision `double` haben eine Größe von 64 bit [23]. Sie repräsentieren Zahlen durch ihr Vorzeichen  $s$ , einen ganzzahligen Exponenten  $E$  und eine rationale Mantisse  $M$  mit den Bits  $M_i$  über Gleichung 3.1 [24, S. 63].

$$\text{Zahl mit Typ } \text{double} = s^{-1} \cdot 2^E \cdot M = s^{-1} \cdot 2^E \cdot \sum_{i=1}^{53} M_i \cdot 2^{-i} \quad (3.1)$$

mit:  $s = 0$  oder  $1$

$$E = -2^{10} + 1, -2^{10} + 2, \dots, 2^{10}$$

$$M_i = 0 \text{ oder } 1$$

Durch diese Art der Repräsentation kann es zum Informationsverlust bei mehreren hintereinander ausgeführten arithmetischen Operationen kommen. Dadurch sind diese Operationen streng betrachtet nicht assoziativ, kommutativ oder distributiv. Außerdem muss ein direkter Vergleich zweier Fließkommazahlen vermieden werden. [24, S. 63]

Statt eines direkten Vergleiches mit dem Vergleichsoperator `==` werden die Funktionen aus Listing A.1 und A.2, die zwei Fließkommazahlen vergleichen, genutzt. Diese ermöglichen den Vergleich mit Angabe einer absoluten oder relativen Toleranz. Für alle Streckenangaben im Quelltext, die immer in der Einheit mm angegeben sind, ist die maximale absolute Abweichung auf  $10 \cdot 10^{-9}$  mm festgelegt. Ist die Differenz zweier Strecken geringer, werden diese als gleich angenommen.

### 3.2.1. Koordinaten

Die Grundlage zur Darstellung von Objekten im dreidimensionalen Raum bilden Punkte oder Vektoren in Koordinatensystemen. Durch sie lassen sich geometrische Angaben über die Beziehungen physikalischer Größen wie Längen, Abstände, Richtungen oder Winkel machen [25, S. 105]. Ein dreidimensionaler Punkt  $P$  und Vektor  $\vec{v}$  lässt sich in einem kartesischen Koordinatensystem durch drei Komponenten beschreiben. Der Punkt  $P$  kann auch durch

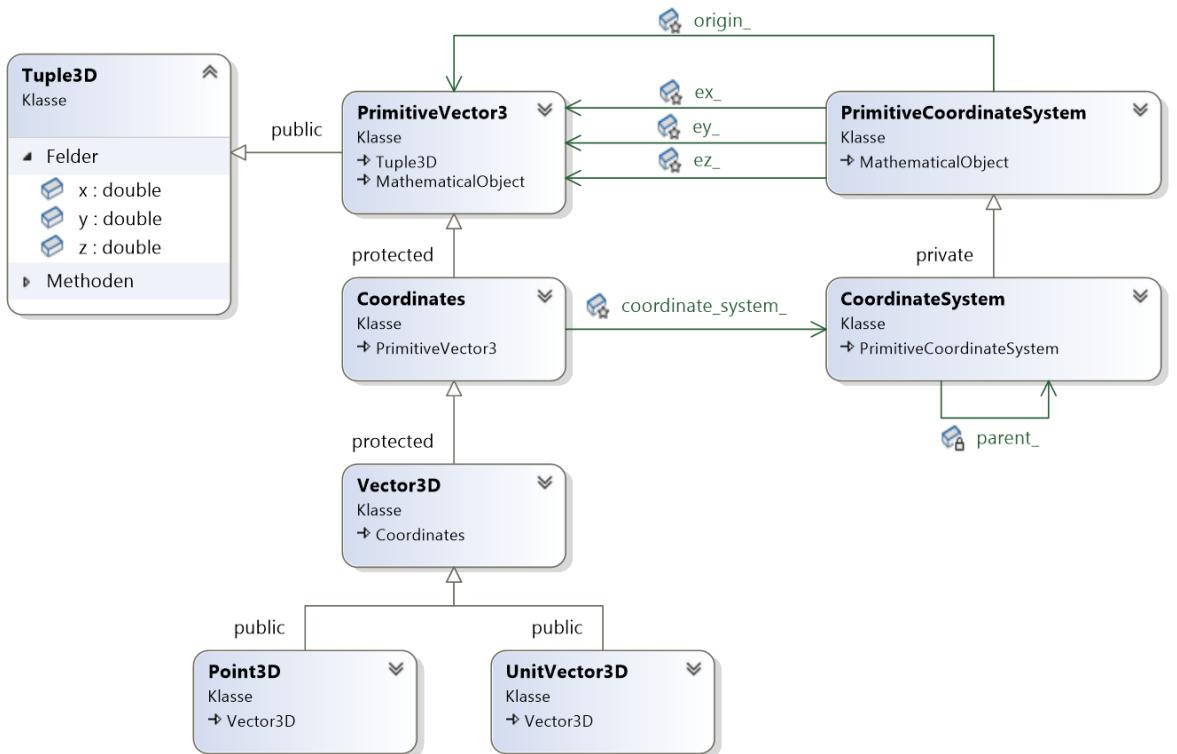
---

<sup>1</sup>Physikalische Größen können zum Beispiel Strecken, Leistungen und Energien sein.

seinen Ortsvektor  $\vec{p}$  mit identischen Komponenten beschrieben werden.

$$\mathbf{P} = (P_x, P_y, P_z) \quad \vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

In Abbildung 3.2 ist das Klassendiagramm zur Implementierung von Vektoren und Punkten in beliebigen Koordinatensystemen dargestellt. Das zugrundeliegende Element in der Implementierung ist die Klasse `Tuple3D`, die drei Zahlen  $x$ ,  $y$  und  $z$  mit dem Datentyp `double` zusammenfasst. In der Klasse `PrimitiveVector3` werden für primitive Vektoren, die nicht im Kontext eines bestimmten Koordinatensystems liegen, einige triviale Operationen implementiert. Darunter fallen Addition, Subtraktion, Multiplikation, Skalierung und Division sowie Vergleichsoperation. Außerdem werden Translation und Rotation implementiert. Die Rotation um eine beliebige Achse, die in Listing A.3 dargestellt ist, stellt dabei die umfangreichste Operation dar.



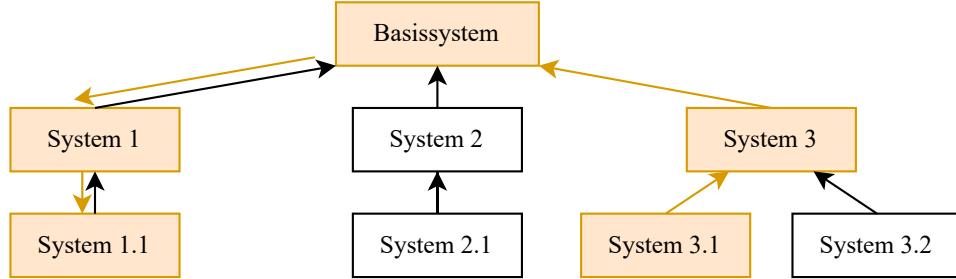
**Abbildung 3.2.:** Klassendiagramm der Klassen, die die Darstellung von geometrischen Objekten ermöglichen

Kartesische Koordinatensysteme sind durch drei senkrecht aufeinander stehende Basisvektoren  $\vec{e}_x$ ,  $\vec{e}_y$ ,  $\vec{e}_z$  und einen Ursprungspunkt O vollständig definiert. Die Basisvektoren sind jeweils Einheitsvektoren  $|\vec{e}_i| = 1$  und bilden die  $x$ -,  $y$ - und  $z$ -Achse des Koordinatensystems. Durch Rotation aller Basisvektoren oder Translation des Ursprunges, lässt sich ein Koordinatensystem beliebig positionieren. Die Klasse `PrimitiveCoordinateSystem` stellt solch ein primitives Koordinatensystem im Kontext des Basissystems dar. Das Basissystem kann als globales System mit gegebenen Basisvektoren und Ursprungspunkt beschrieben

werden.

$$\vec{e}_x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}; \quad \vec{e}_y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}; \quad \vec{e}_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}; \quad O = (0, 0, 0) \quad (3.2)$$

Koordinatensysteme lassen sich ineinander verschachteln, sodass ein System in Bezug auf ein übergeordnetes System angegeben ist. Dadurch lassen sich beliebig komplexe Beziehungen zwischen Koordinaten abbilden. Solch ein Baum aus Koordinatensystem ist in Abbildung 3.3 dargestellt. Implementiert ist diese Struktur durch die Klasse `CoordinateSystem`, von



**Abbildung 3.3:** Verschachtelte Koordinatensystem dargestellt als Baum. In Orange ist der Pfad, der für die Transformation von Koordinaten aus System 3.1 zu Koordinaten aus System 1.1 genommen werden muss, hervorgehoben

welcher jede Instanz über eine Referenz zu dem übergeordneten System verfügt. Koordinaten mit Bezug zu einem bestimmten System sind durch die Klasse `Coordinates` abgebildet. Um Koordinaten aus einem System in ein anderes zu transformieren, muss auf eine bestimmte Art und Weise vorgegangen werden. Zuerst müssen die Koordinaten  $\vec{v}$  vom Startsystem in das Basissystem überführt werden. Dazu werden iterativ für jedes System entlang des Pfades zum Basissystem die Koordinaten  $\vec{v}'$  berechnet.

$$\vec{v}' = \vec{o} + v_x \cdot \vec{e}_x + v_y \cdot \vec{e}_y + v_z \cdot \vec{e}_z \quad (3.3)$$

Um Koordinaten von einem übergeordnetem System in ein untergeordnetes System zu überführen, muss Gleichung 3.3 als lineares Gleichungssystem mit den Unbekannten  $v_x$ ,  $v_y$  und  $v_z$  betrachtet werden.

$$\begin{aligned} v'_x - o_x &= v_x \cdot e_{xx} + v_y \cdot e_{yx} + v_z \cdot e_{zx} \\ v'_y - o_y &= v_x \cdot e_{xy} + v_y \cdot e_{yy} + v_z \cdot e_{zy} \\ v'_z - o_z &= v_x \cdot e_{xz} + v_y \cdot e_{yz} + v_z \cdot e_{zz} \end{aligned} \quad (3.4)$$

In Listing A.5 ist die Lösung linearer Gleichungssystem durch den Gauß-Algorithmus mit Pivotsuche nach [26, S. 155–156] und [27, S. 31] implementiert. Dabei werden die Klassen `SystemOfEquation` und `SystemOfEquationSolution` zur Abbildung des linearen Gleichungssystems und der Lösung genutzt. Der vollständige Algorithmus zur Koordinatentransformation zwischen beliebigen Systemen ist in Listing A.4 dargestellt. Koordinaten, die als Instanzen von `Coordinates` und in unterschiedlichen Koordinatensystemen angegeben

sind, lassen sich durch diesen Algorithmus in das gleiche Koordinatensystem überführen. Damit können alle Operationen, die für die Klasse `PrimitiveVector3` implementiert sind, für beliebige Koordinaten in verschiedenen Systemen durchgeführt werden.

Für die Klassen `Vector3D` (Vektor), `Point3D` (Punkt) und `UnitVector3D` (Einheitsvektor) sind spezifische Operationen implementiert. Sie dienen der eindeutigen Zuordnung von Koordinaten zu diesen Objekten bei zusammengesetzten Klassen, die im nächsten Abschnitt definiert werden.

### 3.2.2. Geraden und Ebenen

Um Strahlen, Detektorpixel oder Voxelränder abzubilden, werden Geraden und Ebenen definiert. Eine Gerade  $\ell$ , auf der die Punkte  $P_\ell$  liegen, kann als Parameterform durch Gleichung 3.5 dargestellt werden. Bei  $\vec{u}_\ell$  handelt es sich um den Stützvektor, der einen beliebigen Punkt auf der Geraden darstellt. Der Vektor  $\vec{v}_\ell$  stellt die Richtung der Geraden dar. Bei dem Richtungsvektor handelt es sich um einen Einheitsvektor  $|\vec{v}_\ell| = 1$ . Der Parameter  $c$  kann beliebig variiert werden und entspricht dem Abstand eines Punktes auf der Geraden zu dem Auflagepunkt  $\vec{u}_\ell$ . Wird der Wertebereich von  $c$  auf  $c \geq 0$  begrenzt, wird ein *Strahl* dargestellt.

$$\ell : \vec{p}_\ell = \vec{u}_\ell + c \cdot \vec{v}_\ell \quad (3.5)$$

mit:  $|\vec{v}_\ell| = 1 \quad \text{und} \quad c \in \mathbb{R}$

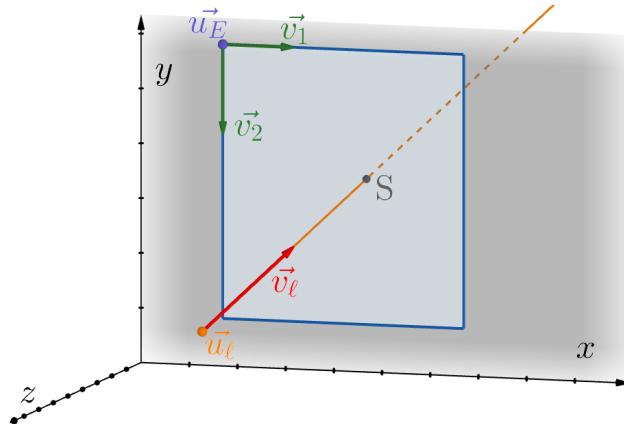
Analog zur Definition einer Geraden kann eine Ebene mit einem zusätzlichem Richtungsvektor definiert werden. Die Richtungsvektoren sind dabei orthogonal zueinander. Durch Variation beider Parameter  $a$  und  $b$  ergeben sich alle Punkte, die in der Ebene liegen. Bei Begrenzung der Wertebereiche der Parameter auf  $a_{\min} \leq a \leq a_{\max}$  und  $b_{\min} \leq b \leq b_{\max}$  entsteht eine rechteckige Fläche mit den Seitenlängen  $a_{\max} - a_{\min}$  und  $b_{\max} - b_{\min}$ .

$$E : \vec{p}_E = \vec{u}_E + a \cdot \vec{v}_1 + b \cdot \vec{v}_2 \quad (3.6)$$

mit:  $\vec{v}_1 \perp \vec{v}_2 \quad \text{und} \quad a, b \in \mathbb{R}$

Eine Ebene kann an Stelle ihrer Richtungsvektoren mit einem Normalenvektor  $\vec{n}$  beschrieben. Dieser ergibt sich aus dem Kreuzprodukt der beiden Richtungsvektoren  $\vec{n} = \vec{v}_1 \times \vec{v}_2$ . In Abbildung 3.4 sind beispielhaft ein Strahl und eine begrenzte Ebene sowie die jeweiligen Stützpunkte und Richtungsvektoren zu sehen. Außerdem ist der Schnittpunkt S der beiden Objekte dargestellt.

Zur Bestimmung eines Schnittpunktes S müssen Gleichung 3.5 und 3.6 gleichgesetzt werden. Daraus ergibt sich ein lineares Gleichungssystem 3.7 mit den Variablen  $a$ ,  $b$  und  $c$ . Dieses kann wie das Gleichungssystem 3.4 über den Gauß-Algorithmus gelöst werden. Die Lösung entspricht dann den Parametern der Geraden und Ebene, bei dem der Schnittpunkt liegt. Gibt es keine Lösung, sind Gerade und Ebene parallel. Sind die Parameter außerhalb des erlaubten

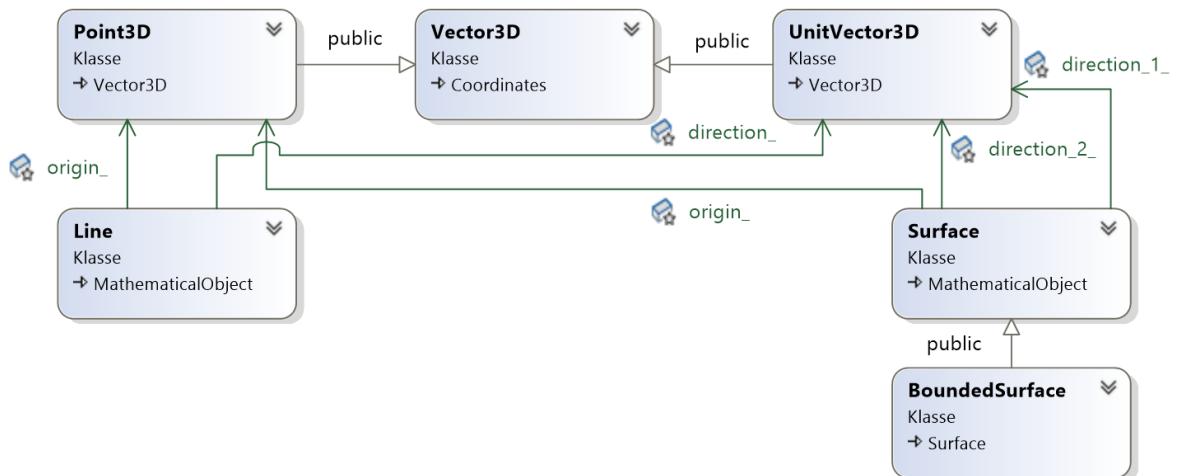


**Abbildung 3.4.:** Gerade und Ebene im kartesischen Koordinatensystem. Die Gerade und Ebene sind durch ihre Stützvektoren  $\vec{u}_\ell$  und  $\vec{u}_E$  sowie ihre Richtungsvektoren  $\vec{v}_\ell$ ,  $\vec{v}_1$  und  $\vec{v}_2$  bestimmt. Sie schneiden sich im Punkt S

Wertebereiches, befindet sich der Schnittpunkt außerhalb der begrenzten Ebene.

$$\begin{aligned} S = \ell \cap E \rightarrow \vec{p}_\ell = \vec{p}_E \\ \Rightarrow \vec{u}_\ell - \vec{u}_E = a \cdot \vec{v}_1 + b \cdot \vec{v}_2 - c \cdot \vec{v}_\ell \end{aligned} \quad (3.7)$$

Implementiert sind Geraden, Ebenen und begrenzte Ebenen entsprechend der Gleichungen 3.5 und 3.6. In Abbildung 3.5 ist das Klassendiagramm dieser Objekte dargestellt. Geraden (Line), Ebenen (Surface) und begrenzte Ebenen (BoundedSurface) enthalten jeweils Stützpunkte (Point3D) und Richtungsvektoren (UnitVector3D) als Elemente. In Listing



**Abbildung 3.5.:** Klassen, die Geraden und Ebenen im dreidimensionalen Raum darstellen

A.6 ist die Funktion zur Berechnung des Schnittpunktes zwischen Geraden und Ebenen zu sehen. Durch die im vorangegangenen Abschnitt beschriebene Koordinatentransformation können auch Geraden und Ebenen, die in unterschiedlichen Koordinatensystemen definiert sind, in Beziehung gesetzt werden.

### Lot fällen

Eine wichtige Operation ist das Fällen des Lotes einer Geraden  $\ell$  durch einen Punkt P. Das Lot ist ein Vektor  $\vec{g}$ , der vom Punkt P auf den Punkt  $P_\ell$ , der auf der Geraden liegt,

zeigt. Das Lot steht senkrecht auf der Geraden. Aus diesen Bedingungen ergeben sich die Ausdrücke in Gleichung 3.8 und 3.9. Letztere beschreibt die Orthogonalität des Lotes mit dem Richtungsvektor  $\vec{v}_\ell$  der Geraden aus Gleichung 3.5.

$$\vec{g} = \vec{p}_\ell - \vec{p} \quad (3.8)$$

$$\vec{g} \perp \vec{v}_\ell \quad (3.9)$$

Damit Gleichung 3.9 erfüllt ist, muss das Skalarprodukt beider Vektoren gleich Null sein [25, S. 26]. Nach Einsetzen der Geradendefinition aus 3.5 in das Skalarprodukt ergibt sich Gleichung 3.10.

$$\begin{pmatrix} u_{\ell,x} & + & c \cdot v_{\ell,x} & - & p_x \\ u_{\ell,y} & + & c \cdot v_{\ell,y} & - & p_y \\ u_{\ell,z} & + & c \cdot v_{\ell,z} & - & p_z \end{pmatrix} \cdot \begin{pmatrix} v_{\ell,x} \\ v_{\ell,y} \\ v_{\ell,z} \end{pmatrix} = 0 \quad (3.10)$$

Die Definition des Skalarproduktes zweier Spaltenvektoren ist durch Gleichung 3.11 gegeben [25, S. 26].

$$\vec{v}_1 \cdot \vec{v}_2 = v_{1,x} \cdot v_{2,x} + v_{1,y} \cdot v_{2,y} + v_{1,z} \cdot v_{2,z} \quad (3.11)$$

Mit dieser Definition und Gleichung 3.10 lässt sich Gleichung 3.12, die nur von der Unbekannten  $c$  abhängt, aufstellen.

$$\begin{aligned} 0 = & u_{\ell,x} \cdot v_{\ell,x} + c \cdot v_{\ell,x}^2 - p_x \cdot v_{\ell,x} + u_{\ell,y} \cdot v_{\ell,y} + c \cdot v_{\ell,y}^2 - p_y \cdot v_{\ell,y} \\ & + u_{\ell,z} \cdot v_{\ell,z} + c \cdot v_{\ell,z}^2 - p_z \cdot v_{\ell,z} \end{aligned} \quad (3.12)$$

Durch Umformung von Gleichung 3.12, lässt sich Gleichung 3.13 herleiten. Die Summanden auf der rechten Seite entsprechen Skalarprodukten und der Faktor auf der linken Seite der quadrierten Länge des Vektors  $\vec{v}_\ell$ . Der Parameter  $c$  lässt sich somit durch Gleichung 3.14 bestimmen. Wird dieser Ausdruck in Gleichung 3.8 eingesetzt, ergibt sich Gleichung 3.15.

$$c \cdot (v_{\ell,x}^2 + v_{\ell,y}^2 + v_{\ell,z}^2) = p_x \cdot v_{\ell,x} + p_y \cdot v_{\ell,y} + p_z \cdot v_{\ell,z} \quad (3.13)$$

$$\Leftrightarrow c = \frac{\vec{p} \cdot \vec{v}_\ell - \vec{v}_\ell \cdot \vec{u}_\ell}{|\vec{v}_\ell|^2} \quad (3.14)$$

$$\Rightarrow \vec{g} = \vec{p}_\ell - \vec{p} = \vec{u}_\ell + \frac{\vec{p} \cdot \vec{v}_\ell - \vec{v}_\ell \cdot \vec{u}_\ell}{|\vec{v}_\ell|^2} \cdot \vec{v}_\ell - \vec{p} \quad (3.15)$$

Gleichung 3.15 ist für beliebige Punkte und Geraden implementiert.

### 3.3. Abbilden der Gerätekomponenten

Zu den Gerätekomponenten zählen die Röntgenröhre und der Detektor. Diese Komponenten sind Teil der Gantry. In Abbildung 3.6 sind die Klassen, die die Röntgenröhre (XRayTube), den Detektor (XRayDetector) und die Gantry (Gantry) repräsentieren, dargestellt. Die Gantry hat als Klassenmember je eine Instanz der Röntgenröhre und des Detektors, die jeweils

in einem Koordinatensystem definiert sind. Diese Koordinatensysteme sind wiederum im Koordinatensystem der Gantry definiert, sodass die Gantry als Ganzes rotiert und translatiert werden kann.



Abbildung 3.6.: Röntgenröhre und Detektor als Komponenten der Gantry

### 3.3.1. Strahlerzeugung

Röntgenstrahlung soll als einzelne Nadelstrahlen simuliert werden (PHY-020). Die Strahlen müssen dabei über bestimmte Eigenschaften verfügen, sodass die Interaktion der Strahlung mit den Volumenelementen des Körpermodells möglichst nahe an den physikalischen Modellen liegt (ALG-001). Wie in Abschnitt 2.1.1 aus Seite 9 erläutert, ist die Interaktion von Strahlung mit Materie energieabhängig. Idealerweise würden einzelne Photonen, die sich geradlinig bewegen und über eine bestimmte Energie verfügen, simuliert werden. Das ist nicht praktikabel, da die Anzahl der Photonen viel zu groß ist. Stattdessen werden zu Nadelstrahlen jeweilige Spektren zugeordnet. Ein Spektrum stellt dann eine große Anzahl von Photonen dar. In Abbildung 3.7 sind die Klassen, die die Nadelstrahlen und die Röntgenröhre abbilden, zu sehen. Die Röntgenröhre (XRayTube) verfügt über Eigenschaften (XRayTubeProperties) aus denen sich das emittierte Spektrum (EnergySpectrum) ergibt. Die Nadelstrahlen (Rays) Stammen von Geraden (Line) ab. Deren Eigenschaften sind durch die Klasse RayProperties beschrieben. Die wichtigste Eigenschaft ist das Spektrum des Strahls, das bei der Erzeugung der Strahlen durch die Röntgenröhre initialisiert wird.

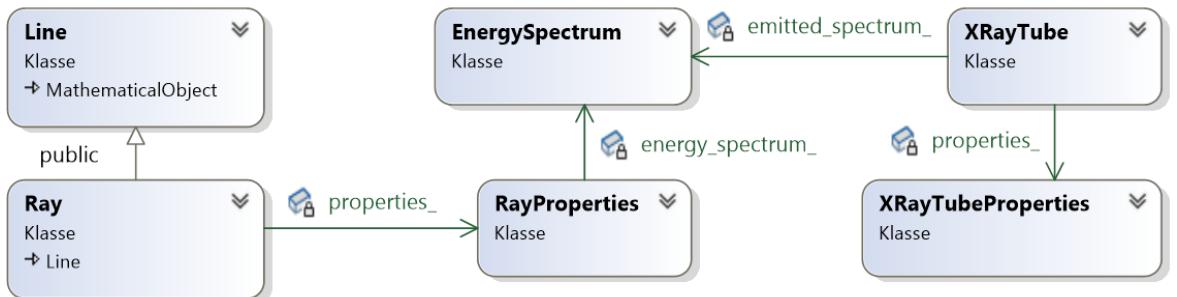


Abbildung 3.7.: Nadelstrahlen und Röntgenröhre als Klassen in der Implementierung

Das Spektrum gibt den Photonenfluss  $n(E)$  je diskretem Energieintervall  $\Delta E$  an. Das Aussehen des von der Röhre emittierten Spektrums hängt dabei von den Röhrenparametern Beschleunigungsspannung  $U_A$ , Anodenstrom  $I_A$ , dem Anodenmaterial mit der Ordnungszahl  $Z$  ab (DEV-010) sowie den Eigenschaften eines optionalen Vorfilters (DEV-011) ab. Das Spektrum wird vereinfacht durch eine Annäherung an das kontinuierliche Spektrum

der Bremsstrahlung aus Abbildung 2.2 (S. 9) implementiert. Dieses Spektrum ist charakterisiert durch die maximale Energie  $E_{\max}$  und der gesamten Strahlungsleistung  $J_{\text{ges}}$ . Die gesamte Strahlungsleistung ist die Summe der Photonenflüsse  $n(E_i)$  in einem Energieintervall gewichtet mit der jeweiligen Photonenergie  $E_i$ .

$$J_{\text{ges}} = \sum_{i=1}^{N_E} n(E_i) \cdot E_i \quad (3.16)$$

Die Erzeugung des Spektrums aus diesen Parametern ist in Listing A.7 implementiert. Dort wird zu Beginn das Spektrum durch einen linearen Verlauf mit sinkendem Photonenfluss bis zur Energie  $E_{\max}$ , ab der der Fluss Null ist, initialisiert. Gleichung 3.17 beschreibt diesen Verlauf. Der Faktor  $f$  stellt dabei den Skalierungsfaktor in der Einheit  $\frac{1}{\text{s}\cdot\text{eV}}$  dar. Die Energie  $E$  ist als diskreter Wert  $E = i \cdot \Delta E$  mit  $i \in \mathbb{N}$  zu verstehen. Die maximale Energie hängt nach Gleichung 2.3 von der Beschleunigungsspannung ab.

$$n(E) = \begin{cases} f \cdot (E_{\max} - E) & , \text{ wenn } 0 < E \leq E_{\max} \\ 0 & , \text{ sonst} \end{cases} \quad (3.17)$$

$$\text{mit: } E_{\max} = e \cdot U_A \quad \text{und: } E = i \cdot \Delta E \quad \text{für: } 1 \leq i \leq N_E$$

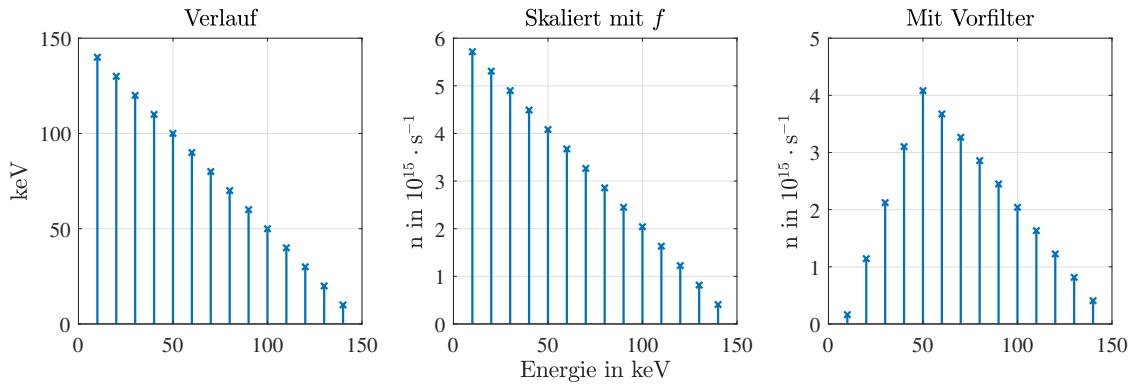
Der Faktor  $f$  ist notwendig, damit die Summe aller diskreten Photonenflüsse gewichtet mit der jeweiligen Energie der gesamten Strahlungsleistung  $J_{\text{ges}}$  aus Gleichung 2.12 (S. 9) entspricht.

$$\begin{aligned} J_{\text{ges}} &= f \cdot \sum_{i=1}^{N_E} (E_{\max} - i \cdot \Delta E) \cdot (i \cdot \Delta E) \stackrel{!}{=} k \cdot U_A^2 \cdot I_A \cdot Z \\ \Rightarrow f &= \frac{k \cdot U_A^2 \cdot I_A \cdot Z}{\sum_{i=1}^{N_E} (E_{\max} - i \cdot \Delta E) \cdot (i \cdot \Delta E)} \end{aligned} \quad (3.18)$$

Das initialisierte Spektrum, dass nur mit seiner Form  $n_{\text{Form}}(E) = (E_{\max} - E) \frac{1}{\text{s}\cdot\text{eV}}$  dem gesuchten Spektrum entspricht, wird mit dem Faktor  $f$  so skaliert, dass die Summe der Photonenflüsse gewichtet mit der jeweiligen Energie der gesamten Strahlungsleistung entspricht. In Abbildung 3.8 sind  $n_{\text{Form}}(E)$  (Links) und das korrekt skalierte Spektrum  $n(E)$  (Mitte) abgebildet. In der mittleren Abbildung entspricht die Summe der energiegewichteten Photonenflüsse der Leistung, die sich mit  $U_A = 150 \text{ keV}$ ,  $I_A = 0,2 \text{ A}$  und  $Z = 74$  ergibt.

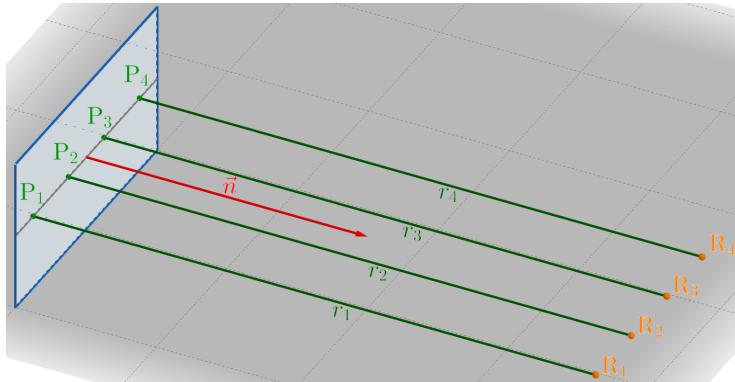
Ein optionaler Vorfilter (DEV-011), der durch eine minimale Energie und einen Gradienten charakterisiert wird, verändert die Form des Spektrums. Die minimale Energie ist jene, unterhalb derer der Photonenfluss als Null angenommen wird. Der Gradient bestimmt die Steilheit der Flanke, die den Photonenfluss begrenzt. In Abbildung 3.8 ist in der rechten Darstellung das Spektrum nach einem Vorfilter zu sehen.

Die Nadelstrahlen werden durch eine Instanz der Röntgenröhre als in einer Ebene liegender Fächerstrahl erzeugt. Dass der Fächerstrahl in einer Ebene liegt, ist darin begründet, dass nur Strahlen erzeugt werden sollen, die den Detektor bei geradlinigem Verlauf treffen können. Andere Strahlen, die nicht zum Signal beitragen, würden die benötigte Rechenzeit erhöhen.



**Abbildung 3.8.:** Erzeugung des diskreten Spektrums. Links: Verlauf des Spektrums. Mitte: Skaliertes Spektrum, in dem die Gesamtenergie den Röhrenparametern entspricht. Rechts: Skaliertes Spektrum mit Vorfilter

Für eine effiziente Simulation werden daher nur Strahlen erzeugt, die Detektorpixel treffen können. Die Anzahl der Strahlen je Pixel ist variabel. Bei mehreren Strahlen pro Pixel werden diese gleichmäßig entlang einer Geraden, die die Pixelkanten aneinander liegender Pixel verbindet, verteilt. In Abbildung 3.9 ist die Erzeugung von vier Strahlen für einen Detektorpixel zu sehen. Die Strahlen werden parallel zum jeweiligen Normalenvektor  $\vec{n}$  des Pixels mit



**Abbildung 3.9.:** Vier Strahlen, die für einen Detektorpixel erzeugt werden.  $\vec{n}$  ist der Normalenvektor des Pixels (blau),  $P_i$  sind die Auftreffpunkte der Strahlen  $r_i$  auf dem Pixel und  $R_i$  sind die Strahlenursprünge

den Ursprüngen  $R_i$  erzeugt. Die Abstände der Ursprünge von der Pixelebene entspricht der Fokus-Detektordistanz  $D_{FD}$ . In Listing A.8 ist der Algorithmus zur Erzeugung der Strahlen aus den Pixelebenen und der Strahlenanzahl je Pixel gegeben. Als Eingabeparameter nimmt er die Pixelebenen und die Anzahl der Strahlen pro Pixel entgegen. Das Spektrum eines jeden erzeugten Strahls ist das skalierte Spektrum der Röntgenröhre. Der Skalierungsfaktor ist mit  $(N_{\text{Pixel}} \cdot N_{\text{Strahlen je Pixel}})^{-1}$  reziprok zur Gesamtanzahl der Strahlen.

### 3.3.2. Detektor

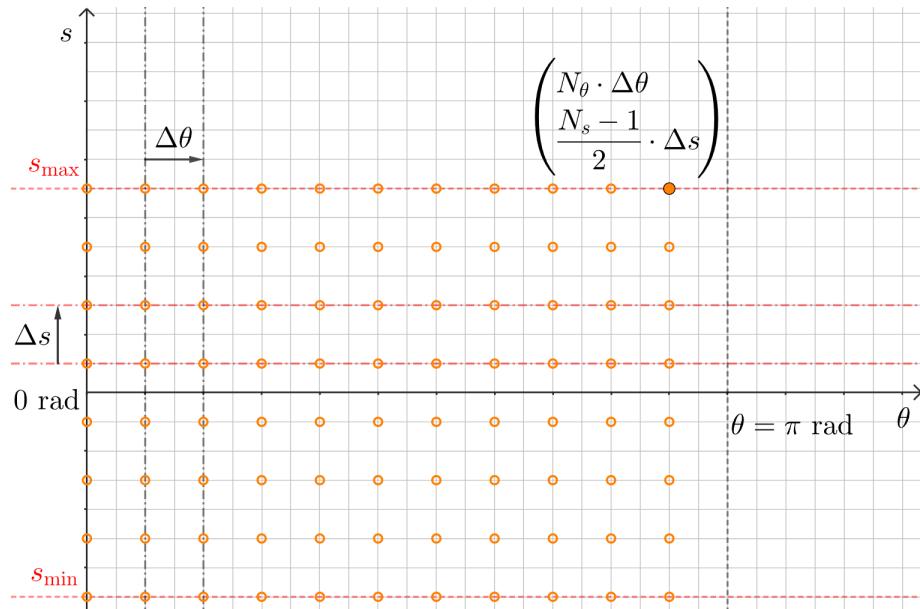
In Abbildung 3.10 sind die Klassen, die für den Detektor (`XRayDetector`) relevant sind, dargestellt. Der Detektor besteht aus mehreren Pixeln (`DetectorPixel`), die von begrenzten Ebenen (`BoundedSurface`) abgeleitet sind. Die Eigenschaften der Projektionen (`ProjectionProperties`) und physikalische Detektoreigenschaften (`PhysicalDetectorProperties`) bestimmen die Pixelanordnung (DEV-021) und die

Detektoreigenschaften (Detectorproperties).



**Abbildung 3.10.:** Zum Detektor zugehörige Klassen

Zu den Projektionseigenschaften zählen die Größe des Messfeldes  $s_{\text{Mess}}$  sowie die Anzahl  $N$  und Auflösung der Abstände  $s$  und Winkel  $\theta$ . Diese Größen geben das Aussehen des Sinogrammrasters, das in Abbildung 3.11 dargestellt ist, und die Pixelanordnung des Detektors vor. Die physikalischen Detektoreigenschaften sind das Vorhandensein sowie der Akzeptanzwinkel des Streustrahlungsrasters, der Röhren-Detektorabstand  $D_{\text{FD}}$  und die Pixelausdehnung in  $z$ -Richtung.



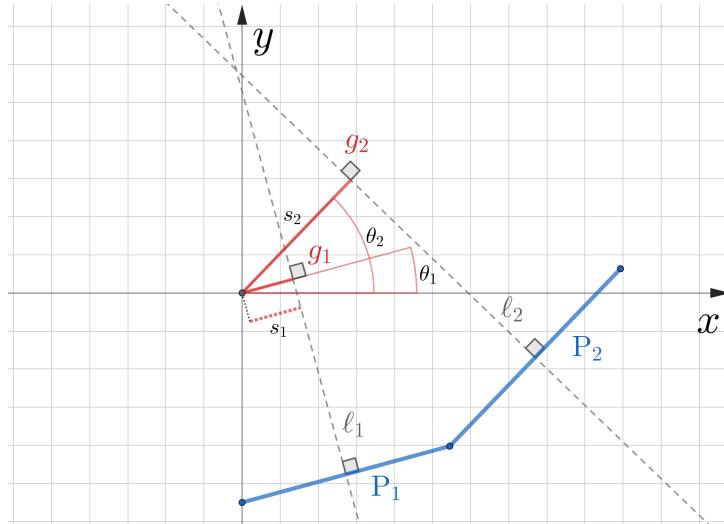
**Abbildung 3.11.:** Punkteraster im Sinogramm. Raster für  $N_\theta$  Winkel mit der Auflösung  $\Delta\theta$  und  $N_s$  Linien je Projektion mit der Auflösung  $\Delta s$

### Detektorkonstruktion

Der Detektor ist ein Pixelarray (DEV-020), das der Röntgenröhre gegenüber um die Drehachse rotiert. Der implementierte Detektor verfügt über eine Zeile, sodass nur ein Schnittbild je Rotation aufgenommen werden kann. Die Anordnung der Pixel hängt von mehreren Eingabe-parametern ab. Darunter fallen die gewünschten Eigenschaften des Rasters  $s_{\text{Mess}}$ ,  $N_s$ ,  $N_\theta$ ,  $\Delta s$ , und  $\Delta\theta$  aus Abbildung 3.11, der Röhren-Detektorabstand  $D_{\text{FD}}$  und die Pixelausdehnung in

$z$ -Richtung. Letztere spielt für die Konstruktion in der Ebene keine Rolle. Die Normalen  $\ell$  der Pixel stellen die Geraden aus Abschnitt 2.2 (S. 16) dar. Deren Lot  $g$  durch den Ursprung ist durch dessen Winkel  $\theta$  zur  $x$ -Achse und Länge  $s$  charakterisiert. Ziel der Konstruktion des Pixelarrays ist, dass durch Rotation des Pixelarrays um den Ursprung für jeden Punkt  $(\theta, s)$  im Sinogrammraster mindestens einmal Daten gesammelt werden (DEV-022). Da bei einer Rotation um den Ursprung die Lotlänge  $s$  gleich bleibt, entspricht die Anzahl der Pixel auch der Anzahl  $N_s$ . Nach einer Rotation des Pixelarrays um  $\Delta\theta$  liegt ein Lot genau an der Stelle, wo vorher ein anderes Lot mit anderer Länge lag. Dadurch wird jeder Winkel von jeder Lotlänge getroffen und somit das Raster abgetastet. Ist das Raster vollständig abgetastet, kann die gefilterte Rückprojektion in Parallelstrahlgeometrie ausgeführt werden.

Die Geometrie des Pixelarrays ist durch die Anforderung an die äquidistante Abtastung nicht trivial. Daher wird ein Konstruktionsalgorithmus implementiert, der die Pixel nacheinander erstellt. In Abbildung 3.12 sind zwei benachbarte Pixel sowie deren Pixelnormalen und Lote zu den Normalen dargestellt. Der Pixel  $P_1$  wird als erstes konstruiert. Ausgangspunkt ist



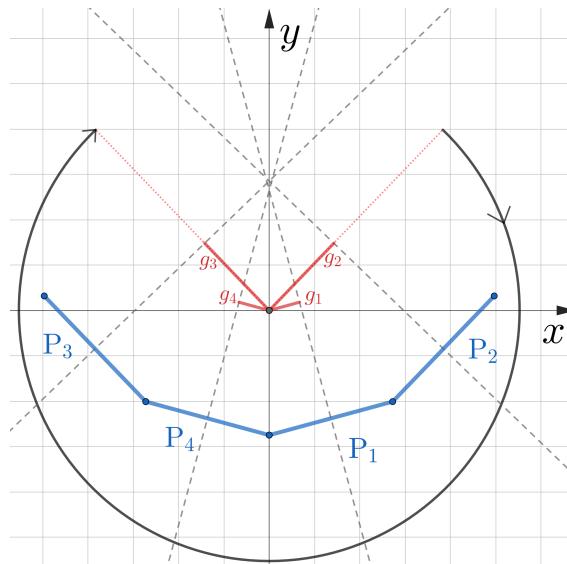
**Abbildung 3.12.:** Konstruktion zweier Pixel des Pixelarrays.  $g_i$  sind die Lote der Geraden  $\ell_i$ , die senkrecht zu den Pixeln  $P_i$  sind. Die Lote sind durch ihre Länge  $s_i$  und ihren Winkel  $\theta_i$  charakterisiert

das Lot mit der gewünschten Länge  $s_1$  und gewünschtem Winkel  $\theta_1$ . Der Stützpunkt der Pixelebene ist jener, der entlang der Geraden  $\ell$  mit dem halben Röhren-Detektorabstand  $\frac{D_{FD}}{2}$  zum Schnittpunkt des Lotes mit der Geraden liegt. Ein Richtungsvektor der Pixelebene ist parallel zum Lot und der andere parallel zur  $z$ -Achse. Die Breite des Pixels in  $x, y$ -Richtung wird so festgelegt, dass mit mittigem Stützpunkt die  $y$ -Achse berührt wird. Für den nächsten Pixel wird auf die gleiche Weise vorgegangen. Der einzige Unterschied liegt darin, dass die Pixelbreite so festgelegt wird, dass sich die benachbarten Pixel berühren.

Durch den Konstruktionsalgorithmus muss nur eine Seite des Pixelarrays erstellt werden. Der Grund dafür ist, dass eine Multiplikation mit  $-1$  der  $x$ -Komponenten aller Stütz- und Richtungsvektoren, die Pixel an der  $y$ -Achse spiegelt. Die Implementierung des eben beschriebenen Konstruktionsalgorithmus ist in Listing A.9 dargestellt.

Eine Rotation des Pixelarrays im Uhrzeigersinn um  $2 \cdot \pi$  in Schritten von  $\Delta\theta$  würde das Raster vollständig abtasten. Die Anzahl der Teilrotationen entspricht dann  $2 \cdot N_\theta$ , da  $\Delta\theta = \frac{\pi}{N_\theta}$  ist. Bei einer vollständigen Rotation würden Punkte im Raster doppelt abgetastet, was unnötig Rechenzeit beansprucht. Damit jeder Punkt im Raster mindestens einmal getroffen wird, muss so weit rotiert werden, dass die Lote der äußersten Pixel übereinanderliegen. Das wird veranschaulicht in Abbildung 3.13. Dort ist ein Pixelarray mit vier Pixeln dargestellt. Dieses wird insgesamt um den Winkel  $\Theta$  in Schritten von  $\Delta\theta$  im Uhrzeigersinn rotiert. Nach der Rotation liegt  $g_2$  an der Position von  $g_3$  vor der Rotation, was dem gleichen Punkt im Raster entspricht. Der Winkel  $\Theta$  lässt sich aus der Pixelanzahl  $N_s$  berechnen. Daraus ergibt sich die Anzahl der Teilrotationen.

$$\begin{aligned}\Theta &= N_{\text{Rotationen}} \cdot \Delta\theta = \pi + (N_s - 1) \cdot \Delta\theta \\ \Rightarrow N_{\text{Rotationen}} &= \frac{\Theta}{\Delta\theta} = \frac{\pi}{\Delta\theta} + N_s - 1 = N_\theta + N_s - 1\end{aligned}\quad (3.19)$$



**Abbildung 3.13.:** Minimale Anzahl der Teilrotationen zur vollständigen Rasterabtastung

## Detection

Um einen Strahl mit dem Detektor zu detektieren, wird für jeden einzelnen Pixel überprüft, ob ein Schnittpunkt der Pixelebene mit dem Strahl existiert. Das wird durch die in Abschnitt 3.2.2 (S. 49) beschriebene Schnittpunktbestimmung eines Strahls mit einer begrenzten Ebene realisiert. Verfügt der Detektor über ein Streustrahlenraster (DEV-023), wird der Winkel zwischen dem Strahl und dem Pixel mit dem Akzeptanzwinkel verglichen. Ist der Winkel zwischen der Ebenennormalen und dem eintreffenden Strahl kleiner als der Akzeptanzwinkel wird der Strahl detektiert. Ist er größer, wird der Strahl verworfen. Der Winkel  $\varphi$  zwischen einer Gerade mit dem Richtungsvektor  $\vec{v}_l$  und einer Ebenen mit dem Normalenvektor  $\vec{n}$ , lässt

sich aus dem Skalarprodukt der beiden Vektoren nach Gleichung 3.20 [25, S. 66] bestimmen.

$$\varphi = \arccos \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| \cdot |\vec{v}_2|} \quad (3.20)$$

Zur Beschleunigung der Detektion, wird bei der Strahlerzeugung aus Abschnitt 3.3.1 der Index des Detektorpixels, dem ein Strahl zu Grunde liegt, gespeichert. Der Pixel mit diesem Index wird als erstes auf einen Schnitt mit dem Strahl überprüft. Strahlen, die nicht gestreut wurden, werden daher immer beim ersten Versuch detektiert. In Listing A.10 ist die Bestimmung des getroffenen Pixels implementiert. Eine Variable in der Strahleigenschaften zeigt dabei an, ob der gespeicherte Pixelindex korrekt<sup>2</sup> ist. Dadurch müssen für die Strahlen erster Ordnung für die Bestimmung des Pixelindex keine Gleichungssysteme gelöst werden. In Listing A.11 ist die Detektion eines Strahls implementiert. Für jeden Detektorpixel werden im Falle eines Treffers die Strahleigenschaften für die spätere Auswertung gespeichert.

## 3.4. Beschreibung von Körpermodellen

Das Körpermodell ordnet allen Punkten in einem begrenzten Raum bestimmte Eigenschaften zu. In Abbildung 3.14 sind die in der Implementierung des Körpermodells definierten Klassen dargestellt. Das Körpermodell (Model) ist definiert in einem eigenen Koordinatensystem und legt die Größe (Tuple3D voxel\_size\_) und Anzahl (Index3D number\_of\_voxel\_3D\_) der Volumenelemente in jeweils allen drei Raumrichtungen fest. Das Körpermodell enthält zu jedem Volumenelement (Voxel) dessen Eigenschaften (VoxelData data\_) bezogen auf die Interaktion mit Röntgenstrahlung. Jedes Voxel verfügt neben diesen Eigenschaften über eine Größe (Tuple3D size\_), einen Ursprung (Point3D origin\_corner\_), von dem aus sich das Voxel ausdehnt, und sechs begrenzte Ebenen, die das Voxel begrenzen.

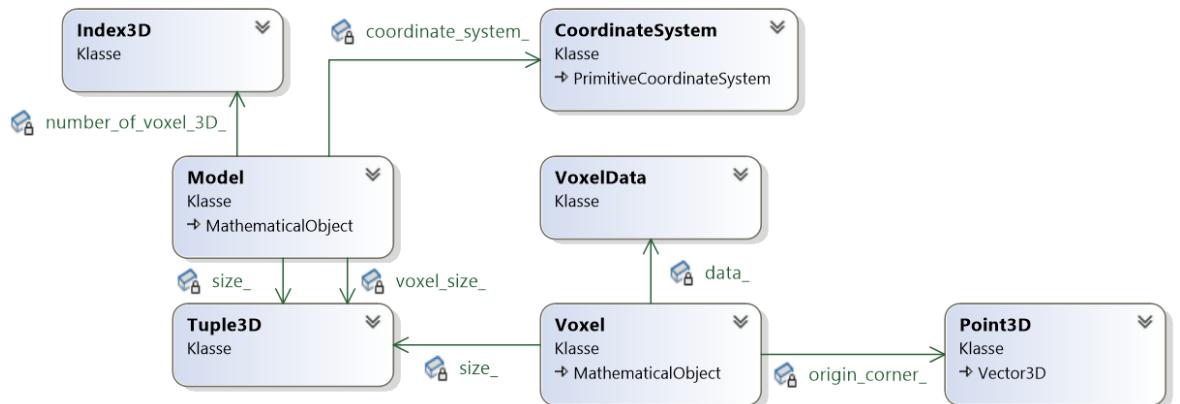


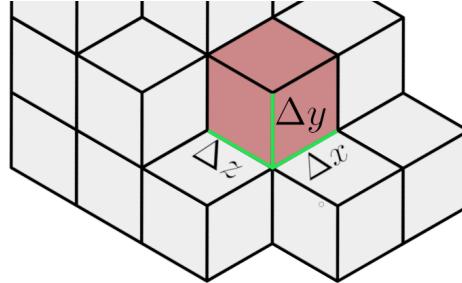
Abbildung 3.14.: Klassen zur Abbildung des Körpermodells

### 3.4.1. Volumenelemente

Die quaderförmigen Voxel (PHY-010) verfügen über eine einheitliche Größe und sind in einem regelmäßigen Gitter angeordnet, sodass jede Quaderfläche immer genau mit einer

<sup>2</sup>Korrekt heißt, dass nicht überprüft wird, ob der Strahl tatsächlich diesen Pixel trifft.

anderen Quaderfläche zusammenhängt. In Abbildung 3.15 ist die Zusammensetzung eines Körpermodells aus mehreren quaderförmigen Voxeln abgebildet. Die Seitenlängen der Voxel sind mit  $\Delta x$ ,  $\Delta y$  und  $\Delta z$  gegeben. Deren Anzahl in jede Raumrichtung mit  $N_x$ ,  $N_y$  und  $N_z$ . Das Körpermodell ist im Koordinatensystem so platziert, dass genau ein Voxel den Ursprung berührt und sich die Voxel nur in positiver Richtung entlang der Koordinatenachsen ausdehnen.



Quelle: [28] modifiziert

**Abbildung 3.15.:** Volumenelemente im Körpermodell. Die quaderförmigen Voxel haben die Seitenlängen  $\Delta x$ ,  $\Delta y$  und  $\Delta z$

Jedes Voxel verfügt über eindeutige ganzzahlige Indizes  $i_{x, y, z}$ , die der Position des Voxels im Gitter entsprechen. Diese Indizes befinden sich entsprechend Gleichung 3.21 in einem bestimmten Wertebereich. Die Ecke, von der sich ein Voxel in positive Richtung ausdehnt, ist durch die Koordinaten  $(i_x \cdot \Delta x, i_y \cdot \Delta y, i_z \cdot \Delta z)$  gegeben. Alle Punkte  $\vec{p}_{\text{Voxel}}$  innerhalb eines Voxels sind durch Gleichung 3.22 gegeben.

$$i_{x, y, z} = 0, 1, 2, \dots, N_{x, y, z} - 1 \quad (3.21)$$

$$\vec{p}_{\text{Voxel}} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \quad \text{mit: } \begin{array}{l} i_x \cdot \Delta x \leq p_x < (i_x + 1) \cdot \Delta x \\ i_y \cdot \Delta y \leq p_y < (i_y + 1) \cdot \Delta y \\ i_z \cdot \Delta z \leq p_z < (i_z + 1) \cdot \Delta z \end{array} \quad (3.22)$$

Ist ein beliebiger Punkt  $P = (x, y, z)$  im lokalen Koordinatensystem des Körpermodells gegeben, ist es notwendig die Indizes des Voxels, in dem der Punkt liegt, bestimmen zu können. Durch Umstellen von Gleichung 3.22 können die Indizes berechnet werden. Da die Indizes nur ganzzahlige Werte annehmen dürfen, muss die Abrundungsfunktion in Gleichung 3.23 zur Herstellung von Gleichheit benutzt werden.

$$\begin{aligned} i_x &\leq \frac{x}{\Delta x} & i_x &= \lfloor \frac{x}{\Delta x} \rfloor \\ i_y &\leq \frac{y}{\Delta y} \Rightarrow i_y &= \lfloor \frac{y}{\Delta y} \rfloor \\ i_z &\leq \frac{z}{\Delta z} & i_z &= \lfloor \frac{z}{\Delta z} \rfloor \end{aligned} \quad (3.23)$$

In Listing A.12 ist die Funktion, die die Berechnung der Indizes aus lokalen Koordinaten vornimmt, dargestellt. Dabei muss nicht explizit die Abrundungsfunktion genutzt werden, da eine Typkonvertierung von Gleitkommazahlen zu Ganzzahlen diesen Effekt innehaltet [29, S. 38].

### 3.4.2. Datenspeicherung

Alle Voxeleigenschaften werden im Körpermodell in einer eindimensionalen Datenstruktur, die kontinuierlich im Speicher vorliegt, gespeichert. Die dreidimensionalen Indizes müssen daher auf einem eindimensionalen Index  $i$  abgebildet werden, um die Daten eines Voxels aus dem Speicher lesen zu können. Dazu wird die Berechnung nach Gleichung 3.24 genutzt. Diese Gleichung liefert nur eindeutige Indizes  $i$ , wenn Gleichung 3.21 erfüllt ist.

$$i = i_x + i_y \cdot N_x + i_z \cdot N_x \cdot N_y \quad (3.24)$$

Zur Berechnung der dreidimensionalen Indizes aus dem eindimensionalen Index kann die Modulo Funktion ( $a \bmod b$ ), die den Rest bei ganzzahliger Division  $a$  durch  $b$  als Wert hat, genutzt werden. Der  $x$ -Index  $i_x$  entspricht dem Divisionsrest von  $\frac{i}{N_x}$ , da  $i_y$  und  $i_z$  jeweils mit Vielfachen von  $N_x$  zu  $i$  beitragen. Der  $y$ -Index entspricht der ganzzahligen Anzahl der vorkommenden  $N_x$  ohne den Beitrag von  $N_y$ . Der  $z$ -Index ist die ganzzahlige Anzahl der  $N_x \cdot N_y$ .

$$i_x = i \bmod N_x \quad (3.25)$$

$$i_y = \left\lfloor \frac{i}{N_x} \right\rfloor \bmod N_y \quad (3.26)$$

$$i_z = \left\lfloor \frac{i}{N_x \cdot N_y} \right\rfloor \quad (3.27)$$

Der benötigte Speicherplatz eines Modells  $B_{\text{Modell}}$  ergibt aus der Größe der Modellinformationen  $B_0$ , die die Voxelanzahl und -größe, den Modellnamen, die Lage des Koordinatensystems und weitere Daten enthält. Hinzu kommt der dominierende Speicherplatzbedarf der Voxeldaten, der von der Voxelanzahl und dem Speicherbedarf je Voxel  $B_{\text{Voxel}}$  abhängt.

$$B_{\text{Modell}} = B_0 + N_x \cdot N_y \cdot N_z \cdot B_{\text{Voxel}} \quad (3.28)$$

### 3.4.3. Voxeleigenschaften

Jedes Voxel enthält Informationen (`VoxelData`) über die Röntgenabsorption (PHY-011) und eine optionale Spezialeigenschaft (PHY-012). Die Röntgenabsorption  $\mu_{\text{abs}}$  wird als eine Zahl mit dem Typ `double` gespeichert. Der gespeicherte, dimensionslose Zahlenwert  $Z_g$  ist als linearer Absorptionskoeffizient in der Einheit  $\frac{1}{\text{mm}}$  zu verstehen und spiegelt die Absorption nach Gleichung 2.35 auf Seite 15 wieder.

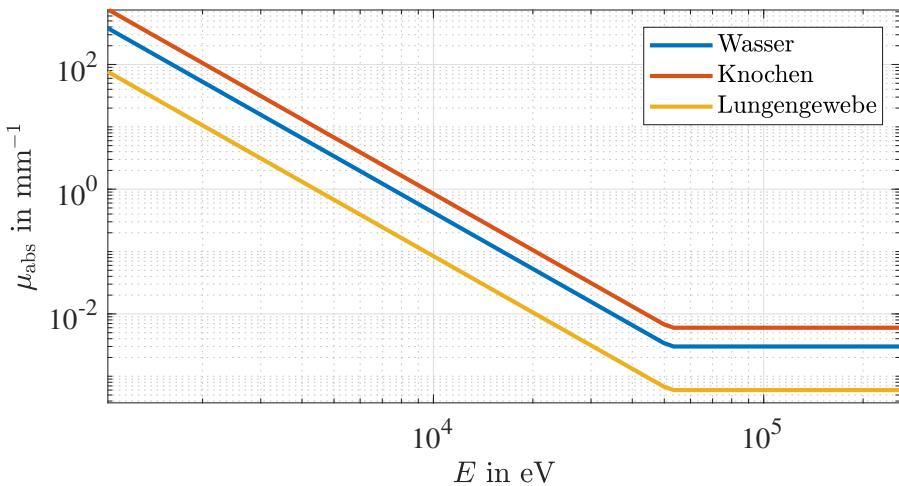
In Abschnitt 2.1.1 auf Seite 16 wurde am Beispiel von Wasser die Energieabhängigkeit der Röntgenabsorption beschrieben. Ein einzelner Wert, wie er in den Voxeleigenschaften gespeichert ist, kann das energieabhängige Verhalten der Absorption nicht vollständig abbilden. Daher wird ein vereinfachtes Modell, das auf dem Verlauf des Absorptionskoeffizienten von Wasser beruht, eingeführt. In diesem Modell wird der Absorptionskoeffizient  $\mu_{\text{abs}}(E)$  für ein Material als Funktion der Energie beschrieben. Für niederenergetische Strahlung ist die Absorption proportional zu  $E^{-3}$ . Für höhere Energien ist sie konstant. Der gespeicherte

Zahlenwert  $Z_g$  entspricht der Absorption im konstanten Bereich. Die Energie, die den energieabhängigen vom konstanten Teil trennt, wird als  $E_W$  bezeichnet. In Gleichung 3.29 ist die Absorption als Funktion der Energie in Abhängigkeit von dem gespeicherten Zahlenwert beschrieben. Gleichung 3.30 drückt diesen Zusammenhang mithilfe der Minimum-Funktion  $\min()$  aus.

$$\mu_{\text{abs}}(E) = \begin{cases} Z_g \frac{1}{\text{mm}} \cdot E_W^3 \cdot E^{-3} & , \text{ wenn } E < E_W \\ Z_g \frac{1}{\text{mm}} & , \text{ wenn } E \geq E_W \end{cases} \quad (3.29)$$

$$\mu_{\text{abs}}(E) = Z_g \frac{1}{\text{mm}} \cdot \left( \frac{E_W}{\min(E, E_W)} \right)^3 \quad (3.30)$$

Die Energie  $E_W$  liegt für Wasser, abgelesen aus Abbildung 2.4, bei ungefähr 52 keV. Da für einen Voxel nur der Zahlenwert  $Z_g$  ohne Informationen über das konkrete Material gespeichert und der größte Teil von Gewebe Wasser ist, wird  $E_W$  für die Berechnung der Absorption eines jeden Voxels genutzt. In Abbildung 3.16 ist die Näherung nach Gleichung 3.30 gezeigt. Dort sind die energieabhängigen Absorptionskoeffizienten für Wasser mit  $\mu_{\text{Wasser}} = 3 \cdot 10^{-2} \frac{1}{\text{mm}}$ , Knochen  $\mu_{\text{Knochen}} = 2 \cdot \mu_{\text{Wasser}}$  und Lungengewebe  $\mu_{\text{Lunge}} = 0,2 \cdot \mu_{\text{Wasser}}$  dargestellt.



**Abbildung 3.16.:** Näherung der energieabhängigen Absorption für Wasser, Knochen und Lungengewebe

Die optionalen Spezialeigenschaften sind durch eine Maske aus acht Bits in den Voxeldaten repräsentiert. Da ein Bit genau einer Eigenschaft entspricht, können bis zu acht Eigenschaften abgebildet werden. Hat ein Voxel keine Spezialeigenschaft, sind alle Bits gleich Null (0b00000000). Durch Setzen einzelner Bits kann jede beliebige Kombination von Spezialeigenschaften einem Voxel zugewiesen werden. In der Implementierung sind bisher nur zwei Eigenschaften abgebildet. Die Eigenschaft *Metall* (0b00000001) und *undefiniert* (0b00000010). Letztere bietet die Möglichkeit einen Standardwert abzubilden. Verfügt ein Voxel über die Eigenschaft Metall (PHY-013) wird der Absorptionskoeffizient des Voxels anders berechnet als eben beschrieben. In Gleichung 3.30 wird der Zahlenwert  $Z_g \frac{1}{\text{mm}}$  durch den Absorptionskoeffizienten von Titan  $\mu_{\text{Ti}}$  für  $E > E_W$  ersetzt. Dieser wird grob abgeschätzt

über jenen von Aluminium  $0,0813 \frac{1}{\text{mm}}$  [11, S. 275] und dem Verhältnis der Dichten von Titan zu Aluminium  $\frac{4,1}{2,7}$  [19, S. 1050]. Der Koeffizient ist damit  $\mu_{\text{Ti}} = 0,0135 \frac{1}{\text{mm}}$ . Die Berechnung des energieabhängigen Absorptionskoeffizienten ist in Gleichung 3.31 abgebildet. Zusätzlich zum Austausch des Zahlenwertes  $Z_g \frac{1}{\text{mm}}$  durch  $\mu_{\text{Ti}}$  skaliert der Faktor  $f_M$  den Koeffizienten, sodass Metallartefakte in ihrer Auswirkung auf das rekonstruierte Bild einstellbar sind.

$$\mu_{\text{abs}}(E) = f_M \cdot \mu_{\text{Ti}} \cdot \left( \frac{E_W}{\min(E, E_W)} \right)^3 \quad (3.31)$$

Der Speicherplatzbedarf eines Voxeldatums beträgt 9 B (Byte), da der Absorptionskoeffizient als `double` (8 B) und die Spezialeigenschaften als `unsigned char` (1 B) gespeichert sind. Der Gesamtbedarf lässt sich mit Gleichung 3.28 für ein Beispielmodell, das über 500 Voxel in jede Raumrichtung verfügt, als ungefähr 1,125 GB berechnen.

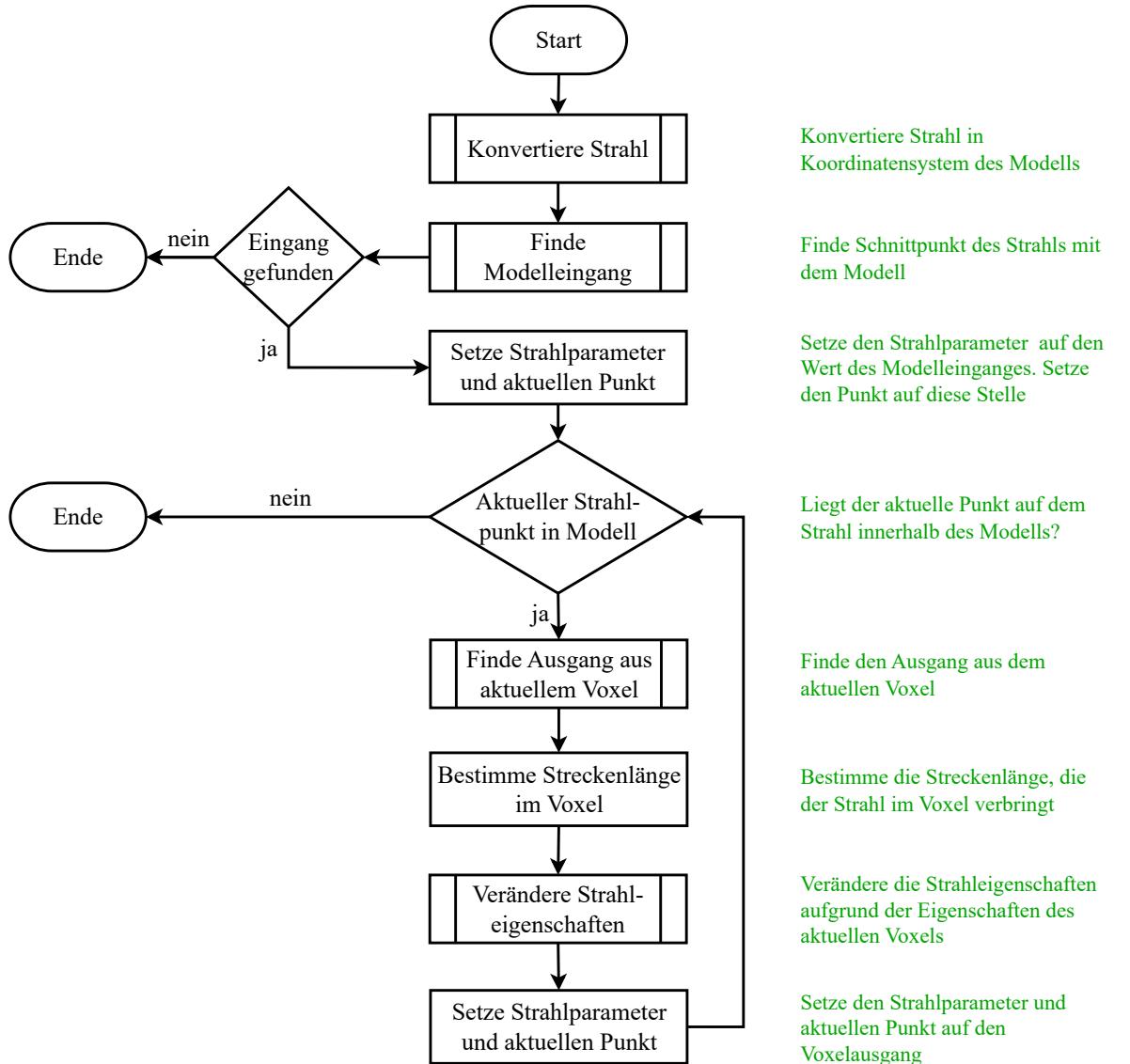
### 3.5. Algorithmus zur Strahlverfolgung

Der Kernbaustein zur Abbildung der physikalischen Modelle im Simulator ist die Strahlverfolgung. Ziel der Strahlverfolgung ist es, die Eigenschaftsveränderungen eines Nadelstrahls bei Durchgang durch das Körpermodell entsprechend den physikalischen Modelle abzubilden. Da jedes Voxel über individuelle Eigenschaften verfügt, muss bestimmt werden, in welcher Reihenfolge welche Voxel getroffen werden (PHY-021). In Abbildung 3.17 ist der Programmablaufplan für die Strahlverfolgung dargestellt. Die Strahlverfolgung ist als Memberfunktion der `Model`-Klasse definiert, da vor allem auf die Voxeleigenschaften zugegriffen wird. In Listing A.13 ist die Strahlverfolgung implementiert.

Als erster Schritt wird der Strahl, der durch das Körpermodell verfolgt werden soll, in das Koordinatensystem des Körpermodells überführt. Dadurch entfällt das mehrfache Lösen von Gleichungssystemen bei der Koordinatentransformation. Im nächsten Schritt wird der Eingang in das Körpermodell, welcher der Schnittpunkt des Strahls  $r$  mit einer der Randflächen des Körpermodells ist, ermittelt. Die Randflächen sind sechs begrenzte Ebenen  $R_{M,j}$  die durch die Größe des Körpermodells  $G_{x,y,z}$ , die in jeder Raumrichtung der Voxelanzahl  $N_{x,y,z}$  multipliziert mit der Seitenlänge der Voxel  $\Delta x$ ,  $\Delta y$  oder  $\Delta z$  entspricht, definiert sind.

$$G_x = N_x \cdot \Delta x; \quad G_y = N_y \cdot \Delta y; \quad G_z = N_z \cdot \Delta z$$

Die Randflächen  $R_{M,j}$  können nach ihrer Lage im Koordinatensystem bezeichnet werden. Da das Körpermodell ein Quader ist, existieren genau drei Ebenen in jeweils zwei Richtungen. Die drei Ebenen sind die jeweils verschobenen  $xy$ -,  $xz$ - und  $yz$ -Ebenen des Koordinatensystems. Diese Verschiebung findet vom Modellmittelpunkt aus betrachtet in negativer (-) oder positiver (+) Richtung entlang der  $z$ -,  $y$ - oder  $x$ -Achse statt, damit die Ebenen auf den Randflächen liegen. Die sechs Randflächen werden damit als  $R_{xy-z}$ ,  $R_{M,xy+z}$ ,  $R_{M,xz-y}$ ,  $R_{M,xz+y}$ ,



**Abbildung 3.17.:** Programmablaufplan der Strahlverfolgung durch das Modell

$R_{M,yz-x}$  und  $R_{M,yz+x}$  bezeichnet.

$$R_{M,xy-z} = \begin{pmatrix} a \\ b \\ 0 \end{pmatrix} \quad R_{M,xy+z} = \begin{pmatrix} a \\ b \\ G_z \end{pmatrix} \quad \begin{aligned} 0 &\leq a < G_x \\ 0 &\leq b < G_y \end{aligned} \quad (3.32)$$

$$R_{M,xz-y} = \begin{pmatrix} a \\ 0 \\ b \end{pmatrix} \quad R_{M,xz+y} = \begin{pmatrix} a \\ G_y \\ b \end{pmatrix} \quad \begin{aligned} 0 &\leq a < G_x \\ 0 &\leq b < G_z \end{aligned} \quad (3.33)$$

$$R_{M,yz-x} = \begin{pmatrix} 0 \\ a \\ b \end{pmatrix} \quad R_{M,yz+x} = \begin{pmatrix} G_x \\ a \\ b \end{pmatrix} \quad \begin{aligned} 0 &\leq a < G_y \\ 0 &\leq b < G_z \end{aligned} \quad (3.34)$$

Zur Bestimmung der Schnittpunkte des Strahls mit den Modellrndern kann die Parameterbeschreibung des Strahls  $r$  mit den Gleichungen 3.32 bis 3.34 der Randflchen gleichgesetzt

werden.

$$r : \vec{p} = \vec{u}_r + c \cdot \vec{v}_r \stackrel{!}{=} R_{M,j} \quad (3.35)$$

Das daraus resultierende Gleichungssystem mit den Variablen  $a$ ,  $b$  und  $c$  kann wie in Abschnitt 3.2.2 auf Seite 48 beschrieben, gelöst werden. Existiert mit keiner der Randflächen ein Schnittpunkt, trifft der Strahl das Körpermodell nicht. Bei einem Schnitt kann das Körpermodell auf vier Arten und Weisen geschnitten werden. Das Körpermodell kann nicht geschnitten werden. Es kann so geschnitten werden, dass der Strahl nur eine Ecke in einem Punkt trifft, oder so, dass der Strahl in einer der Randflächen liegt. Diese beiden Fälle werden durch das schrittweise Vorgehen bei der Strahlverfolgung korrekt berücksichtigt. Der wichtigste Fall ist jener, in dem das Körpermodell genau zwei Randflächen schneidet. Jene Fläche, die mit geringerem Abstand zum Strahlursprung, der dem Parameter  $c$  entspricht, geschnitten wird, ist die Fläche an dem der Strahl in das Körpermodell eintritt. Der Punkt  $P_r$  ist der aktuelle Punkt auf dem Strahl bei einem bestimmten Parameter  $c$ . Der Parameter  $c$  erhöht sich während der Strahlverfolgung, sodass sich der Punkt  $P_r$  entlang des Strahls bewegt. Das erste Voxel auf das der Strahl trifft, ist jenes, das am Modelleingang liegt. Die Indizes des Voxels werden durch die Gleichungen in 3.23 (S. 58) bestimmt. Dabei sind die Komponenten  $x$ ,  $y$  und  $z$  die lokalen Koordinaten des Punktes  $P_r$ .

### 3.5.1. Bestimmung des Voxelausgangs

Der Punkt  $P_r$  liegt nach Eintritt in das Körpermodell gerade so in einem Voxel. Als nächstes muss jenes Voxel, das an dem aktuellen Voxel angrenzt und vom Strahl getroffen wird, bestimmt werden. Außerdem muss die Strecke, die zwischen dem Voxeleingang und -ausgang liegt, ermittelt werden. Diese wird für die Berechnung der Absorption und Streuung benötigt. Dafür werden analog zu den Randflächen des Körpermodells die Randflächen des Voxels  $R_{V,j}$  definiert. Deren Lage hängt alleinig von den Indizes des aktuellen Voxels  $i_{x,y,z}$  und der Voxelgröße  $\Delta x$ ,  $\Delta y$  und  $\Delta z$  ab.

$$R_{V,xy-z} = \begin{pmatrix} a \\ b \\ i_z \cdot \Delta z \end{pmatrix} \quad R_{V,xy+z} = \begin{pmatrix} a \\ b \\ (i_z + 1) \cdot \Delta z \end{pmatrix} \quad \begin{array}{l} i_x \cdot \Delta x \leq a < (i_x + 1) \cdot \Delta x \\ i_y \cdot \Delta y \leq b < (i_y + 1) \cdot \Delta y \end{array} \quad (3.36)$$

$$R_{V,xz-y} = \begin{pmatrix} a \\ i_y \cdot \Delta y \\ b \end{pmatrix} \quad R_{V,xz+y} = \begin{pmatrix} a \\ (i_y + 1) \cdot \Delta y \\ b \end{pmatrix} \quad \begin{array}{l} i_x \cdot \Delta x \leq a < (i_x + 1) \cdot \Delta x \\ i_z \cdot \Delta z \leq b < (i_z + 1) \cdot \Delta z \end{array} \quad (3.37)$$

$$R_{V,yz-x} = \begin{pmatrix} i_x \cdot \Delta x \\ a \\ b \end{pmatrix} \quad R_{V,yz+x} = \begin{pmatrix} (i_x + 1) \cdot \Delta x \\ a \\ b \end{pmatrix} \quad \begin{array}{l} i_y \cdot \Delta y \leq a < (i_y + 1) \cdot \Delta y \\ i_z \cdot \Delta z \leq b < (i_z + 1) \cdot \Delta z \end{array} \quad (3.38)$$

Es wäre möglich die Randfläche, durch die der Strahl aus dem Voxel austritt, wie bei dem Modelleingang, durch Lösen aller sechs Gleichungssysteme zu bestimmen. Da das viel Rechenzeit beansprucht, wird anders vorgegangen. Zu Beginn steht die Überlegung, dass nur durch Betrachtung der  $x$ ,  $y$  und  $z$  Komponenten der Strahlrichtung  $\vec{v}_r$  Randflächen, durch die

der Strahl aus dem Voxel austreten kann, ausgeschlossen werden können. In Listing A.14 ist die Implementierung der Einschränkung von möglichen Ausgangsrandflächen zu sehen. Ein Schnitt mit einer Randfläche ist nur möglich, wenn vom Inneren des Voxels aus gesehen das Vorzeichen der Richtungskomponenten des Strahl zu der betrachteten Randfläche passt. So liegt zum Beispiel die Randfläche  $R_{V,xy-z}$  aus dem Inneren eines Voxels betrachtet immer in negativer  $z$ -Richtung, sodass die  $z$ -Komponente der Strahlrichtung  $v_{r,z}$  negativ sein muss, damit diese Randfläche getroffen werden kann. Analog ergeben sich die Bedingungen für die anderen Randflächen.

$$R_{V,xy-z}, \text{ wenn } v_{r,z} < 0 \quad R_{V,xy+z}, \text{ wenn } v_{r,z} > 0 \quad (3.39)$$

$$R_{V,xz-y}, \text{ wenn } v_{r,y} < 0 \quad R_{V,xz+y}, \text{ wenn } v_{r,y} > 0 \quad (3.40)$$

$$R_{V,yz-x}, \text{ wenn } v_{r,x} < 0 \quad R_{V,yz+x}, \text{ wenn } v_{r,x} > 0 \quad (3.41)$$

Für die möglichen Randflächen kann nun versucht werden, die Schnittpunkte mit dem Strahl zu bestimmen. Dazu werden zunächst die Randflächen als unbegrenzt angenommen. Die Parameter  $a$  und  $b$  sind also unbeschränkt. Zur Schnittpunktbestimmung wird die Parameterdarstellung der Ebene mit jener des Strahls gleichgesetzt. In der Parameterdarstellung des Strahls wird der Strahlursprung durch den aktuellen Punkt  $P_r$ , der auf dem Strahl innerhalb des aktuellen Voxels liegt, ersetzt.

$$\vec{p}_r + c_V \cdot \vec{v}_r \stackrel{!}{=} R_{V,j}$$

Für jede Komponente in den Vektoren ergibt sich eine Gleichung. Davon ist jeweils genau eine Gleichung unabhängig von den Parametern  $a$  und  $b$ . Das wird erkenntlich bei Betrachtung der Gleichungen 3.36 bis 3.38. Am Beispiel der Randfläche  $R_{V,xy-z}$  ist diese Gleichung jene, die sich auf die  $z$ -Komponente bezieht.

$$\begin{aligned} p_{r,x} + c_V \cdot v_{r,x} &= a \\ p_{r,y} + c_V \cdot v_{r,y} &= b \\ p_{r,z} + c_V \cdot v_{r,z} &= i_z \cdot \Delta z \end{aligned} \quad (3.42)$$

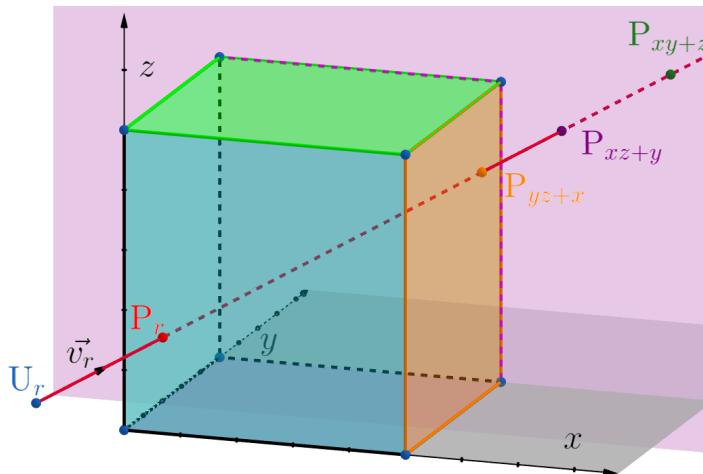
Es lässt sich also der Parameter  $c_V$  allein durch Umstellen der letzten Gleichung bestimmen. Dieser Parameter stellt dann den Abstand des aktuellen Punktes  $P_r$  zum Schnittpunkt mit der jeweiligen Randfläche dar. Die Division durch die Komponente  $v_{r,z}$  ist möglich, da durch die Bedingungen in den Gleichungen 3.39 bis 3.41 ausgeschlossen ist, dass dieser Wert gleich Null ist.

$$c_V = \frac{i_z \cdot \Delta z - p_{r,z}}{v_{r,z}} \quad (3.43)$$

Für alle Randflächen ergibt sich analog die Berechnung des Parameters  $c_V$ .

$$\begin{aligned} R_{V,xy-z}: \quad c_{V,xy-z} &= (i_z - p_{r,z}) \cdot v_{r_z}^{-1} \\ R_{V,xy+z}: \quad c_{V,xy+z} &= ((i_z + 1) \cdot \Delta z - p_{r,z}) \cdot v_{r_z}^{-1} \\ R_{V,xz-y}: \quad c_{V,xz-y} &= (i_y - p_{r,y}) \cdot v_{r_y}^{-1} \\ R_{V,xz+y}: \quad c_{V,xz+y} &= ((i_y + 1) \cdot \Delta y - p_{r,y}) \cdot v_{r_y}^{-1} \\ R_{V,yz-x}: \quad c_{V,yz-x} &= (i_x - p_{r,x}) \cdot v_{r_x}^{-1} \\ R_{V,yz+x}: \quad c_{V,yz+x} &= ((i_x + 1) \cdot \Delta x - p_{r,x}) \cdot v_{r_x}^{-1} \end{aligned}$$

Da durch die bisherigen Überlegungen bis zu drei Randflächen aus Ausgangsflächen infrage kommen, muss nun jene gefunden werden, durch die der Strahl tatsächlich austritt. Das wäre möglich durch vollständiges Lösen des Gleichungssystems 3.42 und Überprüfung der Werte von  $a$  und  $b$  auf ihre Gültigkeit bezüglich der Gleichungen 3.36 bis 3.38. Die alternative und schnellere Methode ist ein Vergleich der für die Randflächen berechneten Parameter  $c_V$ . Der Parameter  $c_V$  entspricht dem Abstand des aktuellen Punktes  $P_r$  zum Schnittpunkt. Nur die Randfläche, dessen Schnittpunkt mit dem Strahl den geringsten Abstand zum aktuellem Punkt  $P_r$  hat, ist die Ausgangsfläche. In Abbildung 3.18 ist ein Strahl, der ein Voxel schneidet, dargestellt. Der aktuelle Punkt  $P_r$  ist der Punkt, in dem der Strahl in das Voxel eintritt. Wegen der Richtung des Strahls kommen nur die drei Randflächen  $R_{V,xy+z}$  (Grün),  $R_{V,yz+x}$  (Orange) und  $R_{V,xz+y}$  (Violet) als Ausgangsebenen in Frage. Dabei ist die orange Randfläche jene, dessen Schnittpunkt mit dem Strahl  $P_{yz+x}$  die kürzeste Distanz zum Punkt  $P_r$  hat.

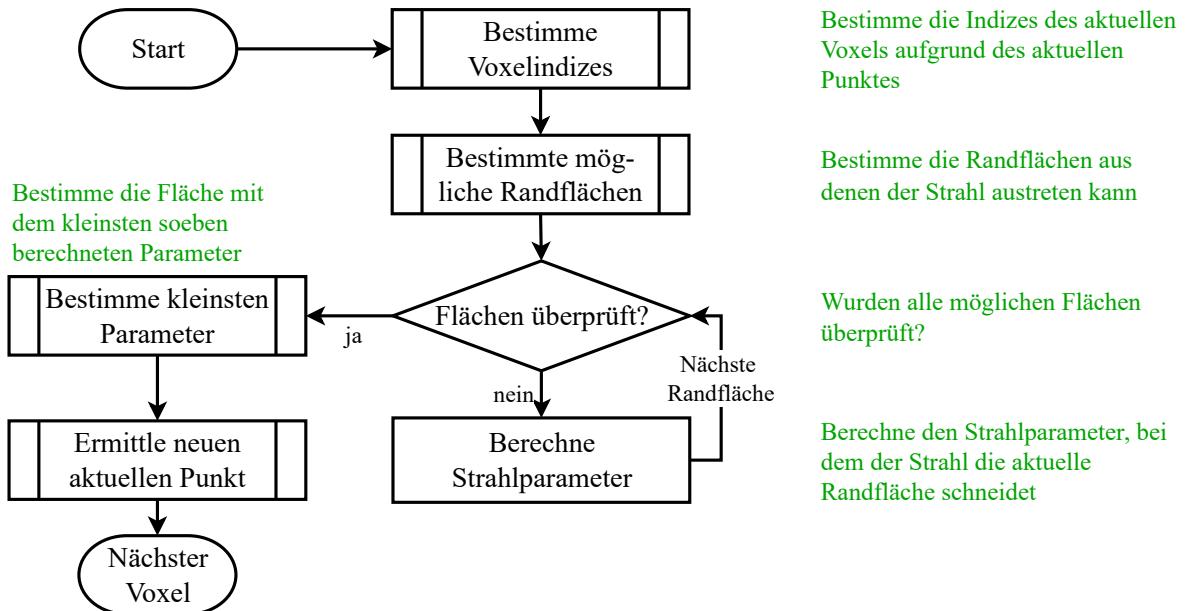


**Abbildung 3.18.:** Schnitt eines Strahls mit einem Voxel. Der Strahl hat Schnittpunkte mit den unbegrenzten Randflächen  $R_{V,xy+z}$  (Grün),  $R_{V,yz+x}$  (Orange) und  $R_{V,xz+y}$  (Violet)

Um sicherzustellen, dass das nächste Voxel eindeutig bestimmt ist, wird der berechnete Parameter  $c_V$  um  $\epsilon$  erhöht und daraus der neue aktuelle Punkt  $P_r$  ermittelt. Die Größe  $\epsilon$  befindet sich in der Größenordnung von  $10 \cdot 10^{-3}$  mm.

$$\vec{p}_{r,\text{neu}} = \vec{p}_{r,\text{alt}} + (c_V + \epsilon) \cdot \vec{v}_r \quad (3.44)$$

In Abbildung 3.19 ist das Vorgehen zur Bestimmung des Strahlaustritts aus einem Voxel als Programmablaufplan abgebildet. Dieses Unterprogramm entspricht dem Prozess „Finde Ausgang aus aktuellem Voxel“ aus Abbildung 3.17.



**Abbildung 3.19.:** Programmablaufplan zur Bestimmung des Voxelausgangs

### 3.5.2. Absorption

In den folgenden Abschnitten, die die Absorption und Streuung beschreiben, wird die Schwächung entsprechend der Implementierung durch lineare Koeffizienten und nicht durch Massenkoeffizienten beschrieben. Würden Massenkoeffizienten genutzt, wäre eine Angabe der Dichte eines Voxels, die nicht bekannt ist, notwendig. Lineare Schwächungskoeffizienten erlauben es, dass für jedes Voxel nur ein Wert gespeichert werden muss.

In jedem Voxel, das von einem Strahl getroffen wird, kommt es zur Absorption einzelner Photonen. Die Anzahl der Photonen, die nicht absorbiert werden, ist dabei durch das Schwächungsgesetz aus Gleichung 2.15 (S. 10) beschrieben. Der Photonenfluss nach Durchgang durch ein Voxel  $n(E)$  mit der Energie  $E$  ist abhängig von dem initialen Photonenfluss  $n_0(E)$ , dem linearen Absorptionskoeffizienten  $\mu_{\text{abs}}(E)$  und der Strecke  $c_v$ . Letztere ist die Weglänge zwischen dem Ein- und Ausgangspunkt des Strahls in das Voxel hinein und aus dem Voxel heraus.

$$n(E) = n_0(E) \cdot e^{-\mu_{\text{abs}}(E) \cdot c_v} \quad (3.45)$$

Wie in Abschnitt 3.3.1 (S. 51) beschrieben, ist für einen Strahl die spektrale Zusammensetzung der Photonenflüsse gegeben. Für jede diskrete Energie wird der Absorptionskoeffizient aus dem für das Voxel gespeicherten Zahlenwert  $Z_g$  nach Gleichung 3.30 (S. 60) berechnet. Mit der Strecke  $c_v$ , die aus der Strahlverfolgung bekannt ist, kann somit für jede Energie der Photonenfluss nach Durchgang des Voxels bestimmt werden. In Listing A.15 ist die Funktion, die das Spektrum entsprechend der Voxel-eigenschaften schwächt, dargestellt.

Um Strahlaufhärtungssartefakte deaktivieren zu können (PHY-025) wird neben der spektralen Zusammensetzung der Photonenflüsse eine *vereinfachte Intensität* einem Strahl zugeordnet. Bei dieser handelt es sich um einen einzelnen Zahlenwert, der bei der Strahlerzeugung mit dem Wert 1 initialisiert wird und nach dem energieunabhängigen Schwächungsgesetz bei einem Voxeldurchgang verringert wird. Dieser Wert repräsentiert den Fall, dass die Strahlung monoenergetisch wäre.

### 3.5.3. Compton-Streuung

Im Simulator ist als einziger Streuprozess die Compton-Streuung abgebildet. Idealerweise würden Streuprozesse durch Betrachtung einzelner Photonen, deren Wechselwirkung mit einzelnen Elektronen probabilistisch modelliert sind, simuliert werden. Wegen der hohen Anzahl von Photonen und damit auch notwendigen einzelnen Berechnungszyklen, werden Photonen zu Mengen zusammengefasst. Statt für jedes einzelne Photon, wird für jede dieser Photonenmengen die Wechselwirkung mit Elektronen simuliert. Da auch die Elektronenanzahl groß ist, können keine Wechselwirkungen mit einzelnen Elektronen betrachtet werden. Stattdessen wird die Anzahl der Streuereignisse für einen Strahl auf ein Streuereignis pro Voxel und Photonenmenge begrenzt. Um das Streuverhalten abzubilden, muss dann lediglich die Wechselwirkungswahrscheinlichkeit einer Photonenmenge mit einem Voxel bestimmt werden.

Die Funktion in Listing A.16 stellt die Simulation der Streuung dar. Sie wird einmalig für jeden Voxel, der getroffen wird, aufgerufen. Der grobe Programmablauf, der die Streusimulation beschreibt, ist in Abbildung 3.20 abgebildet. Jede Energie  $E$  von insgesamt  $N_E$  diskreten Energien im Spektrum wird in  $N_{PM}$  Teile aufgeteilt. Diese Teile entsprechen Photonenmengen mit gleicher Energie. Für jeden Teil jeder Energie wird die Streuwahrscheinlichkeit in einem Voxel berechnet. Für jeden Strahl können also maximal  $N_E \cdot N_{PM}$  Streuereignisse auftreten. Kommt es zur Streuung für eine der Photonenmengen, wird ein zufälliger Streuwinkel bestimmt. Nachdem für jede der  $N_E \cdot N_{PM}$  Photonenmengen bestimmt wurde, ob und unter welchem Winkel ein Streuereignis stattfindet, werden für gleiche Winkel die Energien in dem Spektrum eines neuen Strahls zusammengefasst.

#### Bestimmung der Streuwahrscheinlichkeit

Zur Bestimmung der Streuwahrscheinlichkeit bei Durchgang durch ein Voxel, wird Gleichung 2.28 (S. 13), die die Wahrscheinlichkeit in Abhängigkeit von der Flächendichte  $d$  und dem Massenkoeffizienten  $\mu_C$  ausdrückt, genutzt. Die Flächendichte ist durch die Weglänge  $x$ , die durch die Strahlverfolgung als  $c_v$  bekannt ist und die Dichte  $\rho$  gegeben. Der Massenschwächungskoeffizient  $\mu_C$  ist nach Gleichung 2.26 (S. 13) abhängig von dem Wechselwirkungsquerschnitt  $\sigma_K$  und der Elektronendichte je Masse  $N_{EI}$  eines Materials. Wird statt der Elektronendichte je Masse die Elektronendichte je Volumen eingesetzt, kürzt sich die Dichte heraus. Damit lässt sich die Streuwahrscheinlichkeit als Gleichung 3.46 beschreiben.

$$p_{WW}(x) = 1 - e^{-\sigma_K \cdot N_{EI} \cdot x} \quad (3.46)$$

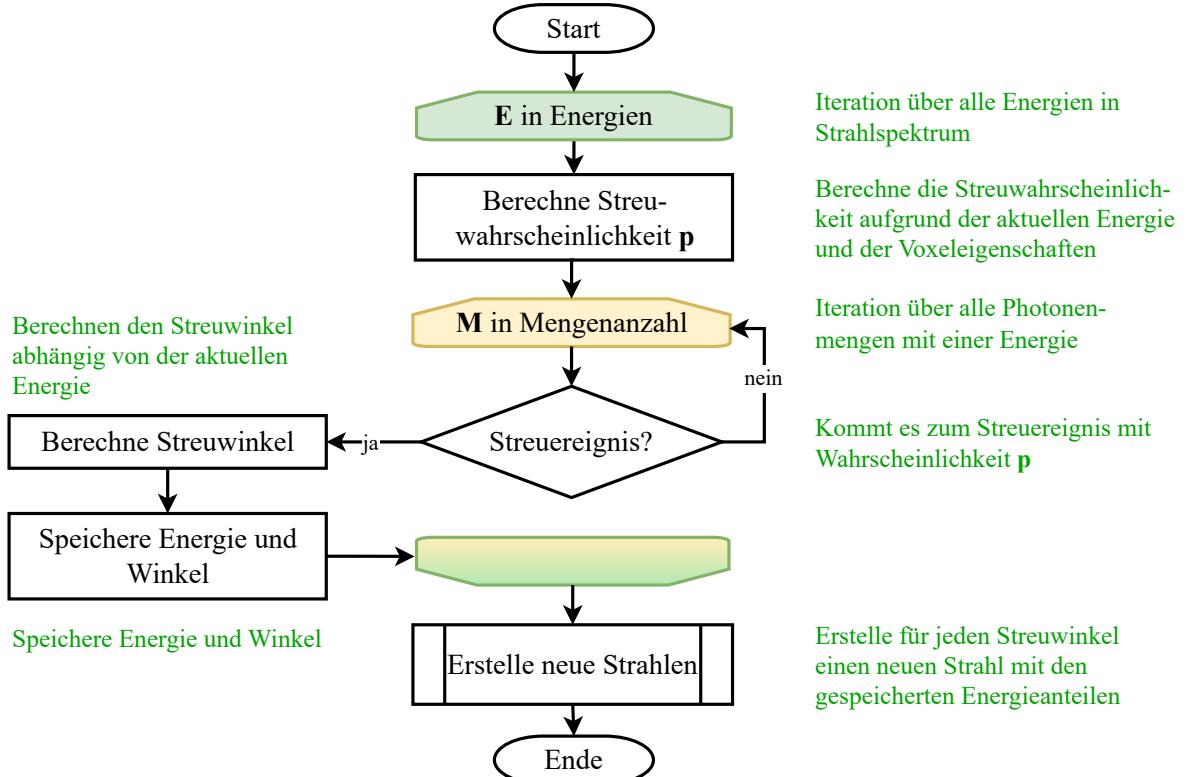


Abbildung 3.20.: Programmablaufplan zum Streualgorithmus

Zur Abschätzung der Elektronendichte innerhalb des Voxels steht nur der in den Voxeleigenschaften gespeicherte lineare Absorptionskoeffizient  $\mu_{\text{abs}}$  des aktuellen Voxels zur Verfügung. Der Absorptionskoeffizient setzt sich, wie in Abschnitt 2.1.1 auf Seite 14 beschrieben, primär aus Beiträgen des Photoeffektes und der Compton-Absorption zusammen. Für den Photoeffekt gilt, dass der lineare Absorptionskoeffizient von der Atomanzahl je Volumen  $N_{\text{At}}$  und der Ordnungszahl  $Z$  abhängt [10, S. 839]. Da das Produkt beider Größen der Elektronendichte je Volumen entspricht, hängt auch der Absorptionskoeffizient des Photoeffektes linear von der Elektronendichte ab. Ebenso verhält es sich mit linearen Absorptionskoeffizienten der Compton-Absorption [11, S. 271]. Daher wird vereinfacht angenommen, dass der gesamte Absorptionskoeffizient linear von der Elektronendichte abhängt. Für einen Voxel, dessen Absorptionskoeffizient bekannt ist, wird die Elektronendichte über diesen Zusammenhang festgelegt. In Gleichung 3.47 wird der Absorptionskoeffizient des Voxels mit jenem von Wasser ins Verhältnis gesetzt und mit der Elektronendichte von Wasser multipliziert, um die Elektronendichte des Voxels zu bestimmen.

$$N_{\text{El}} = N_{\text{El}, \text{Wasser}} \cdot \frac{\mu_{\text{abs}}}{\mu_{\text{abs}, \text{Wasser}}} \quad (3.47)$$

Die Elektronendichte von Wasser ist gegeben durch die Elektronenanzahl je Molekül<sup>3</sup>, der Dichte ( $0,998 \frac{\text{g}}{\text{cm}^3}$ ), der Avogadrokonstante und der molaren Masse ( $18 \frac{\text{g}}{\text{mol}}$ ). Der Absorpti-

<sup>3</sup>10, da 2 Wasserstoffatome ( $Z = 1$ ) und 1 Sauerstoffatom ( $Z = 8$ )

onskoeffizient von Wasser bei 100 keV ist aus Abbildung 2.4 (15) ablesbar.

$$N_{\text{El, Wasser}} = 10 \cdot 0,998 \frac{\text{g}}{\text{cm}^3} \cdot 6,022 \cdot 10^{23} \frac{1}{\text{mol}} \left( 18 \frac{\text{g}}{\text{mol}} \right)^{-1} \quad (3.48)$$

$$= 3,3389 \cdot 10^{20} \frac{1}{\text{mm}^3} \quad (3.49)$$

$$\mu_{\text{abs, Wasser}} = 0,003 \frac{1}{\text{mm}} \quad \text{bei } 100 \text{ keV} \quad (3.50)$$

Zur Berechnung des Wirkungsquerschnittes  $\sigma_K$  wird Gleichung 2.24 (12) implementiert. Da es sich um viele arithmetische Operationen, die für eine Berechnung ausgeführt werden müssen, handelt, werden verschiedene Wirkungsquerschnitte vorberechnet. Der Wirkungsquerschnitt ist nur von der Photonenergie abhängig, sodass für eine Anzahl von Energien  $N_E$  in bestimmten Abständen  $\Delta E$  die Querschnitte  $\sigma_K(E)$  berechnet werden. Die Energieanzahl ergibt sich aus einer gewünschten Auflösung und dem Energiebereich der Röntgenröhre. Durch die Vorberechnung muss nur der Index  $i_E$  des Wirkungsquerschnittes berechnet werden. Die Indizes der Querschnitte, die Energien zugeordnet sind, können einfach berechnet werden. Voraussetzung dafür ist, dass die Energien mit gleichen Abständen zueinander, streng monoton steigend und kontinuierlich vorliegen. Der Index einer beliebigen Energie  $E$  lässt sich somit durch Gleichung 3.51 bestimmen.

$$i_E = \left\lfloor \frac{E - E_{\min}}{\Delta E} + 0,5 \right\rfloor \quad (3.51)$$

In der Implementierung ist die Streuwahrscheinlichkeit zusätzlich mit einem Faktor zwischen 0 und 1 skaliert, um die Streuintensität (PHY-026) einstellen zu können.

### Zufallszahlen

Damit aus Wahrscheinlichkeiten zufällige Ereignisse generiert werden können, müssen Zufallszahlen erzeugt werden. Der C++-11-Standard definiert einen Zahlengenerator, der pseudozufällige<sup>4</sup> ganze Zahlen in einem vorgegebenen Intervall  $[G_{\min}, G_{\max}]$  gleichverteilt (`uniform_int_distribution`) erzeugt [30, S. 1083]. Gleichverteilt bedeutet, dass bei vielen Aufrufen des Generators jede ganze Zahl im Intervall die gleiche Wahrscheinlichkeit hat, erzeugt zu werden. Um für ein Ereignis, das mit der Wahrscheinlichkeit  $p(\text{Ereignis})$  eintritt, festzulegen, ob es eintritt, muss überprüft werden, ob eine zufällig generierte Zahl in einem bestimmten Intervall liegt. Die zugrunde liegende Gleichverteilung ordnet jeder ganzen Zahl  $G$  im Intervall  $[G_{\min}, G_{\max}]$  jeweils eine Wahrscheinlichkeit generiert zu werden, zu. Diese beträgt  $p_G = 100\%/(G_{\max} - G_{\min} + 1)$  [30, S. 1083]. Die Wahrscheinlichkeit, dass eine zufällig generierte Zahl  $G$  in einem Intervall, das  $N_G$  Zahlen enthält, liegt, ist mit der Wahrscheinlichkeit  $p(G \in [G_0, G_0 + N_G - 1])$  gegeben.

$$p(G \in [G_0, G_0 + N_G - 1]) = N_G \cdot p_G = \frac{N_G}{G_{\max} - G_{\min} + 1} \quad (3.52)$$

$$\text{mit: } G_0 \geq G_{\min} \quad \text{und} \quad G_0 + N_G - 1 \leq G_{\max} \quad (3.53)$$

---

<sup>4</sup>Die generierten Zahlen erscheinen zufällig, verhalten sich jedoch deterministisch

Liegt die generierte Zahl  $G$  in dem Intervall mit der Größe  $N_G$ , das sich aus der Ereigniswahrscheinlichkeit  $p(\text{Ereignis})$  ergibt, tritt das Ereignis ein. Wo genau das Intervall in der Gleichverteilung liegt, spielt dabei keine Rolle, da die Wahrscheinlichkeiten über das vollständige Intervall konstant sind. Die Abrundungsfunktion  $\lfloor \cdot \rfloor$  wird implizit bei der Typkonvertierung<sup>5</sup> zur ganzen Zahl genutzt.

$$N_G = \lfloor p(\text{Ereignis}) \cdot (G_{\max} - G_{\min} + 1) \rfloor \quad (3.54)$$

$$\text{Ereignis tritt ein, wenn: } G_{\min} \leq G < G_{\min} + N_G \quad (3.55)$$

Dabei sollte das Intervall der gesamten Verteilung  $[G_{\min}, G_{\max}]$  so groß sein, dass auch kleine Wahrscheinlichkeiten abgebildet werden können. Ist beispielsweise das Intervall durch  $G_{\min} = 0$  und  $G_{\max} = 9$  gegeben, können wegen der Abrundungsfunktion nur Wahrscheinlichkeiten in 10 % Schritten aufgelöst werden.<sup>6</sup> In Listing A.18 ist die Generierung von zufälligen Ereignissen mit einer gegebenen Wahrscheinlichkeit implementiert.

$$\begin{aligned} \lfloor p \cdot (9 - 0 + 1) \rfloor &= 0 & \text{für: } 0 \% \leq p < 10 \% \\ \lfloor p \cdot (9 - 0 + 1) \rfloor &= 1 & \text{für: } 10 \% \leq p < 20 \% \\ &\vdots & & \vdots \end{aligned}$$

Bei der Initialisierung eines Zufallszahlengenerators, der pseudozufällige Zahlen erzeugt, wird ein Parameter, der als *seed* bezeichnet wird, genutzt. Da diese Generatoren deterministisch sind, ist die Abfolge der Zufallszahlen durch den Anfangszustand vollständig bestimmt [29, S. 253]. In der Implementierung wird bei Programmstart als seed der aktuelle UNIX-Zeitstempel in Sekunden gewählt. Somit ist das Ergebnis mehrerer Simulationen mit identischen Parametern unterschiedlich.

Die Generatoren der C++-Standardbibliothek benötigen zur Generation von Zufallszahlen sehr viel Rechenzeit. Je Voxel muss mindestens eine zufällige ganze Zahl generiert werden. In Listing A.17 ist der Zufallsgenerator *xoroshiro128\*\** [31] mit dem Quelltext von [32] implementiert. Es stellt sich heraus, dass dieser Zufallsgenerator wesentlich weniger Rechenzeit beansprucht und somit die Simulationsgeschwindigkeit erhöht.

### **Beliebige Verteilungen**

Neben gleichverteilten Zufallszahlen definiert der C++-11-Standard Verteilungen mit beliebigen Wahrscheinlichkeiten (*discrete\_distribution*). Generiert werden ganze Zahlen  $G \in [0, N]$  jeweils mit der Wahrscheinlichkeit  $p_G$ . Die Wahrscheinlichkeiten werden bei der Initialisierung der Verteilung anhand gegebener Gewichte  $w_G$  nach Gleichung 3.56 berechnet.

---

<sup>5</sup>Die Wahrscheinlichkeit liegt als `double` vor. Der fraktionelle Anteil wird bei der Konvertierung zu einer ganzen Zahl abgeschnitten.

<sup>6</sup>In der Implementierung gibt der Zufallsgenerator Zahlen zwischen 0 und  $2^{32} - 1$  aus.

[30, S. 1097]

$$p_G = \frac{w_G}{\sum_{k=0}^{N-1} w_k} \quad (3.56)$$

Da mit dieser Verteilung nur ganze Zahlen generiert werden können, müssen, um reelle Zufallszahlen zu erzeugen, die Zufallszahlen  $G$  auf eine reelle Reihe  $V_G$  abgebildet werden. Die Zufallszahl  $G$  stellt dabei den Index des Elementes in der Reihe dar.

Da dieser Zufallsgenerator aus der C++-Standardbibliothek ebenfalls wie jener, der eine Gleichverteilung abbildet, viel Rechenzeit beansprucht, wird auch hier alternativ vorgegangen. Die ganzen Zahl  $G$  werden mit ihrer Wahrscheinlichkeit  $p_G$  erzeugt, indem ein Array mit den ganzen Zahlen  $G$  gefüllt wird. Insgesamt hat das Array eine Größe von  $N_{\text{Array}}$ . Die Anzahl  $A_G$ , wie häufig  $G$  in dem Array vorkommt, ist durch ihre Wahrscheinlichkeit durch Gleichung 3.57 gegeben.

$$A_G = p_G \cdot N_{\text{Array}} \quad (3.57)$$

Eine zufällig generierte, gleichverteilte, ganze Zahl kann so als Index für das Array genutzt werden, um eine beliebige Verteilung abzubilden.

### Streuinkel

Der Streuwinkel  $\vartheta$  beschreibt den Winkel zwischen der Trajektorie des ankommenden Strahls zum gestreuten Strahl. Kommt es zu einem Streuereignis soll der Streuwinkel entsprechend seiner Verteilung aus Abbildung 2.3 bestimmt werden. Diese Verteilung beruht auf dem differentiellen Wirkungsquerschnitt aus Gleichung 2.23 (S. 12). Es wird angenommen, dass der differentielle Wirkungsquerschnitt  $\frac{d\sigma_K}{d\Omega}(\vartheta)$  ein Maß für die Wahrscheinlichkeit  $p(\vartheta = \Theta)$ , dass der Streuwinkel bei einem Streuereignis  $\Theta$  beträgt, ist.

$$p(\vartheta = \Theta) = f \cdot \frac{d\sigma_K}{d\Omega}(\Theta) \quad (3.58)$$

$$\text{mit: } f = \left( \sum_{i=1}^{N_\Theta} \frac{d\sigma_K}{d\Omega}(\Theta_i) \right)^{-1} \quad (3.59)$$

Bei  $\Theta_i$  handelt es sich um diskrete Streuwinkel, die in der Implementierung zwischen  $-\pi$  und  $\pi$  liegen können. Der Faktor  $f$  wird so gewählt, dass die Summe aller Wahrscheinlichkeiten gleich eins ist. Die Anzahl der Streuwinkel  $N_\Theta$  ist ungerade, damit 0 rad bei  $i = (N_\Theta - 1)/2$  als Winkel möglich ist. Da sowohl negative als auch positive Winkel möglich und die Winkel diskret sind, muss die Wahrscheinlichkeit für den Winkel 0 rad verdoppelt werden, da er sonst unterrepräsentiert wäre<sup>7</sup>. Analog zu den vorberechneten Wirkungsquerschnitten einer Kollision werden für verschiedene Energien die Wahrscheinlichkeiten der jeweiligen Streuwinkel berechnet. In Listing A.19 ist die Vorberechnung der Streuwinkelwahrscheinlichkeiten abgebildet. Die Verteilungen der Wahrscheinlichkeiten werden in einer Verteilung, wie sie im vorangegangenen Abschnitt beschrieben wurde, gespeichert.

---

<sup>7</sup>Alternativ könnte nur ein Winkel in  $[0, \pi]$  sowie das Vorzeichen mit je 50 % bestimmt werden.

Wurde für ein Streuereignis ein zufälliger Winkel bestimmt, wird die neue Energie der Photonen nach Gleichung 2.32 (S. 14) bestimmt. Das Spektrum der gestreuten Strahlen wird für jeden Streuwinkel, der mindestens einmal pro Strahl und Voxel generiert wurde, aus diesen neuen Energien mit den entsprechenden Photonенflüssen zusammengesetzt.

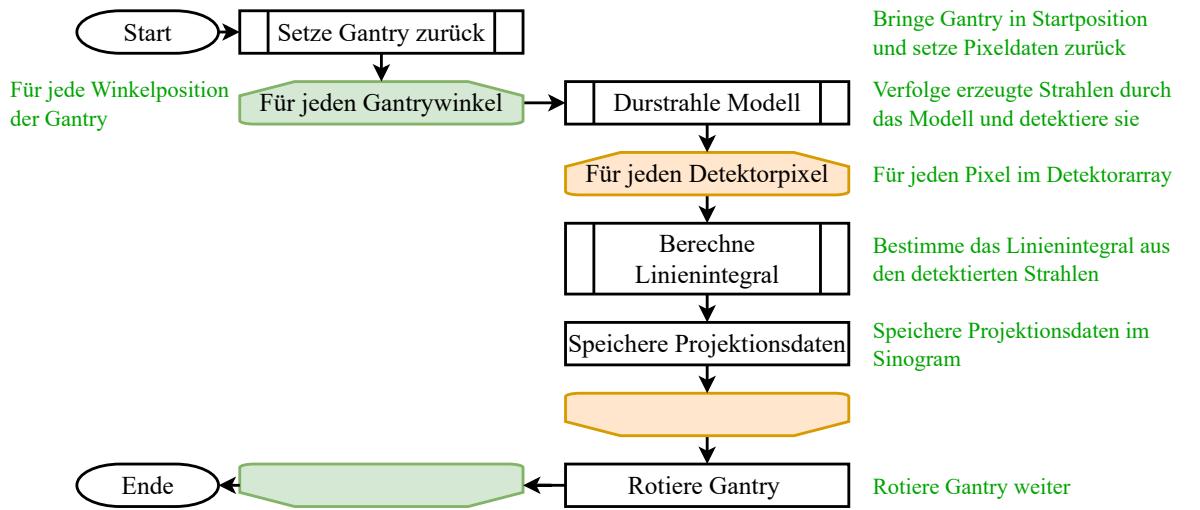
Die bisherige Betrachtung der Streuwinkel ist rein zweidimensional. Es wurde der Streuwinkel, der in der Schnittebene liegt, beschrieben. Physikalisch gesehen bildet ein Streuwinkel jedoch einen dreidimensionalen Kegel um die Trajektorie des an kommenden Strahls herum. Um das in der Implementierung zu berücksichtigen, werden zwei senkrecht aufeinander stehende Winkel in Kugelkoordinaten genutzt. Einer dieser Winkel liegt in der Schnittbildebene und stellt den Streuwinkel dar. Der zweite Winkel ist jener, zwischen dem gestreuten Strahl und dieser Schnittbildebene. Gestreute Strahlen, dessen Winkel zur Schnittbildebene zu groß ist, können das Pixelarray nicht erreichen. Diese Photonen müssen nicht verfolgt und können daher verworfen werden. Die Bedingung für das Verwerfen ist das Überschreiten eines zufällig generierten zweiten Streuwinkels um einen bestimmten Winkel. Dieser Winkel entspricht jenem zwischen den Pixelrändern, die parallel zur Schnittbildebene liegen, betrachtet vom Mittelpunkt der Gantry. Bei diesem Vorgehen handelt es sich um eine Vereinfachung der physikalischen und technischen Modelle. Es wird jedoch verhindert, dass Strahlen unnötig verfolgt werden.

### 3.6. Aufnahme der Projektionen

In den vorangegangenen Abschnitten wurde erläutert, wie die physikalischen Modelle, die den Durchgang von Strahlung durch Materie beschreiben, implementiert wurden. Außerdem wurde die Implementierung der Gerätekomponenten beschrieben. Zur Aufnahme von Projektionsdaten, die die Rekonstruktion eines Schnittbildes ermöglichen, fehlt die Beschreibung der Implementierung des Prozesses, der durch Rotation der Gantry und Verfolgung sowie Detektion der jeweiligen Strahlen, die Projektionsdaten sammelt.

In Listing A.20 ist die Implementierung der Funktion, die die Aufnahme eines Schnittbildes durchführt, zu sehen. In Abbildung 3.21 ist der dazu passende Programmablaufplan dargestellt. Zu Beginn wird die Gantry in eine bekannte Startposition gebracht und zur Auswahl der Schnittebene entlang der  $z$ -Achse verschoben. Außerdem werden die Daten der Detektorpixel zurückgesetzt, da von vorherigen Aufnahmen noch Daten vorliegen könnten. Um einen vollständigen Projektionsdatensatz zur Rekonstruktion zu erhalten, müssen für verschiedene Winkelposition der Gantry die Projektionen gemessen werden. In Gleichung 3.19 auf Seite 56 ist die Anzahl der Winkelpositionen  $N_{\text{Rotationen}}$ , die sich jeweils um  $\Delta\vartheta$  unterscheiden, beschrieben. Für jede dieser Position müssen die Strahlen durch das Körpermodell verfolgt werden, um alle Datenpunkte im Sinogramm zu füllen. Aus den Strahleigenschaften, die von den Pixeln gespeichert werden und der Pixelposition in Relation zu der Radonebene, werden die Werte der Datenpunkte im Sinogramm bestimmt. Die Radonebene ist eine Ebene, die die Rotationsachse als Normale hat und nicht mit rotiert. Auf diese beziehen sich die

Radon-Koordinaten<sup>8</sup>.



**Abbildung 3.21.:** Programmablaufplan zum Sammeln der Projektionsdaten für ein Schnittbild

### 3.6.1. Verfolgung der Strahlen

Für eine Winkelposition der Gantry werden die Strahlen erzeugt, durch das Modell verfolgt und detektiert. Die Implementierung dieses Ablaufes ist in der Funktion aus Listing A.21 dargestellt. Zu Beginn werden die Strahlen wie in Abschnitt 3.3.1 (S. 51) beschrieben, erzeugt. Die Strahlen und die Detektorpixel werden daraufhin in das Koordinatensystem des Körpermodells überführt, da in diesem die Strahlverfolgung stattfindet.

Die initialen Strahlen sind jene mit der Röntgenröhre als Ursprung und werden als Strahlen erster Ordnung bezeichnet. Da es bei der Strahlverfolgung durch Streuung zur Erzeugung neuer Strahlen kommen kann, müssen diese ebenfalls verfolgt und detektiert werden. Strahlen, die durch Streuung von Strahlen erster Ordnung entstehen, sind Strahlen zweiter Ordnung. Dieses Schema ist bis beliebige Ordnungen skalierbar. In der ersten Iteration werden nur die Strahlen erster Ordnung verfolgt und detektiert. Alle Strahlen, die dabei neu entstehen, werden gespeichert. In der nächsten Iteration werden dann diese Strahlen zweiter Ordnung durch das Modell verfolgt. Da so prinzipiell sehr hohe Ordnungen möglich sind, gibt es ein Abbruchkriterium. Dieses Kriterium ist eine obere Grenze für die Ordnung der Strahlen. Sie entspricht der Anzahl wie häufig ein Strahl gestreut werden kann. Wird diese Grenze erreicht, werden alle Strahlen, die noch nicht detektiert wurden, durch das Modell verfolgt und detektiert. Die Streuung ist dabei jedoch deaktiviert, sodass keine weiteren neuen Strahlen entstehen können.

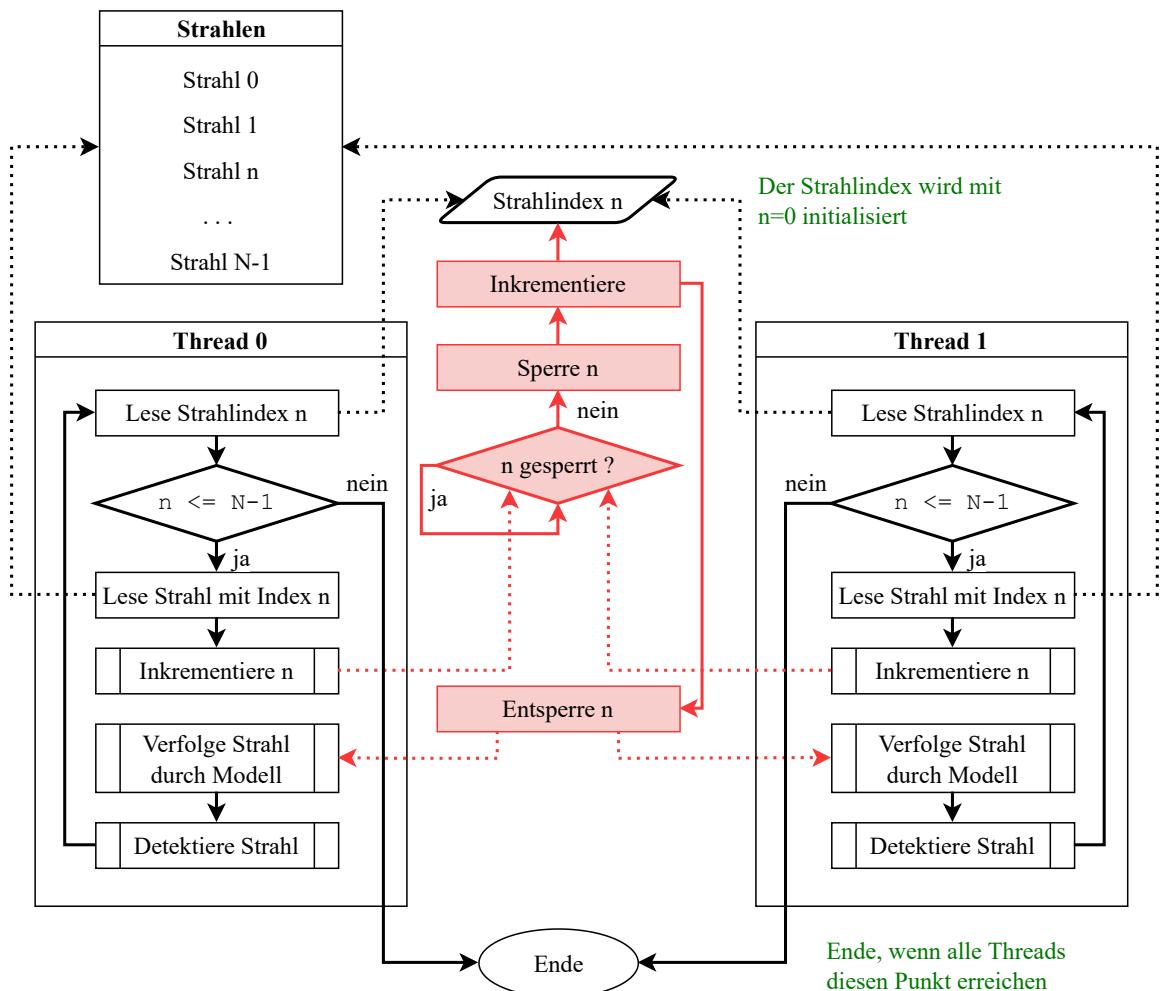
Zusätzlich werden Strahlen, die in der vorletzten Iteration durch Streuung erzeugt werden, daraufhin überprüft, ob diese einen Detektorpixel treffen können. Wenn ja, wird der Pixelindex gespeichert, um die Detektion zu beschleunigen. Kann kein Pixel getroffen werden, ist es unmöglich, dass der Strahl in irgendeiner Art und Weise einen Einfluss auf die gemessenen

<sup>8</sup>Diese Koordinaten sind jene Polarkoordinaten  $(s, \theta)$ , durch die die Strahlrichtung definiert sind.

Projektionen hat. Grund dafür ist, dass in der letzten Iteration keine neuen Strahlen, die von diesem ausgehen, erzeugt werden. Diese Strahlen werden verworfen, um Rechenzeit zu sparen.

### Parallele Berechnung

Heutige Computer bieten die Möglichkeit, parallel Berechnungen durchzuführen (ALG-005). Dadurch lassen sich mehrere Strahlen simultan durch das Körpermodell verfolgen. Parallellaufende Programmteile werden jeweils als *Thread* bezeichnet. Die maximale Anzahl der Threads ist abhängig von der verwendeten Hardware und aktuell nutzbaren Ressourcen, die durch das Betriebssystem verwaltet werden. Ein Thread hat jeweils die Aufgabe Strahlen nacheinander zu verfolgen und zu detektieren, bis alle Strahlen genau einmal behandelt wurde. Wurden alle Strahlen behandelt, was durch die Beendigung aller Threads angezeigt wird, werden die Strahlen der nächsthöheren Ordnung verfolgt. Die Funktion, die innerhalb jedes Threads ausgeführt wird, ist in Listing A.22 implementiert. Der Programmablauf innerhalb eines Threads und die Interaktion mit einem anderen Thread ist in Abbildung 3.22 zu sehen. Alle Strahlen, die durch das Körpermodell verfolgt werden sollen, liegen als Array, aus dem



**Abbildung 3.22.:** Gleichzeitige Strahlverfolgung mit durch Multithreading. Zwei Threads, die auf gemeinsame Ressourcen zugreifen. Bei Schreibvorgängen müssen Ressourcen gesperrt werden (Rot)

alle Threads lesen können, vor. Der Strahlindex gibt vom ersten bis zum letzten Strahl in diesem Array an, welcher Strahl als nächstes verfolgt werden soll. Jeder Thread liest den aktuellen Index und verfolgt den entsprechenden Strahl durch das Körpermodell. Nach dem Lesen des Index wird dieser inkrementiert. Der verfolgte Strahl wird detektiert und mit dem nächsten Strahl fortgefahren.

Da verschiedene Threads auf geteilte Ressourcen zugreifen, müssen Maßnahmen ergriffen werden, um gleichzeitigen Zugriff und daraus resultierende Fehler zu verhindern. Ein simultaner Zugriff auf gleiche Ressourcen kann zu Wettlaufsituationen, die auch als *race condition* bezeichnet werden, führen [33, S. 33]. Ohne schützende Maßnahmen kann es zu einer Situation kommen, in der zum Beispiel ein Thread den Wert des aktuellen Strahlindex liest, um ihn zu inkrementieren. Wenn jedoch im Zeitraum zwischen Lesen des Wertes und Schreiben des inkrementierten Wertes ein anderer Thread den Wert ebenfalls liest, wird dieser den Wert lokal inkrementieren, sodass schlussendlich beide Threads denselben inkrementierten Wert schreiben und den gleichen Strahl verfolgen würden. Um solch einen Fall auszuschließen, werden *wechselseitige Ausschlüsse* genutzt. Deren Prinzip ist in Abbildung 3.22 in Rot dargestellt. Bevor auf die geteilte Ressource zugegriffen wird, wird diese gesperrt. Andere Threads, die während die Ressource gesperrt ist auf diese zugreifen, warten bis sie entsperrt ist. Wenn die Ressource gesperrt wurde, kann die Operation durchgeführt werden und sie entsperrt werden. In der Implementierung sind sowohl das Lesen des Strahlindex als auch das Inkrementieren durch diesen wechselseitigen Ausschluss geschützt. Der Zugriff auf das Array mit Strahlen oder das Körpermodell müssen nicht geschützt werden, da auf diese Ressourcen nur lesend zugegriffen wird. Der Zugriff auf die Detektorpixel jedoch muss geschützt werden, da in diesen die detektierten Strahleigenschaften gespeichert sind. Der C++-Standard definiert die Klasse `mutex` [30, S. 1363], die für den wechselseitigen Ausschluss (engl. *mutual exclusion*) genutzt wird. Diese Klasse stellt die Methoden zum Sperren und Entsperrern zur Verfügung. Auf eine Instanz dieser Klasse haben alle Threads Zugriff. Ist eine Instanz gesperrt, werden andere Threads, die zugreifen möchten, blockiert, bis die Instanz entsperrt ist.

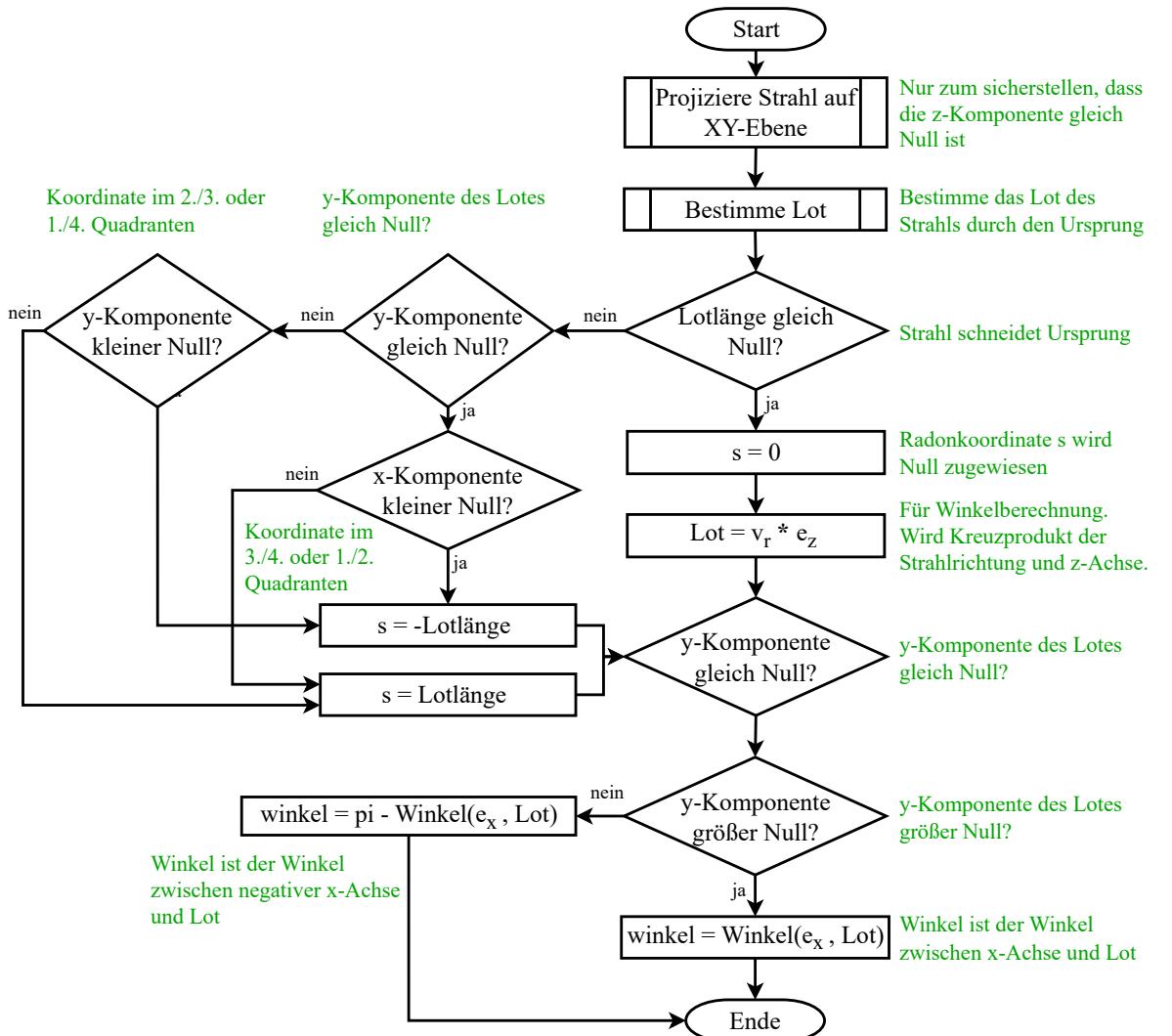
Wenn jeder Thread beendet wurde, bedeutet das, dass alle Strahlen verfolgt wurden und die Projektionsdaten berechnet werden können.

### 3.6.2. Sammeln der Projektionsdaten

Für jede Winkelposition der Gantry werden nach Verfolgung und Detektion der Strahlen die Projektionsdaten berechnet und in einem Raster gespeichert. Das Raster ist das Sinogramm, welches in Abbildung 3.11 (S. 54) zu sehen ist. Jeder Datenpunkt ist durch seine Position, welche den Radon-Koordinaten entspricht, und seinen Wert, der dem Linienintegral entspricht, charakterisiert.

Die Radon-Koordinaten sind die Länge  $s$  des Lotes zur Geraden  $\ell$  durch den Ursprung und der Winkel  $\theta$  zwischen Lot und  $x$ -Achse. Die Gerade  $\ell$  ist jene, die senkrecht auf einem Pixel steht und seine Mitte schneidet. In Abbildung 2.5 (S. 17) sind diese Koordinaten dargestellt. Deren Berechnung aus einer Instanz der Klasse `Line` ist in Listing A.23 abgebildet. Der

dazugehörige Programmablaufplan ist in Abbildung 3.23 dargestellt. Da es sich bei den Radon-Koordinaten um solche handelt, die in einer Ebene definiert sind, wird durch die Projektion der Geraden auf die  $xy$ -Ebene sichergestellt, dass diese über keine  $z$ -Komponente verfügt. Die  $xy$ -Ebene, auf die projiziert wird, ist jene, die zu einem gewählten Koordinatensystem gehört. Es wird das Koordinatensystem der Gantry genutzt, da dessen Ursprung der Rotationsmittelpunkt ist und die Pixel so konstruiert sind, dass deren Normalen in der  $xy$ -Ebene liegen. Zur Bestimmung des Lotes zu einer Geraden wird Gleichung 3.15 (S. 50) genutzt. Ist die Länge des Lotes gleich Null, ist impliziert, dass die Gerade den Ursprung schneidet und auch  $s = 0$  ist. In diesem Fall wird, damit der Winkel bestimmt werden kann, das Lot als Kreuzprodukt des Richtungsvektors der Geraden und dem Einheitsvektor der  $z$ -Achse festgelegt. In jedem anderen Fall werden jeweils die  $x$ - und  $y$ -Komponenten überprüft, sodass nach dem Ordnungsschema auf Seite 16  $s$  und  $\theta$  berechnet werden.



**Abbildung 3.23.:** Programmablaufplan zur Bestimmung der Radon-Koordinaten aus einer Geraden

Neben den Radon-Koordinaten muss der Wert des Datenpunktes bestimmt werden. Der Wert ist das Linienintegral  $p(\theta, s)$  aus Gleichung 2.39 (S. 17). Nach Gleichung 2.91 (S. 29) ist das Linienintegral durch den natürlichen Logarithmus des Verhältnisses der Anfangs- zur

Endintensität gegeben. Es ist möglich, dass mehrere Strahlen auf einen Pixel fallen. Ist das der Fall, werden deren Intensitäten jeweils addiert. Gleichung 3.60 beschreibt die Berechnung des Projektionswertes auf Grundlage der Strahlengesamtleistungen. In Listing A.24 ist diese Berechnung implementiert.

$$p_\theta(s) = \ln\left(\frac{J_0}{J}\right) = \ln\left(\frac{\sum_{r=1}^{N_r} J_{0,r}}{\sum_{r=1}^{N_r} J_r}\right) \quad (3.60)$$

Die Addition der Intensitäten einzelner Strahlen ist möglich, da dies einer Addition der Spektren entspricht. Die Intensität eines Strahls ist durch Akkumulation der Produkte aus den Energien  $E$  und den Photonenflüssen dieser Energien  $n(E)$  nach Gleichung 3.61 gegeben.

$$J_{\text{Strahl}} = \sum_{i=1}^{N_E} E_i \cdot n(E_i) \quad (3.61)$$

## 3.7. Rekonstruktion der Schnittbilder

Nach Aufnahme der Projektionen liegen die Projektionsdaten geordnet in einem Raster, das in Abbildung 3.11 (S. 54) dargestellt ist, vor. Für die Bildrekonstruktion müssen die Projektionsdaten gefiltert und rückprojiziert werden (SIG-000).

### Begrenzung des Wertebereiches

Detektorpixel sind in der Realität nicht in der Lage beliebig kleine Signale aufzulösen oder beliebig große Signale abzubilden. Sie sind in ihrer Sensitivität und ihrem Messbereich begrenzt. Das wird qualitativ dadurch im Simulator abgebildet, dass der Wertebereich der Projektionsdaten begrenzt werden kann. Eine untere Schranke  $p_{\min}$  für die Projektionsdaten ist äquivalent zu einer unteren Schranke des Verhältnis von  $J_0$  zu  $J$ . Wenn also die detektierte Intensität  $J$  größer als  $J_0 \cdot (e^{p_{\min}})^{-1}$  ist, kommt es zur Sättigung. Analog dazu ist mit  $p_{\max}$  eine obere Schranke gegeben, die die begrenzte Sensitivität abbildet. Ist  $J$  kleiner als  $J_0 \cdot (e^{p_{\max}})^{-1}$  können die Intensitäten nicht voneinander unterschieden werden.

$$p_{\min} \leq p_\theta(s) \leq p_{\max} \Rightarrow p_{\min} \leq \ln\left(\frac{J_0}{J_{\theta,s}}\right) \leq p_{\max} \quad (3.62)$$

Standardmäßig ist  $p_{\min} = 0$ , da  $J_{\theta,s}$  nicht größer als  $J_0$  sein kann. Die Ausnahme bilden Fälle, in denen gestreute Strahlen auf einen Pixel treffen und die detektierte Intensität größer als  $J_0$  wird.

### Filterung

Die diskrete Filterung, die in Abschnitt 2.2.3 (S. 25) beschrieben wurde, ist durch die Faltung eines Filterkerns mit den Projektionsdaten entlang der Abstände  $s$  für jeden Winkel  $\theta$  im Ortsraum implementiert. Gleichung 2.83 (S. 27), die diese Faltung darstellt, ist durch Listing A.25 im Simulator abgebildet. Genauer ist es Gleichung 3.63, die die Faltung von diskreten Werten abbildet. Der Index  $k$  ist der Index des Winkels  $\theta = k \cdot \Delta\theta$ . Der Index  $l$  ist der Index

des Abstandes der mit  $s = \left(l - \frac{N_s}{2} + 0,5\right) \cdot \Delta s$  gegeben ist.

$$\tilde{p}_k[l] = \sum_{m=l-\frac{N_s}{2}}^{l+\frac{N_s}{2}} p_k[l-m] \cdot h[m] \cdot \Delta s \quad (3.63)$$

mit:  $l = 0, 1, \dots, N_s - 1$

$$p_k[l-m] = 0, \text{ wenn } l-m \notin [0, N_s - 1]$$

Um die gefilterten Projektionen vollständig zu berechnen, wird über jeden Projektionswinkel  $\theta$  iteriert. Für jeden dieser Winkel wird die Faltung des Filterkerns mit den Projektionswerten, die zu diesem Winkel gehören, durchgeführt. Das entspricht der Filterung einer Projektion bei konstantem Winkel. Die Faltung ist so implementiert, dass für jeden Abstand  $s$  mit dem Index  $l$  der Wert des Filterkerns beim Index  $m$  mit dem Projektionswert beim Index  $l-m$  multipliziert wird. Diese Werte werden akkumuliert. Bei diesem Vorgang kann es passieren, dass der Index  $l-m$  außerhalb der erlaubten Indizes liegt, was bedeutet, dass die Projektionsdaten an diesen Stellen unbekannt sind. In diesem Fall wird als Wert für die Projektionsdaten Null angenommen [13, S. 74].

Es sind drei Filterkerne implementiert (SIG-001). Dazu zählen ein Filterkern, der keinen Effekt hat und somit die Rückprojektion ohne Filterung darstellt, sowie die Kerne nach Ramachandran und Lakshminarayanan als auch nach Shepp und Logan. Die Werte der letzten beiden Filterkerne  $h[m]$  sind für diese Filter entsprechend den in Gleichungen 2.84 und 2.85 (S. 27) berechnet. Der Filterkern, der keiner Filterung entspricht, ist durch Gleichung 3.64 gegeben.

$$h_{\text{Konstant}}[l] = \begin{cases} 1 & , \text{ wenn } l = 0 \\ 0 & , \text{ sonst} \end{cases} \quad (3.64)$$

### Rückprojektion

Die Rückprojektion, die in Abschnitt 2.62 (S. 21) erläutert wurde, ist in Listing A.26 implementiert. Die Grundlage der Implementierung ist Gleichung 2.87 (S. 28). Diese Gleichung beschreibt die Berechnung der Werte für jeden Pixel durch Aufsummierung aller gefilterten Projektionen, die den Pixel „treffen“. Die Anzahl und Größe der Pixel im rekonstruierten Bild kann prinzipiell beliebig gewählt werden. Der tatsächliche Informationsgehalt des rekonstruierten Bildes hängt jedoch primär von der Winkel- und Abstandauflösung ( $\Delta\theta$  und  $\Delta s$ ) der Projektionen beziehungsweise der gefilterten Projektionen ab. Die Anzahl der Bildpixel  $N_p$  ist in  $x$ - und  $y$ -Richtung gleich, da die Projektionen bezüglich jedes Abstandes  $s$  für alle Winkel aufgenommen wurden. Das rekonstruierte Bild ist daher quadratisch, sodass die Gesamtanzahl der Pixel gleich  $N_p^2$  ist. Die Anzahl der Pixel in beide Richtungen sollte zumindest so groß sein, dass für die Winkel  $\theta = 0 \text{ rad}$  oder  $\theta = \pi/2 \text{ rad}$  jeder Pixel genau einmal getroffen wird<sup>9</sup>. Dafür muss die Pixelanzahl in einer Richtung genau  $N_s$  betragen. Der Faktor  $f$  skaliert das

---

<sup>9</sup>Ist die Pixelanzahl kleiner würden Informationen verworfen.

Bild beliebig. Die Größe eines Pixels  $\Delta P$  entspricht mit Gleichung 3.66 einer Strecke im Körpermodell.

$$N_P = f \cdot \left\lfloor \frac{s_{\text{Mess}}}{\Delta s} + 1 \right\rfloor = f \cdot N_s \quad (3.65)$$

$$\Delta P = \frac{s_{\text{Mess}}}{N_P - 1} \quad (3.66)$$

Die Pixelposition ist durch ihre  $x$ - und  $y$ -Koordinate definiert. Da die Pixel diskret sind, lassen sie sich durch Indices  $x_i$  und  $y_i$  sowie der Messfeldgröße  $s_{\text{Mess}}$  und der Pixelgröße  $\Delta P$  beschreiben.

$$x = -\frac{s_{\text{Mess}}}{2} + x_i \cdot \Delta P \quad (3.67)$$

$$y = -\frac{s_{\text{Mess}}}{2} + y_i \cdot \Delta P \quad (3.68)$$

mit:  $x_i = 0, 1, \dots, N_P - 1$

$y_i = 0, 1, \dots, N_P - 1$

Die gesuchte diskrete Funktion  $f[x_i, y_i]$  ist gegeben durch die Summe der gefilterten Projektionen über alle Winkel  $\theta$ . Die Werte der gefilterten Projektionen werden an den Stellen  $s_k$  benötigt. Da der Abstand  $s_k$  nach Gleichung 3.70 beliebige Werte annehmen kann und die gefilterten Projektionen nur bei diskreten  $s_k$  bekannt ist, muss interpoliert werden.

$$f[x_i, y_i] = \Delta\theta \cdot \sum_{k=0}^{N_\theta-1} \tilde{p}_k(s_k) \quad (3.69)$$

mit:  $s_k = x \cdot \cos(k \cdot \Delta\theta) + y \cdot \sin(k \cdot \Delta\theta)$

$$\begin{aligned} &= \left( -\frac{s_{\text{Mess}}}{2} + x_i \cdot \Delta P \right) \cdot \cos(k \cdot \Delta\theta) \\ &+ \left( -\frac{s_{\text{Mess}}}{2} + y_i \cdot \Delta P \right) \cdot \sin(k \cdot \Delta\theta) \end{aligned} \quad (3.70)$$

Mit Gleichung 2.86 (S. 28) können durch Interpolation für beliebige  $s_k$  die Werte der gefilterten Projektionen bestimmt werden. Dafür wird der Wert  $j$ , welcher ganzzahlig wird, wenn der Wert der gefilterten Projektion bei  $s_k$  genau bekannt ist. Die Größe  $j$  ist also als *Index* zu verstehen, der auch Werte zwischen ganzen Zahlen annehmen kann. Die Interpolation geschieht linear mit den ganzzahligen Nachbarn von  $j$  über Gleichung 3.71.

$$\tilde{p}_k(s_k) = \tilde{p}_k[\lfloor j \rfloor] + \left( \tilde{p}_k[\lceil j \rceil] - \tilde{p}_k[\lfloor j \rfloor] \right) \cdot (j - \lfloor j \rfloor) \quad (3.71)$$

$$\text{mit: } j = \frac{s_k}{\Delta s} + \frac{N_s - 1}{2} \quad (3.72)$$

In der Implementierung ist die äußerste Schleife eine, die über alle Winkel iteriert. Auch hier wird wie in Abschnitt 3.6.1 von paralleler Berechnung Gebrauch gemacht. Die Winkel sind über die Threads verteilt und werden nacheinander abgearbeitet. Für jeden Winkel wird der Pixelwert entsprechend Gleichung 3.69 akkumuliert.

Durch Strahlverfolgung, -detektion, Berechnung der Projektionen, deren Filterung und Rück-

projektion lassen sich schlussendlich Schnittbilder bestimmen.

## 3.8. Grafische Benutzeroberfläche

Eine grafische Benutzeroberfläche ermöglicht die Interaktion mit dem Simulator. Ein Benutzer kann Eingaben über Tastatur und Maus tätigen sowie Ausgaben des Simulators auf dem Bildschirm erhalten (GUI-000). Die Eingaben des Benutzers lassen sich dabei direkt bei Eingabe auf Plausibilität prüfen (GUI-001). Weiterhin können dem Benutzer unmittelbar zu Eingabe- oder Ausgabeelementen Texte, die das jeweilige Element erklären, angezeigt werden (GUI-002). Insgesamt sorgt eine grafische Benutzeroberfläche für eine intuitive Bedienung des Simulators.

Für die grafische Benutzeroberfläche wird auf Bibliotheken zurückgegriffen, die die Eingabeelemente und Darstellungen von Text und Bildern ermöglichen. Die Kriterien für Auswahl der Bibliotheken sind die Unterstützung für viele Plattformen<sup>10</sup> sowie ein geringer Aufwand in der Implementierung der grafischen Benutzeroberfläche. Außerdem soll die Benutzung kostenfrei sein und die Lizenz eine Verbreitung des Simulators erlauben. Auf das *Fast Light Toolkit* (FLTK)<sup>11</sup> treffen diese Anforderungen zu. FLTK bietet zur Erstellung grafischer Benutzeroberflächen verschiedene *Widgets*, die zur Eingabe oder Ausgabe von Text dienen, als Knöpfe Aktionen auslösen können oder Bilder darstellen können.

Zur Darstellung des Strahlungsspektrums (GUI-022) und der Pixelanordnung im Detektor (GUI-023) wird auf die Bibliothek *sciplot*<sup>12</sup> zurückgegriffen. Diese Bibliothek bildet eine Schnittstelle zwischen dem in C++ geschriebenem Simulator und dem Programm *gnuplot*<sup>13</sup>. Bei Letzterem handelt es sich um ein Programm für die Darstellung wissenschaftlicher Grafiken. Diese können als Bilder in der Benutzeroberfläche angezeigt werden.

Die grafische Benutzeroberfläche ist in mehrere Fenster strukturiert. Im Hauptfenster können Körpermodelle geladen und bewegt werden. Außerdem können die Eigenschaften der Röntgenröhre und des Detektors eingestellt werden. Weiterhin können die Simulationseigenschaften angepasst werden. Wurde ein Schnittbild aufgenommen, werden in einem separaten Fenster die gesammelten Projektionsdaten und das rekonstruierte Bild angezeigt.

### 3.8.1. Hauptfenster

Das Hauptfenster in Abbildung 3.24 öffnet sich bei Start des Simulators. Über die Kopfzeile lässt sich das Fenster zur Körpermodellerstellung öffnen, bereits durchgeführte Aufnahmen importieren oder der Programmstatus zurücksetzen. Um den Simulator nutzen zu können, muss zu Beginn ein Körpermodell über die entsprechende Schaltfläche geladen werden. Geladene Modelle werden automatisch zentriert, sodass ihr Mittelpunkt bei (0 mm, 0 mm, 0 mm) liegt. Um das Körpermodell betrachten zu können, wird der Schnitt durch das Modell entlang der xy-Ebene bei  $z = 0$  mm angezeigt (GUI-011). Das Körpermodell lässt sich um die x- und

---

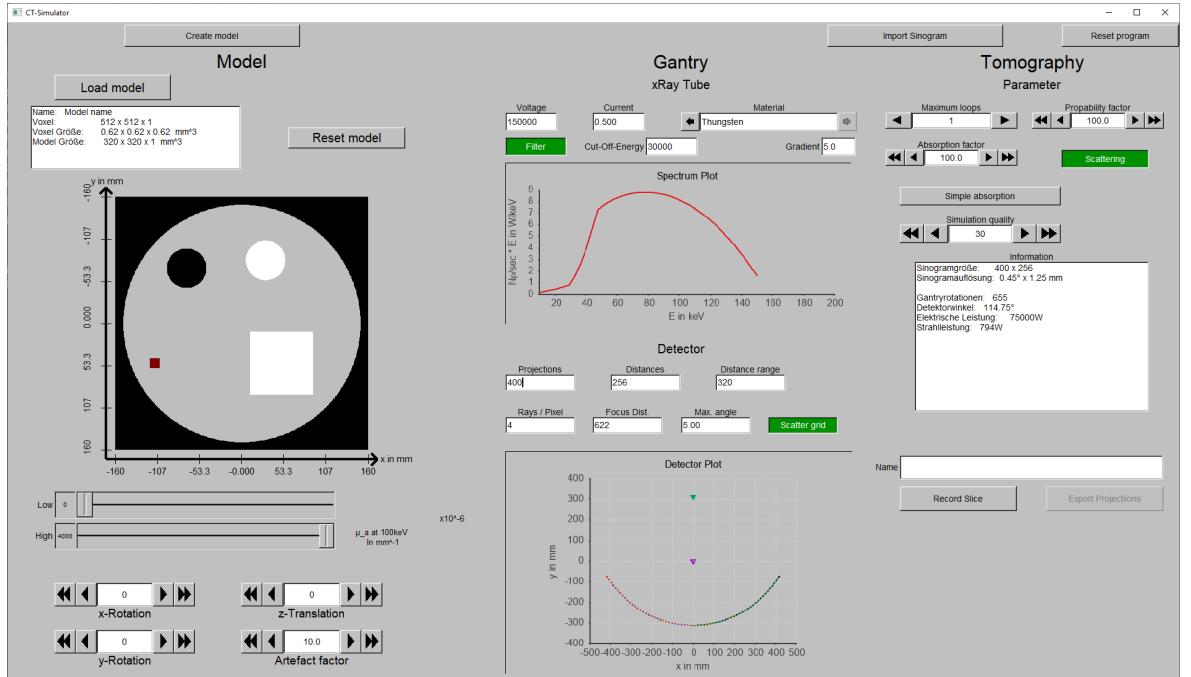
<sup>10</sup>Dazu zählen Windows®, Unix®/Linux® und macOS® Betriebssysteme

<sup>11</sup><https://www.fltk.org/index.php>

<sup>12</sup><https://sciplot.github.io/>

<sup>13</sup><http://www.gnuplot.info/>

$y$ -Achse rotieren und entlang der  $z$ -Achse verschieben (GUI-012). Über zwei Schieberegler lassen sich  $\mu_{\text{Schwarz}}$  und  $\mu_{\text{Weiß}}$  einstellen, sodass der Kontrast des Schnittbildes variabel ist. Weiterhin wird der gespeicherte Absorptionskoeffizient an jener Stelle, auf die der Mauszeiger zeigt, angezeigt. Voxel mit der Eigenschaft Metall werden in Rot angezeigt und lassen sich in ihrer Absorption einstellen<sup>14</sup> (GUI-013).



**Abbildung 3.24.:** Hauptfenster der grafischen Benutzeroberfläche

Für die Röntgenröhre lassen sich Anodenspannung  $U_A$  sowie Anodenstrom  $I_A$  und -material einstellen. Weiterhin kann ein Röntgenvorfilter aktiviert werden. Dieser wird über eine Energie, unterhalb derer die Photonenflüsse nahezu null sind, und einen Gradienten, der die Steilheit der Filterflanke einstellt, charakterisiert. Zu den gewählten Eigenschaften wird das Spektrum der Röhre angezeigt (GUI-022). Das Spektrum stellt die Leistungsdichte in  $\frac{\text{W}}{\text{keV}}$ , die sich aus dem Produkt des Photonenfluxes und der Photonenenergie  $n(E) \cdot E$  ergibt, dar.

Als Detektoreigenschaften werden die Anzahl der Projektionswinkel  $N_\theta$  und der Projektionen je Winkel  $N_s$  sowie die Größe des Messfeldes  $s_{\text{Mess}}$  festgelegt. Außerdem wird der Abstand von Detektor und Röntgenröhre festgelegt. Diese vier Parameter bestimmen vollständig die Anordnung der Pixel. Diese wird in einer Abbildung angezeigt (GUI-023). Weiterhin werden die Anzahl der simulierten Strahlen je Pixel festgelegt und der Akzeptanzwinkel des optionalen Streustrahlungsrasters eingestellt.

Im rechten Teil des Hauptfensters werden Simulationsparameter festgelegt. Dazu zählen die maximale Anzahl der Durchläufe je Gantryposition<sup>15</sup> und ein Faktor für die Wahrscheinlichkeit sowie Absorption<sup>16</sup> bei Streuung. Das Ein- und Ausschalten von Streuung (GUI-030)

<sup>14</sup>Der assozierte Absorptionskoeffizient kann zwischen jenem von Wasser und jenem von Titan eingestellt werden.

<sup>15</sup>Das bezieht sich darauf, ob und wie oft gestreute Strahlen selber gestreut werden können (siehe S. 73).

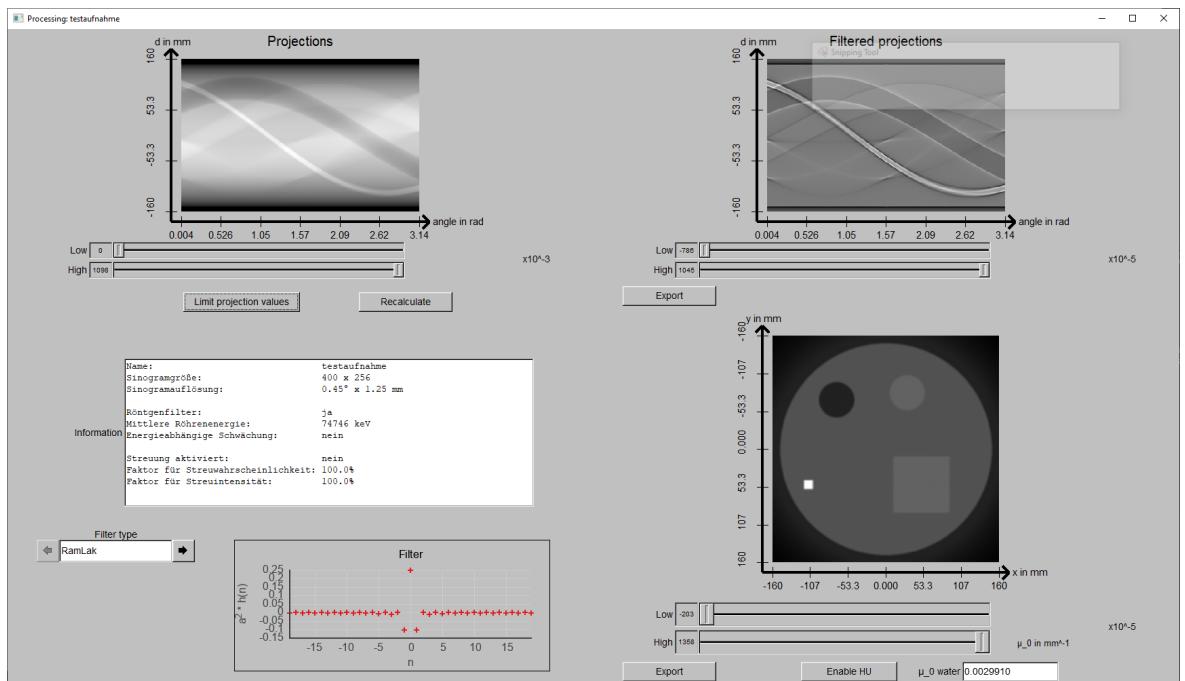
<sup>16</sup>Ein Faktor, der den Energieverlust bei Compton-Streuung verringern kann.

und energieabhängiger Schwächung (GUI-031) ist ebenfalls möglich. Des Weiteren ist die Qualität der Simulation<sup>17</sup> einstellbar. Für eine Aufnahme lässt sich ein Name vergeben und nach fertiggestellter Aufnahme diese exportieren sowie zu späterem Zeitpunkt importieren (GUI-040).

Alle Einstellungen, die im Hauptfenster festgelegt sind, werden bei Schließen des Simulators automatisch gespeichert und beim Öffnen wiederhergestellt (GUI-003). Dazu wird im Programmordner der Ordner `programState` genutzt. Durch Löschen des Ordners wird der Programmstatus vollständig zurückgesetzt. Dazu muss das Programm geschlossen sein. Alternativ kann der Programmstatus über die entsprechende Schaltfläche zurückgesetzt werden. Das Zurücksetzen geschieht, sobald das Programm geschlossen wird.

### 3.8.2. Verarbeitung

Sobald die Projektionsdaten für ein Schnittbild vollständig gesammelt wurden, öffnet sich ein Fenster, das die Rekonstruktion abbildet. Solch ein Fenster ist in Abbildung 3.25 abgebildet. Oben links werden die Rohdaten der Projektionen als Sinogramm angezeigt (GUI-041). Auch hier lassen sich  $\mu_{\text{Schwarz}}$  und  $\mu_{\text{Weiß}}$  beliebig einstellen. Zusätzlich lässt sich über eine Schaltfläche die Begrenzung des Wertebereiches (siehe Abschnitt 3.7 S. 77) auf diese Werte für die weitere Berechnung aktivieren. Es wird jeweils der Projektionswert, der sich an der Mauszeigerposition befindet, angezeigt. Unten links kann der Filtertyp zur Filterung



**Abbildung 3.25.:** Verarbeitungsfenster der grafischen Benutzeroberfläche

der Projektionen ausgewählt werden. Der Filterkern wird in einer Abbildung neben dem Auswahlelement dargestellt.

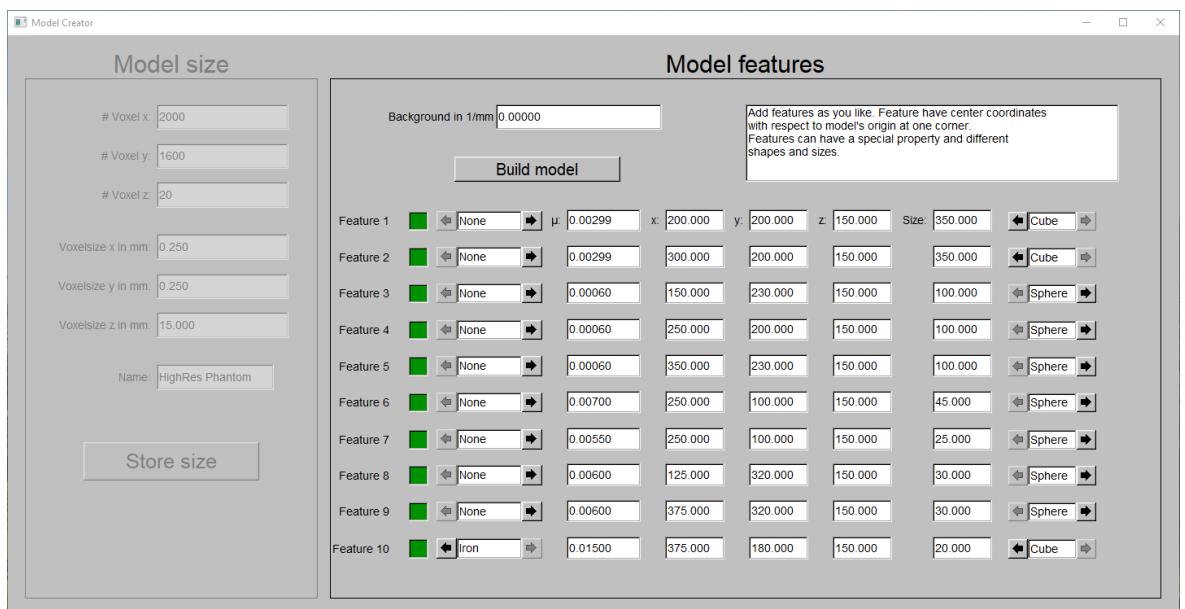
Auf der rechten Seite werden die gefilterten Projektionen (GUI-042) und das rekonstruierte Bild (GUI-042) angezeigt. Das rekonstruierte Bild stellt standardmäßig die berechneten

<sup>17</sup>Die Simulationsqualität wird in Abschnitt 4.5.1 erläutert.

Schwächungskoeffizienten dar. Der Kontrast des rekonstruierten Bildes in ebenfalls einstellbar (GUI-044). Es besteht die Möglichkeit in ein Textfeld den Schwächungskoeffizienten, der für eine Region aus Wasser berechnet wurde, einzugeben. Über einen Knopf lassen sich dann statt der Schwächungskoeffizienten die entsprechenden Hounsfield-Einheiten anzeigen.

### 3.8.3. Modellerstellung

Einfache Körpermodelle, die aus geometrischen Objekten zusammengesetzt werden, können erstellt werden. Die Oberfläche zu Erstellung ist in Abbildung 3.26 zu sehen. Auf der linken Seite werden die Anzahl der Voxel in jede Raumrichtung sowie deren Größe angegeben. Auf der rechten Seite wird das Körpermodell aus bis zu zehn Objekten zusammengesetzt. Ein Standardwert für den Absorptionskoeffizienten aller Voxel kann angegeben werden. Die einzelnen geometrischen Objekte können jeweils aktiviert oder deaktiviert werden. Für die Objekte lassen sich jeweils eine Spezialeigenschaft sowie ein Absorptionskoeffizient angeben. Als Spezialeigenschaft steht nur „Metall“ zur Auswahl. Außerdem kann die Position, Größe und Form des Objektes eingestellt werden. Als Formen stehen „Würfel“ und „Kugel“ zur Verfügung. Das kreierte Körpermodell kann nach Erstellen gespeichert werden.



**Abbildung 3.26.:** Fenster zur Erstellung einfacher Körpermodelle



# 4. Verifikation

Um die Implementierung der unterschiedlichen Simulatorbestandteile zu überprüfen, werden zu einzelnen Komponenten Testfunktionen geschrieben. Die jeweiligen Funktionen simulieren für festgelegte Parameter, die eine verständliche Darstellung der Ergebnisse ermöglichen, einzelne Gerätekomponenten, Algorithmen und physikalische Modelle. Die Ergebnisse werden über eine Schnittstelle, die in den Dateien `plotting.h/.hpp/.cpp` implementiert ist, in MATLAB angezeigt. Dazu werden die anzuzeigenden Daten in Dateien, die durch in MATLAB implementierte Funktionen gelesen werden können, gespeichert.

## 4.1. Gerätekomponenten

Die in Abschnitt 3.3 (S. 50) beschriebene Konstruktion des Detektors sowie die Erzeugung der Strahlen sind abhängig von gewissen Parametern, die für die Erprobung der entsprechenden Programmmodulen festgelegt werden. Für die Detektorkonstruktion sind die Parameter die Eigenschaften des Sinogrammrasters, zu denen die Anzahl der Distanzen und Projektionswinkel sowie die Messfeldgröße zählt. Für die Strahlerzeugung sind die Parameter die Röhrenspannung, der Röhrenstrom und das Anodenmaterial.

**Anzahl Projektionen und Distanzen**

$$N_\theta = 30, N_s = 18$$

**Messfeldgröße**

$$s_{\text{Mess}} = 400 \text{ mm}$$

**Detektor-Fokus-Distanz**

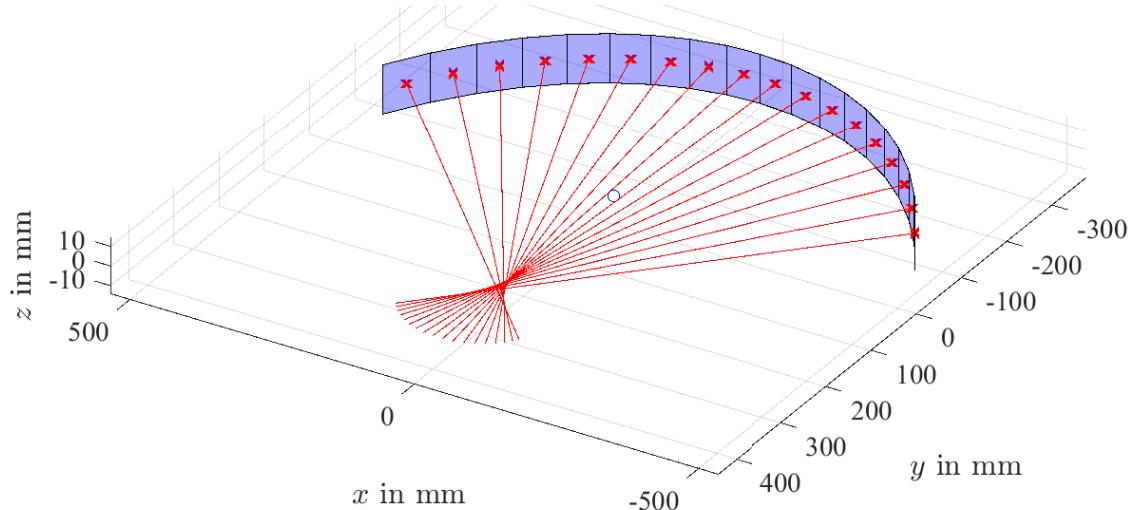
$$D_{\text{FD}} = 650 \text{ mm}$$

**Röhreneigenschaften**

$$U_A = 140 \text{ kV}, I_A = 0,5 \text{ A}, Z = 74 \text{ (Wolfram)}$$

### 4.1.1. Detektorkonstruktion

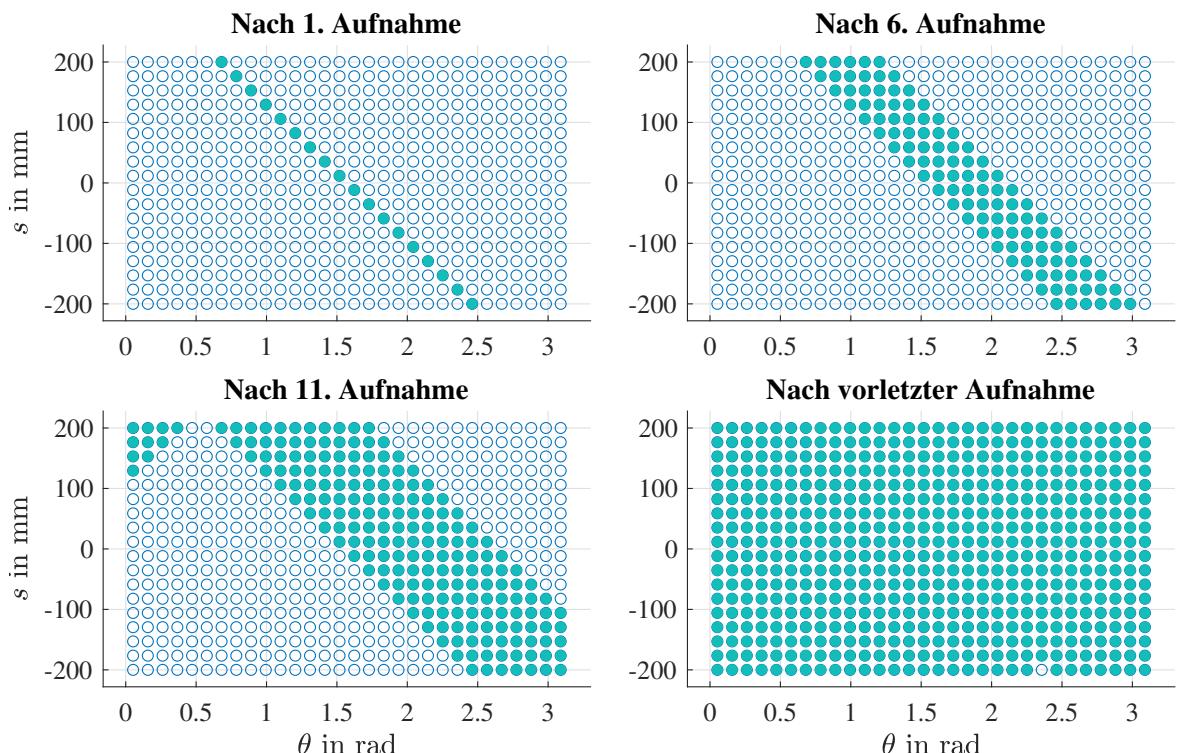
Die Anordnung der Pixel im Detektor ergibt sich aus der Anzahl der Projektionen und Distanzen sowie der Messfeldgröße und der Detektor-Fokus-Distanz. In Abbildung 4.1 ist die Anordnung der Pixel, die durch den Konstruktionsalgorithmus mit  $N_\theta = 30, N_s = 18, s_{\text{Mess}} = 400 \text{ mm}$  und  $D_{\text{FD}} = 650 \text{ mm}$  erzeugt wird, abgebildet. Es ist zu erkennen, dass die



**Abbildung 4.1.:** Aus gegebenen Parametern konstruierte Pixelanordnung. Blau: Pixelebenen. Rot: Pixelnormalen

Anzahl der Pixel der Anzahl der Distanzen  $N_s$  entspricht. Bemerkenswert ist, dass sich die Pixelnormalen nicht in einem Punkt schneiden. Der Grund dafür ist, dass die Pixel so konstruiert sind, dass die Pixelnormale exakt die Rasterpunkte im Sinogramm treffen und dies kein Schnitt in einem gemeinsamen Fokus impliziert. Die Pixel liegen auch aus diesem Grund nicht auf einem kreisförmigen Segment, sondern einer anderen unbekannten geometrischen Figur.

Wird ein Strahl detektiert, wird der Wert des Linienintegrals in das Sinogramm eingetragen. Der Punkt, an dem in das Sinogramm eingetragen wird, ist dabei durch die Pixelnormale des Pixels, auf den der Strahl trifft, bestimmt. Als *Aufnahme* wird die Messung der Projektionsdaten für eine Position des Pixelarrays bezeichnet. In Abbildung 4.2 ist der Prozess wie das Sinogramm von Aufnahme zu Aufnahme mit Daten gefüllt wird, abgebildet. Die kreisförmigen Konturen zeigen die Punkte im Raster mit genau  $\Delta\theta \approx 0,1047$  rad und  $\Delta s \approx 23,529$  mm an. Den Konturen überlagert werden ausgefüllte Kreise, die anzeigen, an welchen Punkten



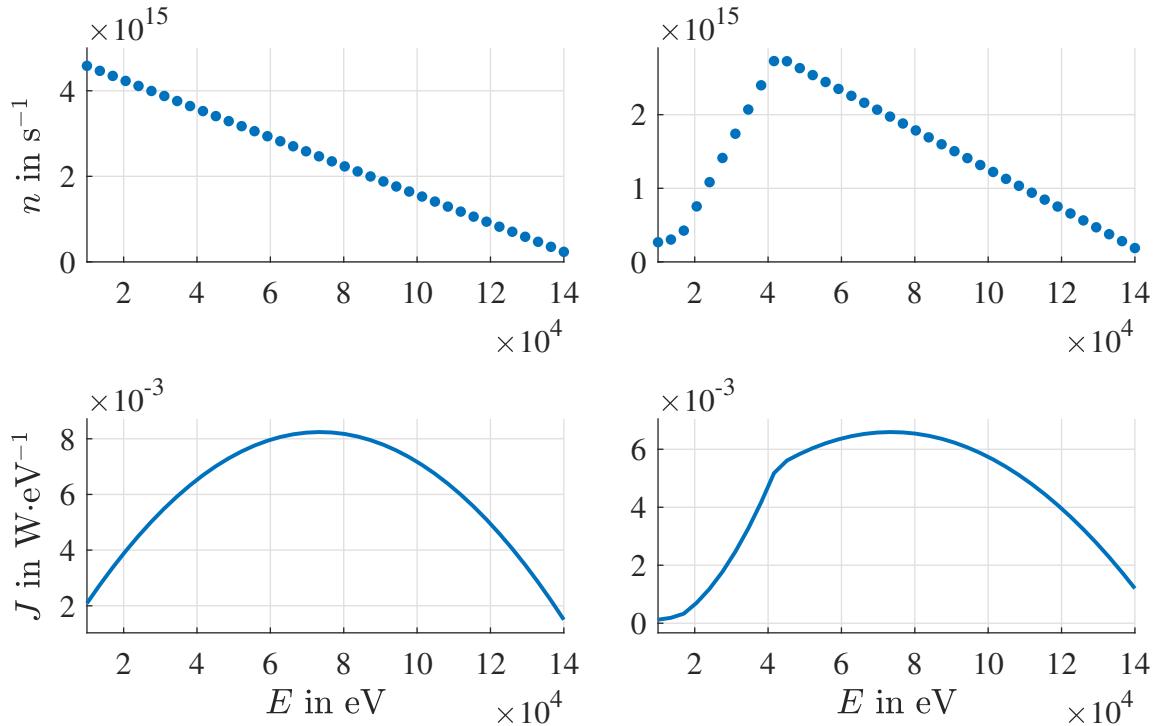
**Abbildung 4.2.:** Schrittweises Füllen des Sinogramms mit Projektionsdaten. Konturen: Rasterpunkte. Füllung: Radon-Koordinaten der Pixelnormalen mit Daten

Projektionsdaten gemessen wurden. Aufgrund der Pixelanordnung liegen die ausgefüllten Kreise exakt innerhalb der Konturen. Außerdem ist zu erkennen, dass das Sinogramm durch die Daten einer Aufnahme „diagonal“ gefüllt wird. Grund dafür ist, dass die Pixel nicht entlang einer Geraden, sondern in einem Bogen angeordnet sind. Lägen die Pixel auf einer Geraden, würde je Aufnahme genau eine Spalte des Sinogramms gefüllt werden. Nach der vorletzten Aufnahme fällt auf, dass nur noch genau ein Rasterpunkt über keine Daten verfügt. Da je Aufnahme immer genau  $N_s$  Rasterpunkte mit Daten gefüllt werden, heißt das, dass bei dieser und mehreren vorherigen Aufnahme Rasterpunkten redundante Daten zugeordnet

werden.

### 4.1.2. Strahlerzeugung

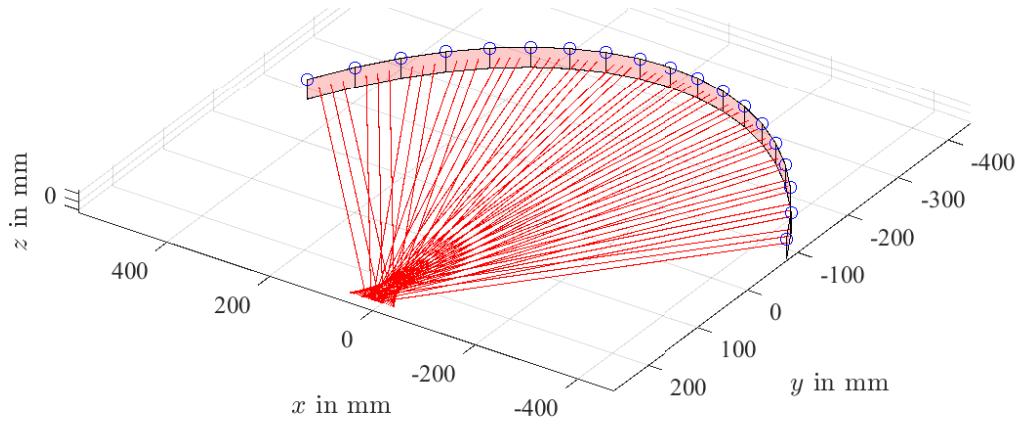
Für die Erzeugung der Strahlen müssen sowohl das Spektrum der Röntgenröhre als auch die Strahlenrichtungen überprüft werden. Die Form des Spektrums ergibt sich aus den gegebenen Parametern  $U_A = 140\text{ kV}$ ,  $I_A = 0,5\text{ A}$  und  $Z = 74$ . In Abbildung 4.3 sind die spektralen Verteilungen der Photonenflüsse für diese Parameter in der oberen Grafik abgebildet. Ohne



**Abbildung 4.3.:** Aus Röhreneigenschaften berechnetes Spektrum der Röntgenröhre. Oben links: Photonenfluss ohne Vorfilter. Oben rechts: Photonenfluss mit Vorfilter. Unten links: Spektrale Leistungsdichte ohne Vorfilter. Unten rechts: spektrale Leistungsdichte mit Vorfilter

Vorfilter (links) entspricht der Verlauf jenem der Bremsstrahlung. Mit Vorfilter (rechts) ist der niederenergetische Teil abgeschwächt. Die unteren Grafiken zeigen jeweils die spektrale Leistungsdichte, die sich aus den Photonenflüssen durch Multiplikation des Photonenflusses mit der Photonenergie ergibt, dargestellt. Diese Kurven sind als kontinuierlich zu verstehen, sodass die Integration über sie die Gesamtstrahlungsleistung  $J_{\text{ges}}$  ergibt.

Für jeden Pixel werden Strahlen, die sie treffen, erzeugt. Die Strahlen stehen senkrecht auf den Pixeln und sind entlang der Pixelebene so verschoben, dass bei mehreren Strahlen pro Pixel, diese gleichmäßig verteilt sind. In Abbildung 4.4 sind jeweils drei Strahlen pro Pixel erzeugt worden.

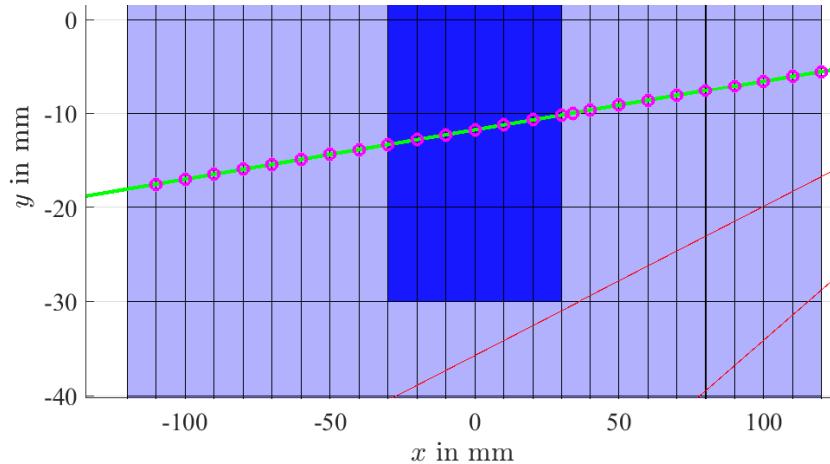


**Abbildung 4.4.:** Erzeugung von drei Strahlen je Pixel. Die erzeugten Strahlen stehen senkrecht und gleichmäßig verteilt auf den Pixeln

## 4.2. Strahlverfolgung

Zur Verifikation der Strahlverfolgung wird ein einfaches Körpermodell genutzt. Das Körpermodell verfügt über  $N_x = 24$ ,  $N_y = 24$  und  $N_z = 24$  Voxel, die jeweils eine Größe von  $\Delta x = 10 \text{ mm}$ ,  $\Delta y = 10 \text{ mm}$  und  $\Delta z = 10 \text{ mm}$  haben. Das Körpermodell besteht aus zwei Regionen. In der Mitte des Körpermodells befindet sich eine würfelförmige Region mit einer Größe von 6 Voxeln in jede Raumrichtung, deren Absorptionskoeffizient von dem der anderen Voxel abweicht.

In Abbildung 4.5 ist ein Strahl sowie ein Schnitt durch das Körpermodell dargestellt. Ebenfalls sind die Schnittpunkte des Strahls mit den Randflächen der Voxel abgebildet. Die Schnittpunkte wurden durch den Algorithmus zur Strahlverfolgung bestimmt. Es ist zu erkennen, dass die eingezeichneten Schnittpunkte an den erwarteten Stellen liegen.



**Abbildung 4.5.:** Schnittpunkte eines Strahls mit den Voxeln im Körpermodell

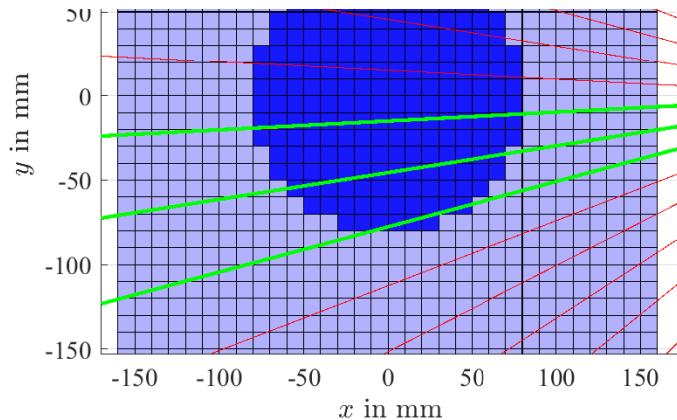
## 4.3. Physikalische Modelle

Zur Überprüfung der physikalischen Modelle, zu denen die Absorption und Streuung zählen, wird ein Körpermodell genutzt, das nur aus Voxeln mit  $\mu = 0 \frac{1}{\text{mm}}$  und einer zentralen Kugel mit  $\mu = 3 \cdot 10^{-3} \frac{1}{\text{mm}}$  und einem Radius von 80 mm besteht. Die Koeffizienten entsprechen den Materialien Luft und Wasser. Die Größe der Voxel ist  $\Delta x = 10 \text{ mm}$ ,  $\Delta y = 10 \text{ mm}$  und

$$\Delta y = 10 \text{ mm}.$$

Zur Darstellung und Überprüfung, wie sich die Strahleigenschaften während des Durchgangs durch das Körpermodell verändern, muss der Änderungsverlauf der Eigenschaften gespeichert werden. Dazu wird für jeden Schnittpunkt des Strahls mit Voxeln der Punkt und die Strahleigenschaften an diesem Punkt gespeichert.

In Abbildung 4.6 ist ein Schnitt durch das Körpermodell mit der zentralen Region aus Wasser zu sehen. Außerdem sind mehrere Strahlen abgebildet.



**Abbildung 4.6.:** Schnitt durch das Körpermodell mit einer zentralen Kugel mit Absorptionseigenschaften wie Wasser und mehrere Strahlen. Die grünen Strahlen werden von Oben nach Unten als *Strahl 1*, *Strahl 2* und *Strahl 3* bezeichnet

### 4.3.1. Absorption

In Abbildung 4.6 sind jene Strahlen, die näher untersucht werden, in Grün hervorgehoben. In diesem Abschnitt wird nur die Absorption untersucht. Aus diesem Grund ist die Streuung während der Simulationen in diesem Abschnitt deaktiviert. In Abbildung 4.7 sind sowohl die Veränderung der Gesamtstrahlungsleistung des Strahls  $J_{\text{ges}}$  und die mittlere Energie des Spektrums  $\bar{E}$  in Abhängigkeit von der durchstrahlten Weglänge  $d$  abgebildet. Es ist zu erkennen, dass die Gesamtstrahlungsleistung, die nach 3.61 (S. 77) berechnet ist, erwartungsgemäß beim Durchgang durch das Körpermodell abnimmt.

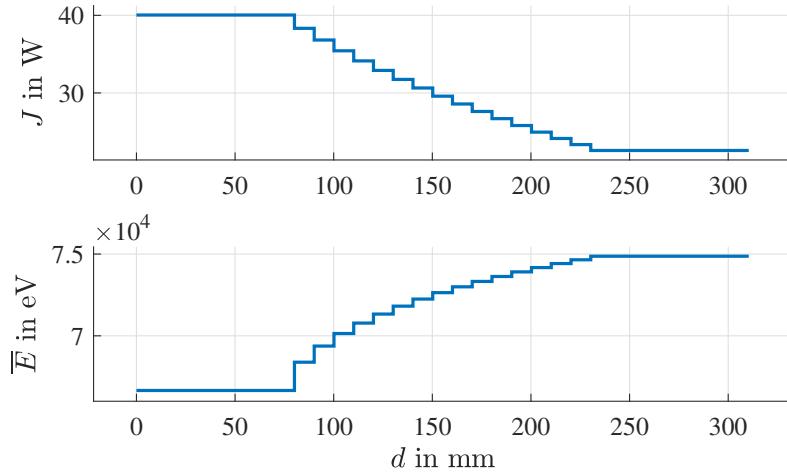
#### Strahlaufhärtung

Die mittlere Energie, wird nach Gleichung 4.1 berechnet. In dieser Gleichung werden die Energien mit den jeweiligen Photonenflüssen gewichtet und durch den gesamten Fluss der Photonen geteilt. Der so berechnete Wert hat die Dimension einer Energie und kann analog zu einem Erwartungswert einer Wahrscheinlichkeitsverteilung verstanden werden, da  $E_i$  jeweils mit der relativen Häufigkeit  $n(E_i) \cdot (\sum_i n(E_i))^{-1}$  gewichtet wird.

$$\bar{E} = \frac{\sum_i E_i \cdot n(E_i)}{\sum_i n(E_i)} \quad (4.1)$$

Am Verlauf der mittleren Energie in Abbildung 4.7 ist die Strahlaufhärtung, die durch den Anstieg der mittleren Energie charakterisiert ist, deutlich zu erkennen.

Um den Effekt, den die Strahlaufhärtung auf die Berechnung der Projektionswerte hat, zu

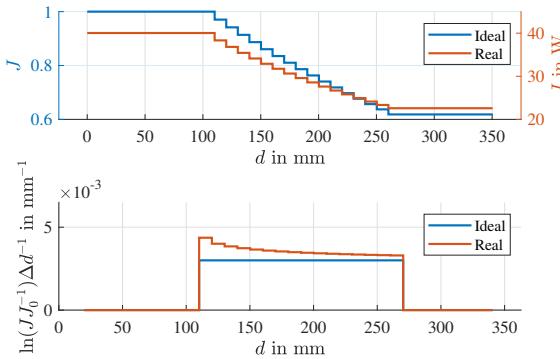


**Abbildung 4.7.:** Veränderung der mittleren Strahlungsenergie bei Transmission. Gesamtstrahlungsleitung  $J_{\text{ges}}$  und mittlere Energie  $\bar{E}$  des Strahl 1 in Abhängigkeit von der durchstrahlten Strecke

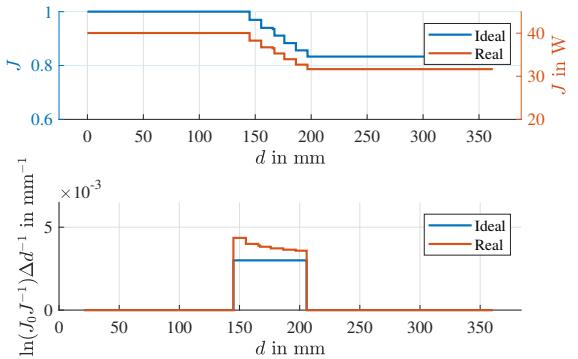
zeigen, wurden die Abbildungen 4.8.a und 4.8.b erstellt. Diese stellen in der oberen Grafik die ideale und reale Intensität in Abhängigkeit von der Strecke entlang des Weges durch das Körpermodell dar. Die ideale Intensität ist jene, die in Abschnitt 3.5.2 (S. 66) als *vereinfachte Intensität* beschrieben wird. Sie beschreibt die Intensität für den monoenergetischen Fall. Die reale Intensität ist die Gesamtstrahlungsleitung  $J_{\text{ges}}$ , die aus dem Spektrum berechnet wird. Unten in den Abbildungen ist für jeden Streckenabschnitt jeweils der Schwächungskoeffizient nach Gleichung 4.2 berechnet. Jeder Streckenabschnitt entspricht der Strecke innerhalb eines Voxels. Die Länge der Strecke ist  $\Delta d$  und die Intensität beim Eintritt in das Voxel  $J_0$ . Die Intensität beim Austritt aus dem Voxel ist  $J$  oder  $J_{\text{ges}}$ .

$$\mu = \ln \left( \frac{J_0}{J} \right) \cdot \Delta d^{-1} \quad (4.2)$$

Die Voxel in der zentralen Kugel haben einen Absorptionskoeffizienten von  $\mu = 3 \cdot 10^{-3} \frac{1}{\text{mm}}$ . Wie zu erwarten, liefert die Berechnung des Absorptionskoeffizienten nach Gleichung 4.2 für den idealen Fall den erwarteten Wert von  $3 \cdot 10^{-3} \frac{1}{\text{mm}}$ . Der reale Fall wird durch die Simulation des Spektrums und der energieabhängigen Schwächung im Simulator abgebildet. Es fällt auf, dass der Absorptionskoeffizient für alle Voxel überschätzt wird. Besonders jedoch für jene, die vom Strahl zuerst durchlaufen werden. Wie in Abschnitt 2.5.3 beschrieben, ist das begründet darin, dass die Absorption energieabhängig ist. So werden bei der Absorption im ersten Voxel jene geringeren Energien stärker geschwächt, die auch im nächsten Voxel stärker geschwächt würden. Da aber nach dem ersten Voxel die mittlere Energie des Spektrums größer wird (Abb. 4.7), ist das Verhältnis  $\frac{J_0}{J}$  und damit der berechnete Koeffizient geringer. Hervorzuheben ist noch, dass in 4.8.b der Strahl eine kürzere Strecke in der Wasserregion verbringt. Die Differenz zwischen dem idealen und realen Fall ist dort größer. Werden nur die Intensitäten am Ein- und Ausgang der Wasserregion betrachtet und für diese der Absorptionskoeffizient



4.8.a: Strahl 1. Längere Strecke



4.8.b: Strahl 3. Kürzere Strecke

**Abbildung 4.8.:** Abhängigkeit der Strahlungsgesamtleistung und berechneten Absorptionskoeffizienten in Abhängigkeit von der Strecke entlang des durchstrahlten Weges für den idealen und realen Fall

berechnet, ist ebenfalls die Differenz bei der kürzeren Strecke größer.

$$\text{Abb. 4.8.a: } \mu = \ln\left(\frac{40,04}{22,29}\right) \cdot (260,38 - 110,15)^{-1} \frac{1}{\text{mm}} \approx 3,9 \cdot 10^{-3} \frac{1}{\text{mm}} \quad (4.3)$$

$$\text{Abb. 4.8.b: } \mu = \ln\left(\frac{40,04}{31,66}\right) \cdot (196,70 - 144,94)^{-1} \frac{1}{\text{mm}} \approx 4,5 \cdot 10^{-3} \frac{1}{\text{mm}} \quad (4.4)$$

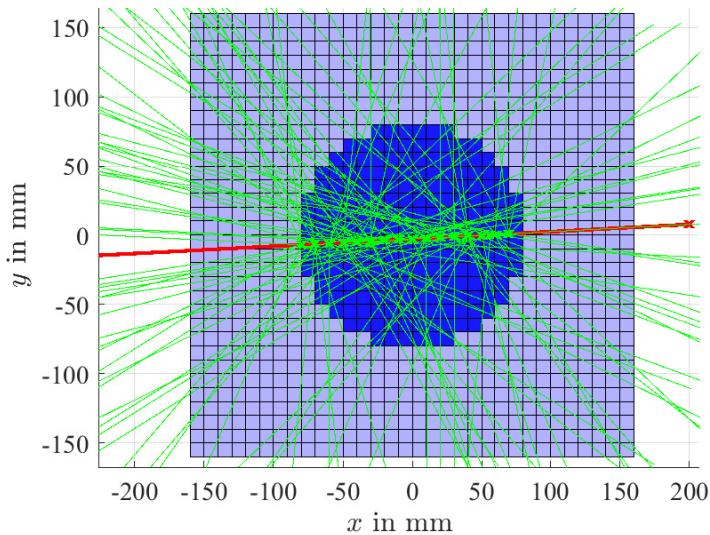
Die Absorption bei kürzerer Strecke ist also stärker überschätzt, was zu den in Abschnitt 2.5.3 (S. 34) beschriebenen *Cupping*-Artefakten führt. Diese zeichnen sich dadurch aus, dass in den Randbereichen, in denen Strahlen eine geringere Strecke zurücklegen, der berechnete Koeffizient größer als jener in den innen liegenden Bereichen ist.

### 4.3.2. Streuung

Es sollen mehrere Eigenschaften der Compton-Streuung verifiziert werden. Dazu zählen der energieabhängige Wirkungsquerschnitt, die Verteilung der Streuwinkel und die Schwächung von Strahlen durch Streuung. Untersucht wird die Streuung erneut am Wassermodell aus dem vorherigen Abschnitt. Die Anzahl der möglichen Streuwinkel ist auf 63 festgelegt. In Abbildung 4.9 ist das Wassermodell sowie ein Strahl, der gestreut wird, abgebildet. Ebenfalls sind die Strahlen zweiter Ordnung, die die gestreuten Strahlen aus dem initialen Strahl sind, zu sehen.

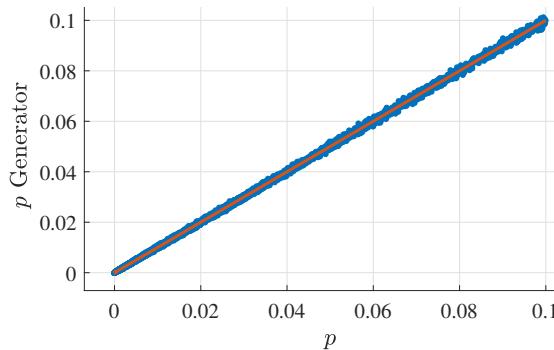
Ob der Zufallszahlengenerator die erwarteten Ergebnisse liefert, soll überprüft werden. Dazu werden für jeweils 10000 Iterationen für alle möglichen Wahrscheinlichkeiten mit Gleichung 3.55 (S. 70) durch den Zufallsgenerator bestimmt, ob ein zufälliges Ereignis eintritt. Die relative Häufigkeit der eingetretenen Ereignisse sollte dabei der Wahrscheinlichkeit, dass das Ereignis eintritt, entsprechen. In Abbildung 4.10.a sind für die Wahrscheinlichkeiten zwischen 0 % und 10 % die relativen Häufigkeiten, dass Ereignisse eintreten, abgebildet. Es ist zu erkennen, dass die relative Häufigkeit der eingetretenen Ereignisse den jeweiligen Wahrscheinlichkeiten entspricht.

Zur Überprüfung, ob durch Berechnung der Streuwahrscheinlichkeiten und der Erzeugung zufälliger Streuereignisse die Wechselwirkungsquerschnitte korrekt abgebildet sind, werden für

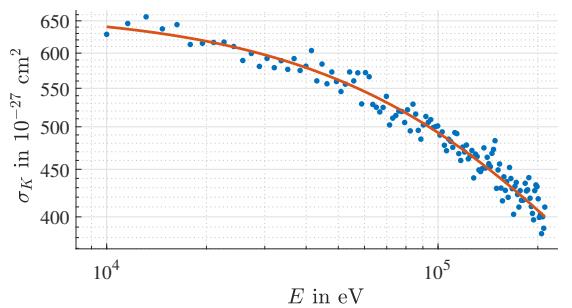


**Abbildung 4.9.:** Initialer (Rot) und gestreute (Grün) Strahlen im Wassermodell.

jede diskrete Energie im Strahlenspektrum zufällige Ereignisse ausgewertet. In 100 000 Iterationen werden dazu jeweils für die Energien die Wechselwirkungsquerschnitte nach Gleichung 2.24 (S. 13) bestimmt. Diese werden nach Gleichung 2.28 (S. 13) in eine Wahrscheinlichkeit überführt. Der Zufallsgenerator bestimmt, wie in Abschnitt 3.5.3 (S. 69) beschrieben, anhand dieser Wahrscheinlichkeit, ob Streuung stattfindet. Findet Streuung statt, wird für die aktuelle Energie ein Zähler um Eins erhöht. Nach den 100 000 Durchläufen werden die Zähler durch diese Anzahl geteilt und in eine relative Häufigkeit der Streuung umgewandelt, welche wiederum der Wahrscheinlichkeit entsprechen sollte. Durch Inversion von Gleichung 2.28 lässt sich wiederum der Wirkungsquerschnitt berechnen und mit dem erwarteten Wert vergleichen. In Abbildung 4.10.b sind die Wirkungsquerschnitte, die durch den eben beschriebenen Prozess erzeugt wurden, abgebildet. Ebenfalls sind die aus Gleichung 2.24 berechneten Wirkungsquerschnitte dargestellt. Die Simulation stimmt mit der Erwartung überein.



**4.10.a:** Relative Häufigkeiten zufälliger Ereignisse mit bestimmten Wahrscheinlichkeiten.

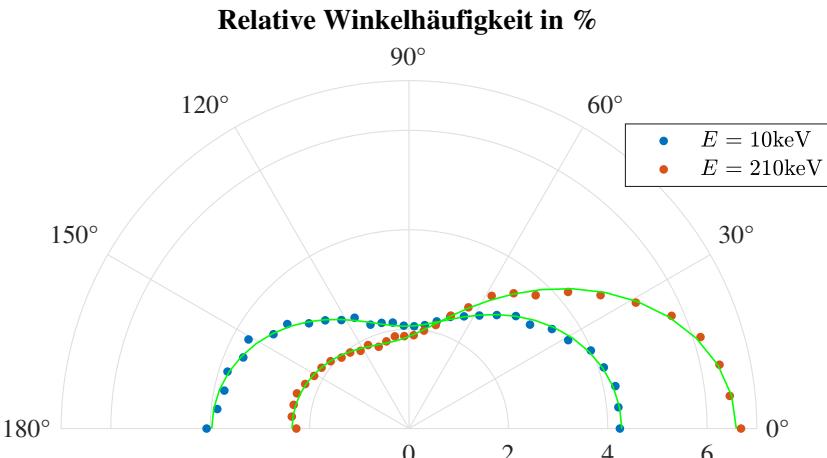


**4.10.b:** Aus zufälligen Ereignissen ermittelte Wirkungsquerschnitte der Compton-Streuung

**Abbildung 4.10.:** Erprobung des Zufallszahlengenerators und dem zufälligen Eintreten von Streuereignissen in Abhängigkeit der Energie

Bei der Bewertung der zufälligen Auswahl von Streuwinkeln werden ebenfalls für viele Iterationen Winkel wie in Abschnitt 3.5.3 beschrieben, erzeugt und deren relative Häufigkeit bestimmt. In Abbildung 4.11 sind für zwei Energien für jeweils 32 diskrete Winkel die

relativen Häufigkeiten, dass diese aus der Gesamtmenge ausgewählt werden, abgebildet. In Grün ist der erwartete Verlauf, der jenem aus Abbildung 2.3 (S. 14) entspricht, abgebildet. Die grünen Kurven werden als 32 einzelne Punkte<sup>1</sup> nach Gleichung 2.29 berechnet und dann so skaliert, dass deren Summe genau Eins ist. Dadurch korrespondieren diese Punkte mit den simulierten relativen Häufigkeiten.

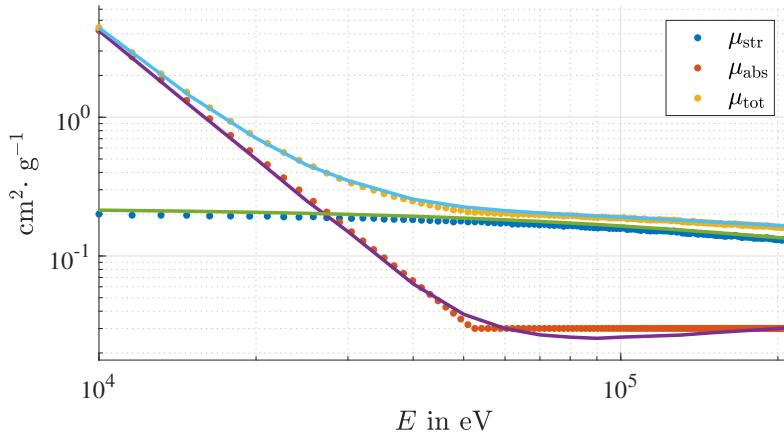


**Abbildung 4.11.:** Relative Häufigkeit diskreter Streuwinkel bei Eintreten von Streuereignissen für zwei Energien

### 4.3.3. Gesamte Schwächung

Die Absorption und Streuung tragen jeweils zur gesamten Schwächung der Röntgenstrahlen bei. Der gesamte Schwächungskoeffizient  $\mu_{\text{tot}}$  ist für die Simulation die Summe des Absorptionskoeffizienten durch den Photoeffekt  $\mu_{\text{abs}}$  und die Schwächung durch Compton-Streuung  $\mu_{\text{str}}$ . Zur Verifikation werden drei Spektren für einen Strahl, der durch das Wassermodell verfolgt wird, simuliert. Die Spektren werden jeweils nur durch Absorption, Streuung und deren Kombination modifiziert. Die Koeffizienten werden dann jeweils nach Gleichung 4.2 (S. 90) berechnet. Die Strecke  $\Delta d$  ist dabei jene Strecke, die der Strahl innerhalb der Wasserregion des Körpermodells verbringt. In Abbildung 4.12 sind die einzelnen Koeffizienten, die sich aus der Simulation ergeben, in Abhängigkeit von der Photonenergie als Punkte abgebildet. Ebenfalls sind die realen Verläufe als Kurven dargestellt. Der Verlauf des Absorptionskoeffizienten  $\mu_{\text{abs}}$  entspricht der Implementierung in der Hinsicht, dass dieser bis zur Energie 50 keV mit  $\propto E^{-3}$  fällt und für größere Energien konstant ist. Im Vergleich dazu ist der reale Verlauf der Absorption aus Abbildung 2.4 (S. 15) in Violet dargestellt. Zusätzlich sind der Schwächungskoeffizient durch Streuung  $\mu_{\text{str}}$  und die gesamte Schwächung  $\mu_{\text{tot}}$  und deren reale Entsprechung abgebildet. Es ist erkennbar, dass  $\mu_{\text{tot}}$  wie zu erwarten die Summe von  $\mu_{\text{abs}}$  und  $\mu_{\text{str}}$  ist. Insgesamt entsprechen die Koeffizienten, die sich aus den Veränderungen der Spektren während der Durchstrahlung des Wassermodells ergeben, in adäquater Weise der Realität.

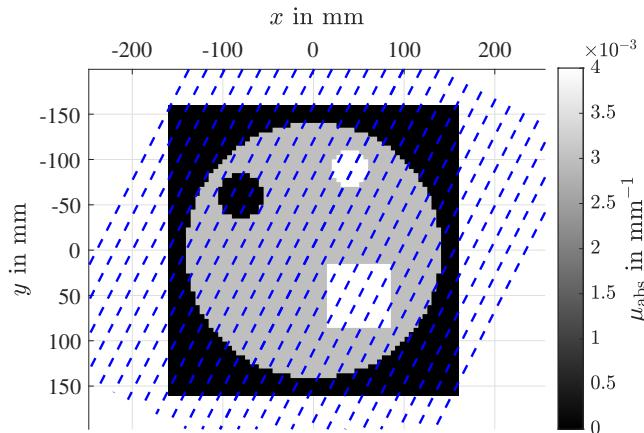
<sup>1</sup>Die Punkte werden von MATLAB durch Verbinden mit Linienelementen zu einer kontinuierlichen Kurve



**Abbildung 4.12.:** Schwächungskoeffizienten, die sich aus der Simulation ergeben und deren Entsprechung aus den physikalischen Modellen

## 4.4. Rückprojektion

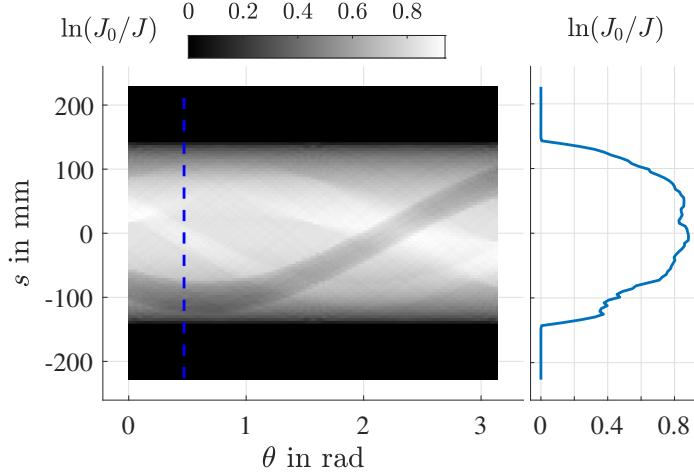
Um die Algorithmen für die Bildrekonstruktion zu überprüfen, wird ein Körpermodell genutzt, das über mehrere Volumen mit verschiedenen Schwächungseigenschaften verfügt. In Abbildung 4.13 ist dieses Körpermodell dargestellt. Je dunkler der Grauton, desto geringer ist der gespeicherte Absorptionskoeffizient. Schwarz entspricht einer Region ohne Schwächung, Graue Regionen entsprechen dem Material Wasser und weiße Regionen entsprechen Gewebe mit ungefähr 333 HU. Das Körpermodell ist zusammengesetzt aus  $N_x = 64$ ,  $N_y = 64$  und  $N_z = 64$  Voxeln, die jeweils eine Größe von  $\Delta x = 5$  mm,  $\Delta y = 5$  mm und  $\Delta z = 5$  mm haben. In der Abbildung sind zusätzlich die Geraden, entlang derer die Projektionen für einen bestimmten Winkel gemessen werden, dargestellt.



**Abbildung 4.13.:** Schnitt durch das Körpermodell zur Erprobung der Rekonstruktionsalgorithmen. In Blau dargestellt sind die Geraden, entlang derer für einen Winkel Projektionen aufgenommen werden

Für das Körpermodell werden für die in Abbildung 4.13 zu sehende Schnittebene die Projektionen aufgenommen. Es werden unter 180 Projektionswinkeln jeweils 108 Projektionen mit einer Auflösung von 4,23 mm gemessen. Es wird jeweils ein Strahl pro Detektorpixel simuliert. Für diese Aufnahme sind Streuung und energieabhängige Schwächung deaktiviert. In Abbildung 4.14 ist das Sinogramm, das die Projektionsdaten zeigt, abgebildet. Die Grau-

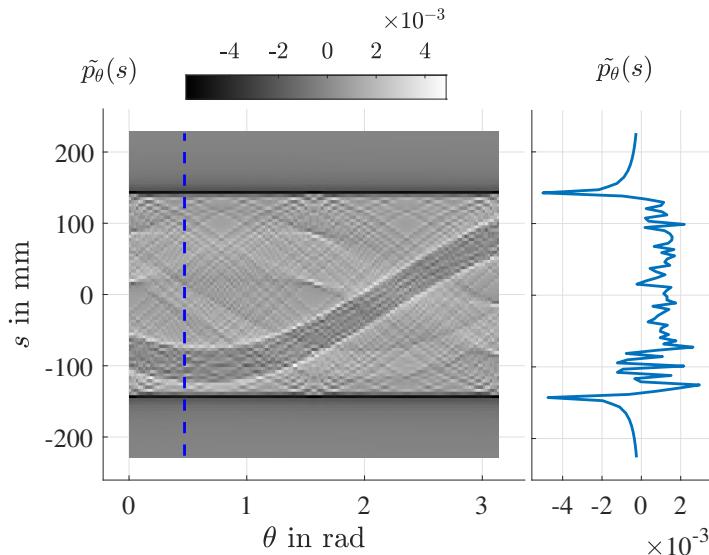
werte sind die Werte der Linienintegrale. Ist der Wert einer Linienintegrals Null, befindet sich entlang der Geraden kein Voxel, der Strahlung schwächt. Je heller ein Pixel im Sinogramm, desto mehr wurde ein Strahl geschwächt. Auf der blauen Linie liegen jene Projektionswerte, die zu den Geraden in Abbildung 4.13 korrespondieren. Auf der rechten Seite sind diese Projektionswerte als Kurve dargestellt. Im Sinogramm sind die typischen sinusförmigen



**Abbildung 4.14.:** Aufgenommene Projektionsdaten eines Schnittbildes. In Blau sind die Projektionswerte für einen Winkel dargestellt

Kurven, die je einem der Objekte aus dem Körpermodell zugeordnet werden können, zu sehen. Die schwarzen Ränder stellen den Hintergrund mit  $\mu = 0 \frac{1}{\text{mm}}$ . Die graue Region, die für größere Abstände  $s$  vom Ursprung dunkler wird, korrespondiert mit der großen, runden Region mit  $\mu = 3 \cdot 10^{-3} \frac{1}{\text{mm}}$ . Sie wird nach außen dunkler, da der Weg der Strahlen, die weiter entfernt vom Ursprung sind, kürzer ist.

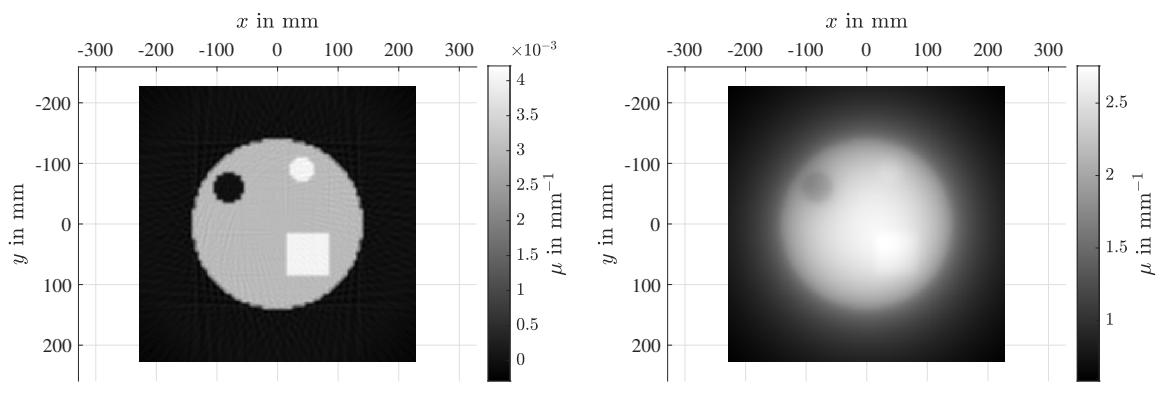
Zur Rekonstruktion werden die Projektionen gefiltert. Das geschieht jeweils durch Faltung der Projektionsdaten für einen Winkel entlang des Abstandes  $s$ . In Abbildung 4.15 sind die gefilterten Projektionen dargestellt. Auf der rechten Seite ist die Faltung der Projektionen-



**Abbildung 4.15.:** Gefilterte Projektionen. In Blau sind die gefilterten Projektionswerte für einen Winkel dargestellt

werte mit dem Filterkern zu sehen. Die Projektionswerte sind jene auf der rechten Seite in Abbildung 4.14. Der Filterkern ist jener nach Ramachandran und Lakshminarayanan. Das Hochpassverhalten des Filterkerns ist darin erkennbar, dass die Kanten hervorgehoben werden. Außerdem weisen Regionen, in denen die Projektionswerte homogen sind, nach Faltung einen Wert von Null auf.

Nach Gleichung 2.63 (S. 21) werden zur Rekonstruktion für jeden Pixel im rekonstruierten Bild die gefilterten Projektionsdaten der Geraden, die den jeweiligen Pixel „treffen“, aufsummiert. Den rekonstruierten Bildern in Abbildung 4.16 liegen die Projektionen und gefilterten Projektionen aus den Abbildungen 4.14 und 4.15 zu Grunde. In Abbildung 4.16.b ist das rekonstruierte Bild, das entsteht, wenn die ungefilterten Projektionen rückprojiziert werden, zu sehen. In Abbildung 4.16.a ist das Bild, das über die gefilterte Rückprojektion erzeugt wird, abgebildet. Durch die gefilterte Rückprojektion entsteht ein rekonstruiertes Bild,



**4.16.a:** Gefilterte Rückprojektion

**4.16.b:** Ungefilterte Rückprojektion

**Abbildung 4.16.:** Vergleich von ungefilterter und gefilterter Rückprojektion

das dem Schnitt aus dem Körpermodell aus Abbildung 4.13 entspricht. Die unterschiedlichen Regionen korrespondieren bezüglich ihrer Position und Form dem Schnitt durch das Körpermodell. Ebenfalls stimmen die berechneten Koeffizienten überein. Das rekonstruierte Bild und das ideale Schnittbild stimmen jedoch nicht vollständig überein. Innerhalb der im Körpermodell homogenen Regionen sind im rekonstruierten Bild Heterogenitäten sichtbar. Ebenfalls sind die scharfen Kanten der Voxel im rekonstruierten Bild verschmiert. Grund dafür ist die diskrete Abtastung der Projektionen, die nicht fein genug ist, um die Ränder scharf aufzulösen. Das in Abbildung 4.16.b rekonstruierte Bild zeigt die Notwendigkeit der Filterung von Projektionsdaten. In diesem Bild sind die einzelnen Regionen unterschiedlicher Absorptionseigenschaften ineinander verschmiert und die berechneten Koeffizienten um drei Größenordnungen zu den Koeffizienten der Voxel verschieden.

## 4.5. Gesamte Simulation

Zur Erprobung des Simulators in seiner Gesamtheit werden mehrere Schnittbilder mit verschiedenen Eingabeparametern aufgenommen. Dazu werden die grafische Benutzeroberfläche genutzt und jeweils die rekonstruierten Schnittbilder abgebildet. Dabei stellen die Grauwerte

die berechneten Schwächungskoeffizienten dar. Die Grauwerte, die zwischen 0 und 255 liegen, werden dazu einem Wertebereich der Schwächungskoeffizienten  $\mu_{\text{Schwarz}}$  zugeordnet  $\mu_{\text{Weiß}}$ . Der Grauwert ist dann mit Gleichung 4.5 gegeben.

$$\text{GW}(\mu) = 255 \cdot \frac{\mu - \mu_{\text{Schwarz}}}{\mu_{\text{Weiß}} - \mu_{\text{Schwarz}}} \quad (4.5)$$

Durch Auswahl des Wertebereiches über  $\mu_{\text{Schwarz}}$  und  $\mu_{\text{Weiß}}$  sowie Begrenzung der Grauwerte auf  $[0, 255]$  besteht die Möglichkeit den Kontrast des rekonstruierten Bildes einzustellen. Liegt  $\mu$  unter  $\mu_{\text{Schwarz}}$  oder über  $\mu_{\text{Weiß}}$  werden diese Bildpunkte vollständig schwarz beziehungsweise weiß angezeigt.

Wenn nicht anders beschrieben werden folgende Parameter für die Simulation genutzt:

**Anzahl Projektionen und Distanzen**

$$N_\theta = 400, N_s = 256$$

**Messfeldgröße**

$$s_{\text{Mess}} = 320 \text{ mm}$$

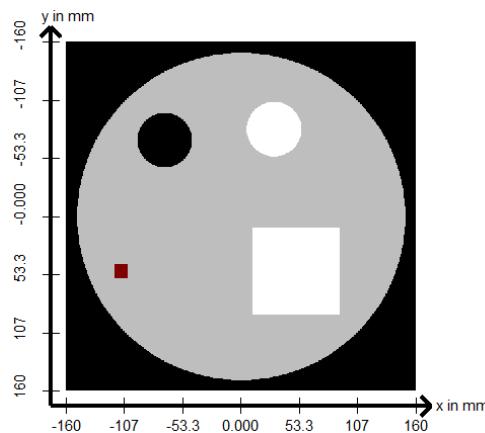
**Röhreneigenschaften**

$$U_A = 150 \text{ kV}, I_A = 0,2 \text{ A}, Z = 74 \text{ (Wolfram)}$$

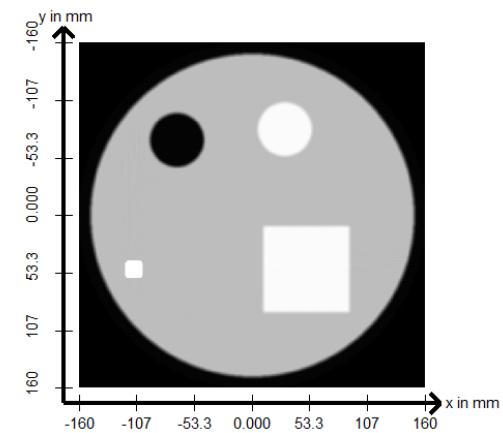
**Strahlen pro Pixel**

$$N_r = 10$$

In Abbildung 4.17.a ist der Schnitt durch das Körpermodell, in dessen Ebene alle Projektionen in diesem Abschnitt aufgenommen werden, abgebildet. Das Körpermodell verfügt innerhalb der Schnittebene über  $256 \times 256$  Voxel mit einer Größe von  $0,625 \text{ mm} \times 0,625 \text{ mm}$ . Der rote Punkt stellt Voxel, die über die Metalleigenschaft verfügen dar. Ihre Absorptionskoeffizienten betragen  $\mu_{\text{Titan}} = 13,5 \cdot 10^{-3} \frac{1}{\text{mm}}$ . Schwarze Regionen absorbieren mit  $\mu = 0 \frac{1}{\text{mm}}$  gar nicht, graue Regionen mit  $\mu = 3 \cdot 10^{-3} \frac{1}{\text{mm}}$  wie Wasser und weiße Regionen mit  $\mu = 4 \cdot 10^{-3} \frac{1}{\text{mm}}$  ähnlich wie Knochen. In Abbildung 4.17.b ist das rekonstruierte Bild dargestellt, wenn Streuung



4.17.a: Schnitt durch das Körpermodell



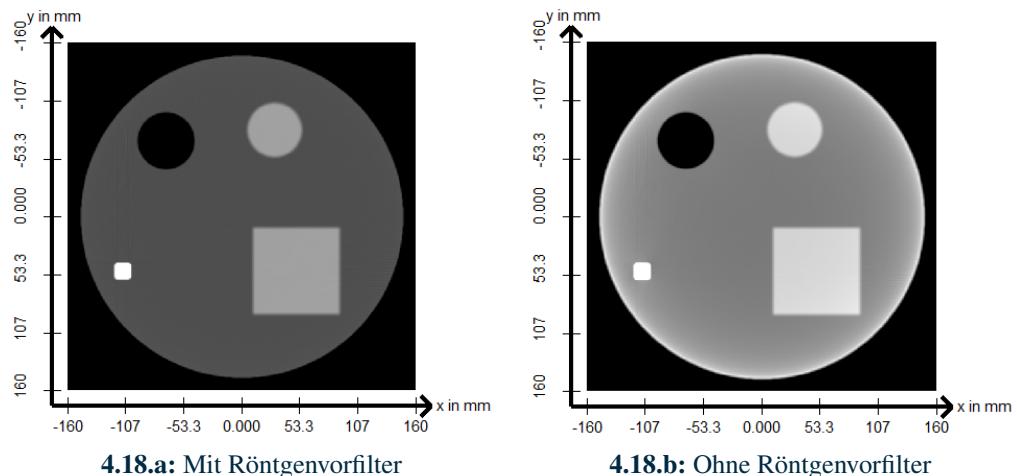
4.17.b: Rekonstruiertes Bild ohne Streuung und energieabhängige Schwächung

**Abbildung 4.17.:** Ideales Schnittbild aus den Körpermodelldaten und rekonstruiertes Bild durch Aufnahme von Projektionen. Schwarz:  $\mu = 0 \frac{1}{\text{mm}}$ . Weiß:  $\mu = 4 \cdot 10^{-3} \frac{1}{\text{mm}}$

und energieabhängige Absorption deaktiviert sind. Die berechneten Werte im rekonstruierten Bild (Abb. 4.17.b) entsprechen genau den Absorptionskoeffizienten der Volumina im Körpermodell. Die metallene Region wird durch die Begrenzung des Wertebereiches auf

$\mu \in [0 \frac{1}{\text{mm}}, 4 \cdot 10^{-3} \frac{1}{\text{mm}}]$  gleich dargestellt wie Strukturen aus Knochen. Dieses rekonstruierte Bild stellt den Idealfall dar. Die verschiedenen Regionen werden bezüglich ihrer Geometrie und Eigenschaften genau abgebildet. Einzig die Ränder sind unscharf, da die Auflösung mit 0,8 mm nicht jener des Körpermodells entspricht und mehrere Strahlen pro Pixel simuliert werden.

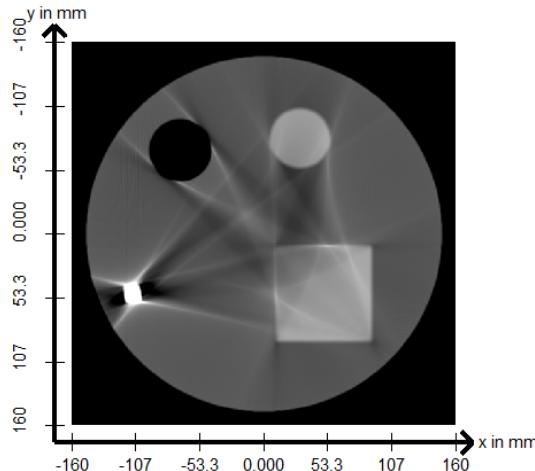
In Abbildung 4.18 wurde mit realem Spektrum simuliert. In Abbildung 4.18.a ist der Röntgenvorfilter aktiviert. In Abbildung 4.18.b ist kein Filter vorhanden. In dieser Abbildung sind deutliche Cupping-Artefakte erkennbar. Der Kontrast des Bildes ist so eingestellt, dass die Artefakte hervorgehoben werden. Während in Abbildung 4.18.a die berechneten Absorptionskoeffizienten wegen des Filters wie in Abbildung 4.17.b dem Körpermodell sehr genau entsprechen, weichen die rekonstruierten Koeffizienten in Abbildung 4.18.b von jenen des Körpermodells ab.



**Abbildung 4.18.:** Auswirkung des Röntgenvorfilters auf das rekonstruierte Bild. Wertebereich  $\mu_{\text{Schwarz}} = 2 \cdot 10^{-3} \frac{1}{\text{mm}}$  und  $\mu_{\text{Weiß}} = 5 \cdot 10^{-3} \frac{1}{\text{mm}}$

Detektorpixel sind in der Realität, wie in Abschnitt 3.7 (S. 77) beschrieben, nicht ideal. Ihre Sensitivität ist limitiert, da keine beliebig kleinen Intensitäten gemessen werden können. Durch Begrenzen der Projektionswerte auf  $p_{\max}$  kann das im Simulator veranschaulicht werden. In Abbildung 4.19 ist  $p_{\max}$  willkürlich auf 0,95 festgelegt. Das impliziert, dass alle detektierten Strahlen, deren Intensität weniger als 38,7 % der Ausgangsintensität beträgt, nicht voneinander unterschieden werden können. In der Abbildung sind deutliche Streifenartefakte und schwarze Schatten zwischen den verschiedenen Regionen sichtbar. Die Bereiche, in denen diese Artefakte auftreten, lassen sich bestimmten Regionen im Sinogramm zuordnen. Nämlich jenen, in denen sich die Sinuskurven der einzelnen Objekte schneiden und somit für Werte größer als  $p_{\max}$  sorgen. Durch die Begrenzung ist innerhalb dieser lokalen Bereiche der Projektionswert konstant. Die Ränder dieser lokalen Bereiche sind den weißen Streifen und das Innere dieser Bereiche den Schatten im rekonstruierten Bild zugeordnet. Besonders am metallenen Objekt unten links im Bild sind diese Artefakte sichtbar.

Um die weißen Streifen und schwarzen Schatten erklären zu können, muss die Filterung

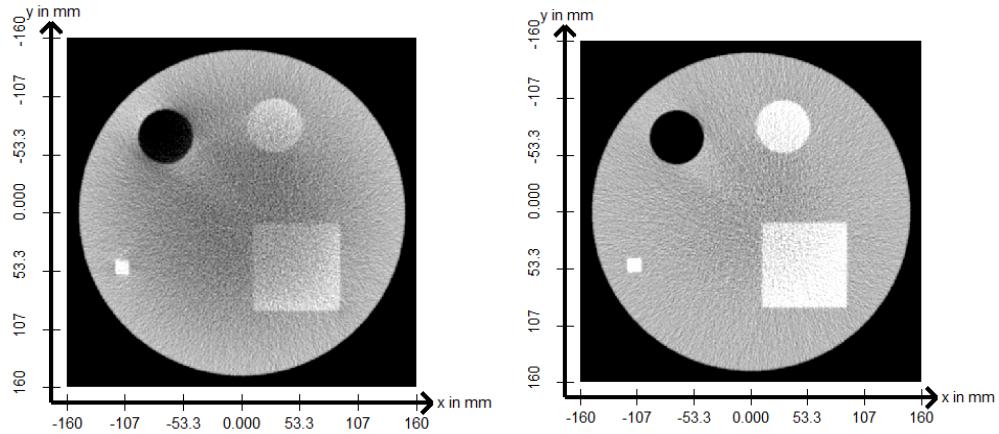


**Abbildung 4.19.:** Streifenartefakte durch begrenzte Detektorsensitivität. Wertebereich  $\mu_{\text{Schwarz}} = 2 \cdot 10^{-3} \frac{1}{\text{mm}}$  und  $\mu_{\text{Weiß}} = 5 \cdot 10^{-3} \frac{1}{\text{mm}}$

betrachtet werden. Im Sinogramm liegen durch die Begrenzung der Projektionswerte lokale Regionen mit konstantem Projektionswert vor. Die Filterung verfügt über einen Hochpasscharakter. Die im Sinogramm benachbarten Werte innerhalb der lokalen Regionen haben nach Filterung einen Wert, der geringer ist, als ohne die Begrenzung. Dieser eigentlich zu geringe Wert sorgt bei der Rückprojektion für die Schatten, die durch Projektionen aus anderen Richtungen nicht kompensiert werden. Die weißen Streifen können ebenfalls durch genauere Betrachtung der Filterung begründet werden. Der genutzte Filterkern verfügt neben einem positivem Peak in seiner Mitte über jeweils einen negativen Peak als Nachbarn. Dadurch werden bei der Filterung der unbegrenzten Projektionen die Sinuskurven von einem Bereich mit negativen Werten umschlossen. Dieser Bereich geht bei der Begrenzung verloren, sodass an den Rändern der Sinuskurven, welche genau den weißen Streifen im rekonstruierten Bild entsprechen, negative Werte fehlen. Diese würden im Normalfall die weißen Streifen kompensieren, sodass die Regionen zwischen den stark absorbierenden Bereichen homogen wären.

#### 4.5.1. Streuung

Wird Streuung aktiviert, ändert sich das Aussehen des rekonstruierten Bildes. In Abbildung 4.20 sind die rekonstruierten Bilder zu sehen, wenn dieselben Aufnahmeparameter wie im vorherigen Abschnitt gegeben sind und Streuung aktiviert ist. Die Streuung ist so implementiert, dass jede Energie im Spektrum in die gleiche Anzahl Teile zerlegt wird. Für jeden dieser Teile wird zufällig festgelegt, ob Streuung stattfindet. Für diese Aufnahmen werden 83 diskrete Energien im Spektrum und jeweils 77 Teile je Energie simuliert. In Abbildung 4.20.a ist kein Streustrahlungsraster vorhanden. In Abbildung 4.20.b werden alle Strahlen, deren Winkel zur Pixelnormalen über  $0,5^\circ$  liegt, nicht detektiert. Ist Streuung aktiviert, handelt es sich bei den berechneten Koeffizienten um Schwächungskoeffizienten, da die Schwächung durch Streuung hinzukommt. Daher ist ebenfalls der Wertebereich der Koeffizienten verschieden zu jenem der Messungen aus dem vorherigen Abschnitt. Nur mit Streustrahlenraster sind die berechneten Schwächungskoeffizienten der verschiedenen Regionen annähernd korrekt. Ohne



4.20.a: Kein Streustrahlungsraster

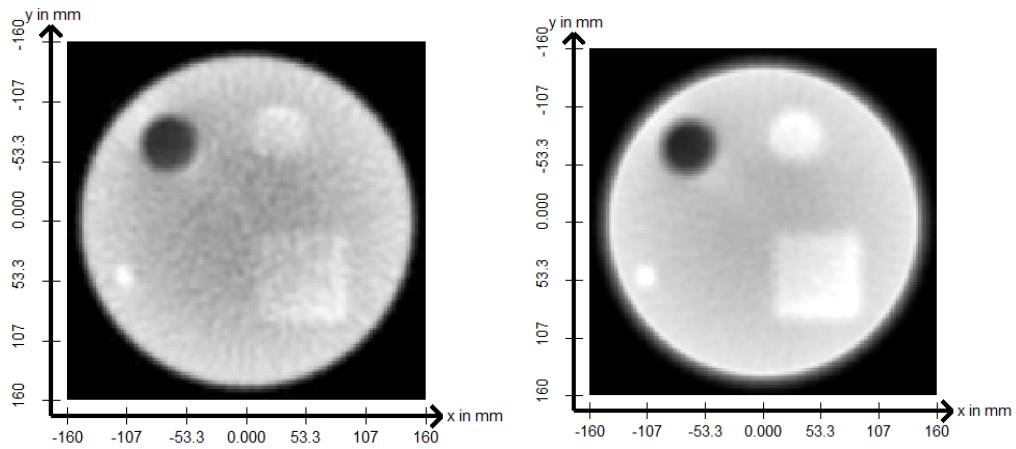
4.20.b: Mit Streustrahlungsraster mit einem Akzeptanzwinkel von  $0,5^\circ$ 

**Abbildung 4.20.:** Simulation von Streuung und Effekt des Streustrahlenrasters. Wertebereich  $\mu_{\text{Schwarz}} = 5 \cdot 10^{-3} \frac{1}{\text{mm}}$  und  $\mu_{\text{Weiß}} = 25 \cdot 10^{-3} \frac{1}{\text{mm}}$

Streustrahlungsraster sind sowohl die einzelnen Regionen inhomogener als auch quantitativ falschen Schwächungskoeffizienten zugeordnet. Insgesamt ist Abbildung 4.20.a dunkler, da die Intensität der detektierten Sekundärstrahlung größer ist. Dadurch wird die Schwächung unterschätzt.

### Simulationsqualität

Die Auflösung verschiedener Vorgänge während der Simulation bestimmt über deren Nähe zur Realität und die notwendige Rechenzeit. Davon unabhängig sind jene Eigenschaften wie die Anzahl der Projektionswinkel und Anzahl der Projektionen je Winkel. Diese sind gerätespezifisch. Besonders die Anzahl der Energien im Spektrum und die Anzahl der Teile, in die die Photonenflüsse einer Energie für die Streuung zerlegt werden, spielen eine große Rolle. In Abbildung 4.21 sind für zwei verschiedene Simulationsqualitäten Schnittbilder aufgenommen worden. Die Anzahlen der Projektionswinkel und der Projektionen je Winkel



4.21.a: Geringe Simulationsqualität

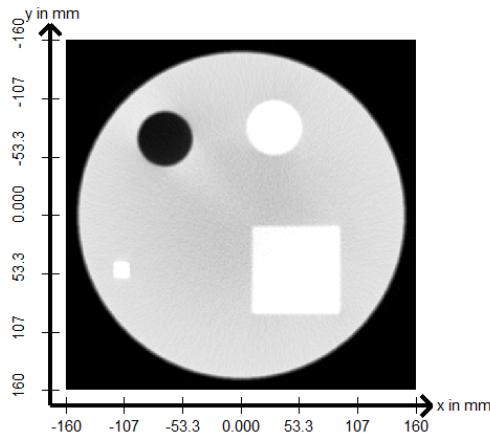
4.21.b: Hohe Simulationsqualität

**Abbildung 4.21.:** Vergleich von geringer und hoher Simulationsqualität. Wertebereich  $\mu_{\text{Schwarz}} = 0 \frac{1}{\text{mm}}$  und  $\mu_{\text{Weiß}} = 22 \cdot 10^{-3} \frac{1}{\text{mm}}$

sind für diese Simulationen auf  $N_\theta = 96$  und  $N_s = 64$  reduziert worden, da die Rechenzeit für

Abbildung 4.21.b sehr groß gewesen wäre. Abbildung 4.21.a wurde aufgenommen mit 68 diskreten Energien im Spektrum und 62 Teilen je Energie. Für jeden Voxel, den ein Strahl trifft, wird 4216-mal auf das Eintreten eines Streuereignis überprüft. Für Abbildung 4.21.b sind es  $158 \cdot 152 = 24016$  mögliche Streuereignisse. Bei feinerer Einteilung ist die Anzahl der gestreuten Strahlen größer. Die gestreuten Strahlen verfügen dabei jedoch über eine geringere Leistung. Je feiner die Einteilung, desto realitätsnäher wird die Streuung simuliert. Ist die Einteilung zu grob, ist der Effekt von Streustrahlung in Abbildung 4.21.a gegenüber der Realität sehr groß.

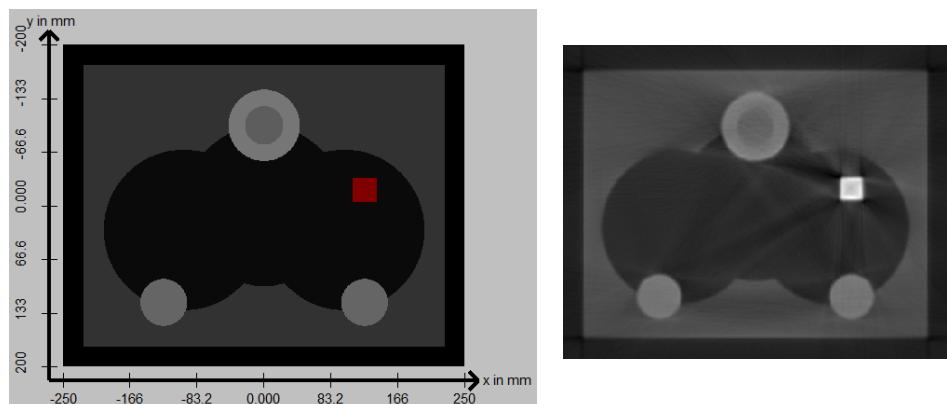
In Abbildung 4.22 ist eine Aufnahme abgebildet mit der Simulationsqualität wie Abbildung 4.21.b und den Geräteeigenschaften  $N_\theta = 400$  und  $N_s = 256$ .



**Abbildung 4.22.:** Aufnahme mit hoher Simulationsqualität und -auflösung. Wertebereich  $\mu_{\text{Schwarz}} = 0 \frac{1}{\text{mm}}$  und  $\mu_{\text{Weiß}} = 22 \cdot 10^{-3} \frac{1}{\text{mm}}$

### Komplexeres Körpermodell

In Abbildung 4.23 ist die Aufnahme eines Schnittbildes durch ein Körpermodell mit mehreren Strukturen zu sehen. Die Strukturen entsprechen mit ihren Schwächungseigenschaften jenen von Gewebe. Das Körpermodell ist hochauflösend und die Simulationsqualität sehr groß gewählt. Das Modell eignet sich besonders gut, um die implementierten Artefakte darzustellen.



**4.23.a:** Schnittbild aus Voxeldaten

**4.23.b:** Rekonstruiertes Bild

**Abbildung 4.23.:** Simulation mit einem Körpermodell aus vielen Strukturen. Schwarze Regionen: Luft. Hellgraue Regionen: Knochen. Dunkelgrau: Wasser



# 5. Diskussion und Fazit

In dieser Arbeit wurde ein Simulator für die Computertomographie entwickelt. Es ist das Ziel, die physikalische Realität sowie technische Aspekte der Computertomographie abzubilden und den Simulator als Werkzeug für die Lehre zu gestalten.

## 5.1. Bewertung der Implementierung

Herzstück des Simulators bilden die Abbildung von physikalischen Modellen und Gerätekomponenten sowie die grafische Benutzeroberfläche.

### 5.1.1. Physikalische Modelle

#### Röntgenspektrum

Das Spektrum der Röntgenröhre ist als diskrete Zuordnung von Photonenflüssen  $n(E)$  in  $\frac{1}{\text{s}}$  zu einer Photonenergie  $E$  implementiert. Quantitativ beschreibt das Spektrum eine Leistung beziehungsweise einen Energiefluss für jede diskrete Photonenergie. Die Form des Spektrums ist maßgeblich von den Röhreneigenschaften bestimmt. Dabei wird nur das kontinuierliche Spektrum der Bremsstrahlung berücksichtigt. Außerdem ist die Form durch einen linearen Zusammenhang von Photonenfluss und Energie angenähert. Diese Implementierung entspricht den Modellen aus [4, S. 9]. Die Gesamtleistung des Spektrums entspricht [4, S. 11 Gl. 2.4]. Was das implementierte Röhrenspektrum nicht abbildet, ist die räumliche Verteilung von Photonen. In dieser Arbeit wurden daher die Strahlungsleistung und die Strahlungsintensität als synonyme Begriffe verwendet. Die Strahlungsintensität jedoch bezieht die Strahlungsleistung auf eine durchstrahlte Fläche [19, S. 587].

Insgesamt ermöglicht die Implementierung, dass Strahlaufhärzungartefakte abgebildet werden können. Weiterhin spielen die Röhreneigenschaften eine Rolle in deren Ausprägung. Da vor allem das Verhältnis der Strahlungsleistungen oder -intensitäten verschiedener Photonenergien zueinander relevant für die physikalischen Effekte wie Absorption und Streuung ist, werden diese zufriedenstellend abgebildet.

#### Strahlen

Röntgenstrahlung ist als Nadelstrahlen mit bestimmten Eigenschaften implementiert. Zu diesen Eigenschaften zählt jeweils ein, wie im vorherigen Abschnitt beschriebenes, Spektrum. Nadelstrahlen lassen sich durch das Körpermodell verfolgen und entsprechend der Voxel-eigenschaften in ihren Eigenschaften verändern. Im Prinzip simuliert ein Nadelstrahl eine große Menge an Photonen verschiedener Energien, die sich auf identischer Trajektorie bewegen. Nadelstrahlen entsprechen dem mathematischen Ansatz, Linienintegrale zu bestimmen. Dass sich jedoch alle Photonen entlang einer Geraden bewegen, entspricht nicht den physikalischen Modellen. Im Prinzip müsste jedes Photon einzeln simuliert werden, um diesen Modellen gleichzukommen. Für den Zweck dieses Simulators ist die Simulation von Strahlung als Nadelstrahlen ausreichend, um die meisten relevanten Aspekte der Computertomographie abzubilden. Die Ausnahme davon bilden Teilvolumenartefakte, die voraussetzen, dass ein Pixel

Photonenbündel, die räumlich ausgedehnt sind, detektiert. Nur durch Simulation mehrerer Strahlen je Pixel besteht die Möglichkeit diese Artefakte abzubilden.

### Schwächung von Strahlung

Strahlen, die durch das Körpermodell verlaufen, werden für jeden einzelnen Voxel geschwächt. Die Schwächung entspricht dabei einer Verminderung der Photonenflüsse im Spektrum und ist entsprechend des Lambert-Beerschen Gesetzes abhängig von der Strecke, die ein Strahl in einem Voxel verbringt. Dass die Schwächung von Photonen energieabhängig ist, ist im Simulator durch die Berechnung der Schwächung für die diskreten Energien im Strahlungsspektrum umgesetzt. Die Eigenschaft, die Körpermodelle für jedes Voxel gespeichert haben, ist ein Absorptionskoeffizient. Aus diesem leiten sich alle Schwächungseigenschaften einzelner Voxel ab. In der Realität sind es jedoch chemische Elemente und Verbindungen, die das Schwächungsverhalten bestimmen. Im Simulator sind nur der Photoeffekt und die Compton-Streuung als Wechselwirkungsprozesse mit dem Körpermodell abgebildet. Diese Effekte haben den größten Anteil an der Schwächung in dem Energiebereich, der für die Computertomographie genutzt wird.

Absorption<sup>1</sup> entsteht durch den Photoeffekt und die Compton-Absorption. Beide Effekte hängen in ihrem Absorptionsvermögen von der Photonenenergie ab. Sie werden gemeinsam durch eine Näherung, die die Abhängigkeit  $\mu_{\text{abs}} \propto E^{-3}$  für geringe Energien und Konstanz für größerer Energien darstellt, abgebildet. Grundsätzlich existieren physikalische Modelle, die es ermöglichen für beliebige Materialien die Absorption exakt zu beschreiben [11, S. 272]. Der hier gewählte einfache Ansatz liefert jedoch im Kontext der Computertomographie Ergebnisse, die dem Zweck des Simulators genügen. Konkret sind es primär die Strahlaufrärtungsefekte, die durch Implementierung des energieabhängigen Koeffizienten gezeigt werden können.

Die Compton-Streuung ist jener Aspekt, der im Simulator am schwierigsten abbildbar ist. Grund dafür ist, dass Streuung probabilistisch für jedes einzelne Photon beschrieben werden müsste. Da dieses Vorgehen jedoch wegen der hohen Rechenzeit impraktikabel ist, müssen große Mengen Photonen zusammengefasst werden. Das Verfahren ist ähnlich zu Monte-Carlo Methoden [4, S. 19–20]. Die Konsequenz ist jedoch, dass bei geringer Simulationsqualität (Abschnitt 4.5.1 S. 100) das Auftreten eines Streuereignisses, das viele Photonen betrifft, die Projektionsdaten in großem Maße beeinflusst. Dadurch ist das rekonstruierte Bild stark zu jenem mit hoher Simualtionsqualität verändert. In der Implementierung ist es jedoch gelungen das zufällige Auftreten von Streuereignissen und den Streuwinkeln so abzubilden, das die Simulation diesbezüglich nahe an den physikalischen Modellen liegt. Die Ausnahme davon bildet die Tatsache, dass Strahlen nur innerhalb einer Ebene gestreut werden. Das hat jedoch, da der Detektor nur über eine Pixelzeile verfügt, keinen Einfluss auf die Messung der Projektionen.

---

<sup>1</sup>Damit ist tatsächlicher Energieverlust an Elektronen gemeint. Streuung impliziert nur eine Richtungsänderung.

### 5.1.2. Gerätekomponenten

#### Röntgenröhre

In der Implementierung stellt die Röntgenröhre die Quelle und den Eigenschaftengeber der Röntgenstrahlen dar. Die Strahlrichtungen sind jedoch alleinig von der relativen Postion der Detektorpixel zur Röntgenröhre bestimmt. Die Röhre ist somit eher ein Hilfsobjekt, das die Eigenschaften und Funktionen abbildet, aber keine Entsprechung zur realen Röntgenröhren hat. Weiterhin ist die Röntgenröhre als ideales Bauteil, das aus den Eingabeparametern ein diskretes Spektrum von Photonenflüssen erzeugt, implementiert. In Computertomographen verfügen Röntgenröhren zusätzlich über Eigenschaften, die die Form des Photonenbündels und des Brennflecks beeinflussen [4, S. 25]. Weiterhin spielt in der praktischen Anwendung die tatsächliche Photonenmenge und nicht der Photonenfluss die relevante Rolle. Die relevante Eigenschaft ist die Belichtung, die als Strom-Zeit-Produkt [34, S. 24] beschrieben werden kann. Diese Eigenschaft ist maßgebend für die Bildqualität, da Rauschartefakte unterdrückt werden können [34, S. 28]. Das Strom-Zeit-Produkt wird im Simulator nicht abgebildet.

Ein wichtiges Element zur Artefaktreduktion bei der Computertomographie ist ein Röntgenvorfilter zur initialen Strahlaufhärtung. Dieser ist in Grundzügen implementiert und wirkt sich erwartungsgemäß positiv auf die Bildqualität aus. Die Parameter, zu denen eine Grenzenergie und Flankensteilheit zählt, des Vorfilters sind frei wählbar. Dadurch kann jeder Filtercharakter abgebildet werden, auch solche, die nicht mit typischen Materialien wie Aluminium [4, S. 31] oder Kupfer [2, S. 29] möglich sind.

#### Röntgendetektor

Durch die Freiheit, die die Simulation bietet, ist es möglich einen idealen Detektor zu konstruieren. Ideal bedeutet in diesem Fall, dass die Anordnung der Detektorpixel exakt so ist, dass die zentrierten Pixelnormalen exakt zu Radon-Koordinaten, die in einem gewünschten Raster liegen, korrespondieren. In Computertomographen kann die Pixelanordnung nicht beliebig gewählt werden, sondern muss so sein, dass die Röntgenröhre im Schnittpunkt der Pixelnormalen liegt.

Im Simulator werden keine Eigenschaften der Detektorpixel abgebildet. Sie sind in der Lage kleinste Intensitäten zu detektieren und aufzulösen. Weiterhin verfügen sie über eine konstante spektrale Empfindlichkeit. Diese liegt nicht bei realen Detektoren vor, da grundsätzlich Absorptionsprozesse, die energieabhängig sind, Teil der Detektion sind. Ebenfalls sind reale Detektoren in ihrer Empfindlichkeit und Fähigkeit Quantenrauschen zu unterdrücken, begrenzt. Es existiert eine Grenze, unter welcher gemessene Intensitäten nicht vom Quantenrauschen unterschieden werden können. Diese Grenze wird durch die *Detective Quantum Efficiency* (DQE) [4, S. 69] charakterisiert. Vor allem die Größe der Pixel, die im Simulator keine Rolle bei Erzeugung des Messsignals spielt, beeinflusst durch eine unterschiedlich große Quantenzahl, die auf einen Pixel trifft, die DQE.

Ein weiterer Aspekt, der im Simulator idealisiert implementiert ist, ist die vollständige Absorption von Strahlung durch das Streustrahlungsraster. Da die Lamellen des Rasters in

realen Detektoren eine endliche Dicke haben, wird Streustrahlung nicht vollständig absorbiert.

### 5.1.3. Rekonstruktionsprozesse

Im Simulator werden Bilder durch die gefilterte Rückprojektion rekonstruiert. Dabei handelt es sich um eine Standardmethode, die sowohl mathematisch als auch in ihrer Implementierung einfach ist. Die gefilterte Rückprojektion ist durch Faltung mit drei wählbaren Filterkernen implementiert. Dabei entspricht ein Filterkern jenem, der keine Filterung vornimmt. Die beiden anderen Filterkerne sind Standardkerne. Die Implementierung der gefilterten Rückprojektion liefert korrekte Ergebnisse. Wird ein Schnittbild mit idealen physikalischen Modellen<sup>2</sup> aufgenommen, ist das rekonstruierte Bild dem Schnitt durch das Körpermodell entsprechend.

### 5.1.4. Benutzeroberfläche

Die Benutzeroberfläche liefert die Möglichkeit einer effizienten Interaktion mit dem Simulator. Die Benutzereingaben steuern das konkrete Verhalten des Simulators. Ergebnisse und Zwischenergebnisse werden angezeigt. Durch eingebettete Hinweise bezüglich verschiedener Ein- und Ausabeelemente ist die Benutzeroberfläche zu einem gewissen Maß intuitiv. Eingaben werden ebenfalls auf ihre Plausibilität geprüft und das Speichern und Laden aufgenommener Projektionen ermöglicht schnellen Zugriff auf aufgenommene Messungen.

Die genutzten Bibliotheken zur Erstellung der grafischen Benutzeroberfläche ermöglichen eine effiziente Entwicklung der Oberfläche und Erstellung eigener grafischer Elemente. Einzig marginal auftretende Renderartefakte oder eher konservative Ästhetik sind Mängel der Bibliotheken.

Die Anordnung und Größenverhältnisse der verschiedenen Oberflächenelemente sind nicht ideal. Eine andere räumliche Gliederung könnte die Benutzererfahrung verbessern. Außerdem ließen sich weitere Parameter, die aktuell konstant im Quelltext definiert sind, in der Oberfläche veränderbar machen. Bezuglich der Anzeige des rekonstruierten Bildes ist zu erwähnen, dass Computertomographen Hounsfield-Einheiten in einem Wertebereich, der  $2^{12} = 4096$  diskrete Grauwerte [1, S. 92] enthält, definieren. Die grafische Benutzeroberfläche erlaubt jedoch nur  $2^8 = 256$  diskrete Grauwerte.

### 5.1.5. Programmierung

Die Wahl einer objektorientierten Sprache zur Programmierung des Simulators erwies sich als vorteilhaft, da so Gerätekomponenten und komplexere mathematische Objekte sowie deren Operationen abgebildet werden konnten. Mit C++ als Sprache wird eine adäquate Simulationsgeschwindigkeit erreicht und kann auf effiziente Standardbibliotheken zurückgegriffen werden. Weiterhin existieren die notwendigen Bibliotheken für C++. Es ist gelungen, die Implementierung zu modularisieren und den Quelltext übersichtlich zu gestalten. Ebenfalls ist der Quelltext vollständig dokumentiert. Grundsätzlich wäre es für einige der implementierten Module möglich gewesen, bereits bestehende Bibliotheken zu nutzen, um die Entwicklungszeit zu verringern.

---

<sup>2</sup>Ideal heißt: keine energieabhängige Schwächung und keine Streuung

## 5.2. Bewertung des Nutzens

Er war das Ziel dieser Arbeit, einen Lehrsimulator für die Computertomographie zu entwickeln. In dem Simulator sollten ausgewählte Gerätekomponenten, physikalische Modelle und die Bildrekonstruktion abgebildet werden. Wie in Abschnitt 5.1 beschrieben, wurde dieses Ziel trotz Möglichkeiten zur Weiterentwicklung, erreicht. Bei diesem Simulator steht primär der didaktische Nutzen im Vordergrund. Besonders die Möglichkeiten, physikalische Effekte und die daraus resultierenden Artefakte in ihren Ausmaßen einstellen zu können, können eingesetzt werden, um diese in Lehrveranstaltungen zu erläutern. Wegen der technischen Unterschiede zu Computertomographen in der klinischen Anwendung und Ungenauigkeiten der physikalischen Modelle, ist eine Anwendung in der Lehre von radiologischem Personal kritisch zu bewerten. Eine Einbindung des Simulators in Lehrveranstaltungen jedoch ist denkbar. Eine selbstständige Beobachtung verschiedener Effekte und Artefakte sowie die Auswirkung von Geräteparametern könnte eine Methode zur Vermittlung von Lehrinhalten sein. Ebenfalls ist durch die transparente Darstellung der Rekonstruktionsprozesse und Nutzung einfacher Körpermodelle das Potenzial gegeben, diese Prozesse intuitiv nachzuvollziehen.

## 5.3. Ausblick

In den vorherigen Abschnitten wurde der Simulator bezüglich der physikalischen Modelle, den Gerätekomponenten und der Rekonstruktionsprozesse bewertet. Daraus ergeben sich konkrete Möglichkeiten zur Verbesserung und Erweiterung des Simulators. Das Ergebnis wäre eine Simulation, die der physikalischen und technischen Realität genauer entspräche.

Um der physikalischen Realität eher zu entsprechen, könnte in Körpermodellen die relative Zusammensetzung chemischer Verbindungen und deren Dichte gespeichert werden. Daraus ließen sich nach [35] analytisch die Wirkungsquerschnitte für die Wechselwirkungsprozesse<sup>3</sup> bestimmen. Aus den Wirkungsquerschnitten und den Materialdichten können die linearen und energieabhängigen Schwächungskoeffizienten sowie Wechselwirkungswahrscheinlichkeiten für Streuereignisse berechnet werden. Werden zusätzlich eine größere Anzahl von Strahlen beziehungsweise Photonenzahlmengen<sup>4</sup> simuliert, könnte die Schwächung, den physikalischen Modellen entsprechend, abgebildet werden.

Bei der Strahlerzeugung könnte neben der Bremsstrahlung abhängig vom Anodenmaterial charakteristische Strahlung simuliert werden. Diese hat einen nicht zu vernachlässigen Anteil an der Gesamtintensität eines Spektrums und damit auch Auswirkungen auf die Energieverhältnisse bei Schwächung, was Einfluss auf Artefakte<sup>5</sup> hat. Implementiert ist ein Spektrum als diskrete Photonenzahlmengen. Stattdessen könnte ein Strahlenspektrum als Photonenzahlmengen, die sowohl von Röhreneigenschaften als auch von der Belichtungszeit bestimmt sind, beschrieben werden. Das ermöglicht ebenfalls die Betrachtung spektraler Empfindlichkeiten eines Detektorpixels und Einbezug von Eigenschaften wie der DQE. Für Röntgenvorfilter könnten

---

<sup>3</sup>Photoelektrische Absorption, Compton-Streuung, Rayleigh-Streuung, Paar-Erzeugung

<sup>4</sup>Das bezieht sich auf den hier implementierten Algorithmus für die Streuung, in dem das Spektrum in Gruppen gleichenergetischer Photonen geteilt wird.

<sup>5</sup>Hier sind vor allem Strahlauflaufhärtingsartefakte gemeint.

konkrete Voreinstellungen für zum Beispiel Aluminium oder Kupfer [2, S. 29] getroffen werden. Es ist denkbar, dass durch Angabe eines Materials und der Filterdicke beliebige Vorfilter simuliert werden könnten. Insgesamt könnte sich bezüglich der Gerätekomponenten an realen Computertomographen orientiert werden. Das bezieht sich sowohl auf die Detektorkonstruktion beziehungsweise Pixelanordnung als auch die Eigenschaften der Röntgenröhre. Konkret ließen sich zum Beispiel Computertomographen bestimmter Generation konstruieren. Bezuglich der Bildrekonstruktion könnten statt der gefilterten Rückprojektion andere Rekonstruktionsmethoden implementiert werden. Die gefilterte Rückprojektion ist vom Aufwand der Implementierung und notwendiger Rechenzeit eine Standardmethode. Dieser gegenüber stehen iterative oder statistische Verfahren, die für eine Reduktion von Artefakten sorgen können [2, S. 201]. Weiterhin ist die Implementierung von Korrekturalgorithmen wie der Wasservorkorrektur denkbar, um deren Wirkung darzustellen.

Zur Verbesserung der Benutzerfreundlichkeit könnte eine vollständige Erneuerung der grafischen Benutzeroberfläche führen. Das könnte vor allem von Vorteil sein, da die jetzige Benutzeroberfläche ohne vorherige detaillierte Konzeptionierung gewachsen ist.

Es ist deutlich, dass trotz umfangreicher Implementierung von Gerätekomponenten, physikalischen Modellen und Rekonstruktionsprozessen vielerlei Möglichkeiten zu Verbesserung des Simulators bestehen.

# Literaturverzeichnis

- [1] Thomas M. Buzug und Thomas Flohr. „Computertomographie“. In: *Medizinische Bildgebung*. Hrsg. von Olaf Dössel und Thomas M. Buzug. 7. Aufl. Berlin/Boston: de Gruyter, 2014, S. 59–111.
- [2] Thorsten M. Buzug. *Computed Tomography*. Springer Verlag, 2008.
- [3] *Umweltradioaktivität und Strahlenbelastung: Jahresbericht 2020 (Parlamentsbericht)*. Techn. Ber. 2023. URL: <http://nbn-resolving.de/urn:nbn:de:0221-2023092039276>.
- [4] Olaf Dössel. *Bildgebende Verfahren in der Medizin*. 2. Aufl. Springer Vieweg, 2016.
- [5] Siemens Healthcare GmbH. *Siemens SOMATOM Definition AS Brochure*. Techn. Ber. CC CT 1373 03161. Erlangen, DE, 2016. URL: [https://marketing.webassets.siemens-healthineers.com/1800000000032845/f98fd15f1a4c/ct\\_somatom\\_definition\\_as\\_brochure\\_1800000000032845.pdf](https://marketing.webassets.siemens-healthineers.com/1800000000032845/f98fd15f1a4c/ct_somatom_definition_as_brochure_1800000000032845.pdf).
- [6] *CT Simulator*. URL: <https://www.acionline.com/ct-simulator.cms> (besucht am 23.04.2024).
- [7] *ScanLabCT*. URL: <https://scanlabmr.com/scanlabct> (besucht am 23.04.2024).
- [8] *CT - Simulation*. URL: [https://www.didaktik.physik.uni-muenchen.de/archiv/inhalt\\_materialien/ctsim/index.html](https://www.didaktik.physik.uni-muenchen.de/archiv/inhalt_materialien/ctsim/index.html) (besucht am 23.04.2024).
- [9] O. Haxel. „Entstehung, Eigenschaften und Wirkungen ionisierender Strahlen“. In: *Physikalische Grundlagen und Technik*. Hrsg. von H. Vieten. Handbuch der medizinischen Radiologie. Springer Verlag, 1968. Kap. A.
- [10] „Die Elemente und die Chemie“. In: *Gerthsen Physik*. Hrsg. von Dieter Meschede. 25. Aufl. Springer-Verlag, 2015. Kap. 17.
- [11] R. D. Evans. „Compton Effect“. In: *Corpuscles and Radiation in Matter II*. Hrsg. von S. Flügge. Bd. XXXIV. Encyclopedia of Physics. Springer Verlag, 1958. DOI: [10.1007/978-3-642-45898-9](https://doi.org/10.1007/978-3-642-45898-9).
- [12] Johann Radon. „Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten“. In: *Berichte über die Verhandlungen der Sächsische Akademie der Wissenschaften* 69 (1917), S. 262–277. DOI: [10.1093/jicaru/\\_ndr011](https://doi.org/10.1093/jicaru/_ndr011).
- [13] Avinash C. Kak und Malcolm Slaney. „Algorithms for Reconstruction with Nondiffracting Sources“. In: *Principles of Computerized Tomographic Imaging*. Society for Industrial and Applied Mathematics, 2001. Kap. 3, S. 49–112. DOI: [10.1137/1.9780898719277](https://doi.org/10.1137/1.9780898719277).

- [14] Marc Kachelrieß. „Computertomographie“. In: *Medizinische Physik. Grundlagen – Bildgebung – Therapie – Technik*. Hrsg. von S. Schlegel, C. P. Karger und O. Jäkel. Springer Spektrum Berlin, Heidelberg, 2018. Kap. 8, S. 153–203. DOI: 10.1007/978-3-662-54801-1.
- [16] Martin Meyer. *Signalverarbeitung*. 8. Aufl. Springer Vieweg, 2017. DOI: 10.1007/978-3-658-18321-9.
- [17] Avinash C. Kak und Malcolm Slaney. „Measurement of Projection Data - The Non-diffracting Case“. In: *Principles of Computerized Tomographic Imaging*. Society for Industrial und Applied Mathematics, 2001. Kap. 4, S. 113–175. DOI: 10.1137/1.9780898719277.
- [18] Frank Herold. „Die 10 Modellannahmen der Computertomographie“. In: *tm - Technisches Messen* 87.2 (2020), S. 93–100. DOI: doi:10.1515/teme-2019-0115.
- [19] Dieter Meschede, (Hrsg.). *Gerthsen Physik*. 25. Aufl. Springer-Verlag, 2015.
- [20] Ulrich Kaiser und Martin Guddat. *C/C++: Das umfassende Handbuch*. 5. Aufl. Galileo Press, 2014.
- [21] Rui Pereira u. a. „Ranking programming languages by energy efficiency“. In: *Science of Computer Programming* 205.102609 (2021). DOI: <https://doi.org/10.1016/j.scico.2021.102609>.
- [22] *The top programming languages*. 2022. URL: <https://octoverse.github.com/2022/top-programming-languages> (besucht am 05.02.2024).
- [23] *Fundamental types*. 2024. URL: <https://en.cppreference.com/w/cpp/language/types> (besucht am 01.02.2024).
- [24] Jens Gustedt. *Modern C*. Manning, Nov. 2019. URL: <https://inria.hal.science/hal-02383654>.
- [25] Wolfgang Werner. *Vektoren und Tensoren als universelle Sprache in Physik und Technik 1: Tensoralgebra und Tensoranalysis*. Springer Vieweg Wiesbaden, 2019. DOI: 10.1007/978-3-658-25272-4.
- [26] A. Kielbasinski und H. Schwetlick. „Numerische lineare Algebra“. In: *Mathematik für Naturwissenschaft und Technik*. Hrsg. von H. Heinrich und H. Schubert. Bd. 18. VEB Deutscher Verlag der Wissenschaften, 1988.
- [27] Lehrstuhl für Angewandte Mathematik. *3 Lineare Gleichungssysteme, direkte Verfahren*. 2024. URL: [https://angemath.unileoben.ac.at/fileadmin/shares/amat/docs/num1/ss23/SkriptS23Kap3\\_4.pdf](https://angemath.unileoben.ac.at/fileadmin/shares/amat/docs/num1/ss23/SkriptS23Kap3_4.pdf) (besucht am 02.02.2024).
- [29] *Information technology — Programming languages — C*. Standard ISO/IEC 9899:2018(E). Juli 2018.

- [30] *Information technology — Programming languages — C++*. Standard ISO-IEC 14882-2017. International Organization for Standardization, Dez. 2017.
- [31] David Blackman und Sebastiano Vigna. „Scrambled Linear Pseudorandom Number Generators“. In: *ACM Trans. Math. Softw.* 47.4 (Sep. 2021). DOI: 10.1145/3460772.
- [32] Sebastiano Vigna. *xoshiro/xoroshiro generators and the PRNG shootout*. URL: <https://prng.di.unimi.it/> (besucht am 13.03.2024).
- [33] Simon Hoffmann und Rainer Lienhart. „OpenMP. Eine Einführung in die parallele Programmierung mit C/C++“. In: *Informatik im Fokus*. Hrsg. von O. Günther u. a. Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-73123-8.
- [34] Paul Stoltzmann und Robert Götti. „Protokollparameter und Bildqualität“. In: *Wie Funktioniert Ct?* Hrsg. von Hatem Alkadhi u. a. Springer Verlag, 2011, S. 23–29.
- [35] Gladys White Grodstein. *Circular of the Bureau of Standards no. 583: x-ray attenuation from 10 kev to 100 Mev*. Techn. Ber. Gaithersburg, MD, 1957. DOI: 10.6028/NBS.CIRC.583.
- [36] Hans-Günter Schiele. *Computergrafik für Ingenieure*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-23843-7.

## Online-Bildquellen

- [15] Taschenrechner. *sampling by multiplication with Dirac comb*. 2005. URL: [https://upload.wikimedia.org/wikipedia/commons/6/6a/Dirac-comb\\_-\\_Sampling.png](https://upload.wikimedia.org/wikipedia/commons/6/6a/Dirac-comb_-_Sampling.png) (besucht am 16.01.2024).
- [28] Vossman und M. W. Toews. *A series of voxels in a stack with a single voxel shaded*. 2006. URL: <https://de.wikipedia.org/wiki/Datei:Voxels.svg> (besucht am 13.02.2024).



# A. Anhang

## A.1. Kompilieren und Benutzung des Simulators

Um den Simulator als ausführbare Datei erstellen zu können, muss sowohl die Bibliothek FLTK als auch der Simulator kompiliert werden. Hier wird dieser Prozess für Windows und Unix basierte Betriebssysteme beschrieben. Zu Beginn muss das Quelltextverzeichnis aus <https://github.com/janwolzenburg/ct-simulator><sup>1</sup> heruntergeladen werden. Das Verzeichnis, in dem der Quelltext liegt, wird als ct-simulator bezeichnet.

### A.1.1. Simulator erstellen

#### 1. Vorbereitung

**Windows** Microsoft Visual Studio herunterladen und installieren

Dabei die Komponenten für die Desktopentwicklung in C++ installieren

**Unix** Notwendige Pakete installieren

cmake, g++, gdb, git, autoconf, libx11-dev, libglu1-mesa-dev, libasound2-dev und libxft-dev jeweils mit dem Befehl sudo apt-get install PAKETNAME installieren

#### 2. Für FLTK<sup>2</sup> den Quelltext herunterladen

#### 3. Heruntergeladenes Verzeichnis entpacken

Dazu kann zum Beispiel 7-zip für Windows oder das Kommando

tar -xf fltk-x.y.z-source.tar.gz für Unix verwendet werden

#### 4. In den Ordner fltk-x.y.z/ navigieren

In dem Ordner befinden sich Readme-Dateien, die alternative Wege für die Kompilierung beschreiben

#### 5. FLTK erstellen

**Windows** Datei ide/VisualC2010/fltk.sln öffnen und als Release erstellen

**Unix** Konfigurieren mit ./configure Mit dem Kommando make FLTK erstellen

und mit sudo make install installieren. In den Ordner png wechseln und sudo make install ausführen. In den Ordner zlib wechseln und sudo make install ausführen.

#### 6. **Windows** FLTK Bibliotheksdateien kopieren

Die Dateien im Ordner lib fltk.lib, fltk\_images.lib, fltk\_png.lib und fltk\_z.lib in den Order ct-simulator/lib kopieren

Den Ordner FL in den Ordner ct-simulator/lib/include kopieren

#### 7. gnuplot installieren

**Windows** gnuplot<sup>3</sup> installieren

**Unix** gnuplot durch sudo apt-get install gnuplot installieren

---

<sup>1</sup>Wenn git installiert ist: git clone <https://github.com/janwolzenburg/ct-simulator>

<sup>2</sup><https://www.fltk.org/software.php> Datei:fltk-x.y.z-source.tar.gz

<sup>3</sup><https://sourceforge.net/projects/gnuplot/files/gnuplot/>

8. sciplot<sup>4</sup> herunterladen
9. Ordner sciplot nach ct-simulator/lib/include kopieren
10. Simulator erstellen

**Windows** ct-simulator.sln öffnen und Simulator erstellen

Für die Entwicklung als *Debug* und für die Benutzung als *Release*

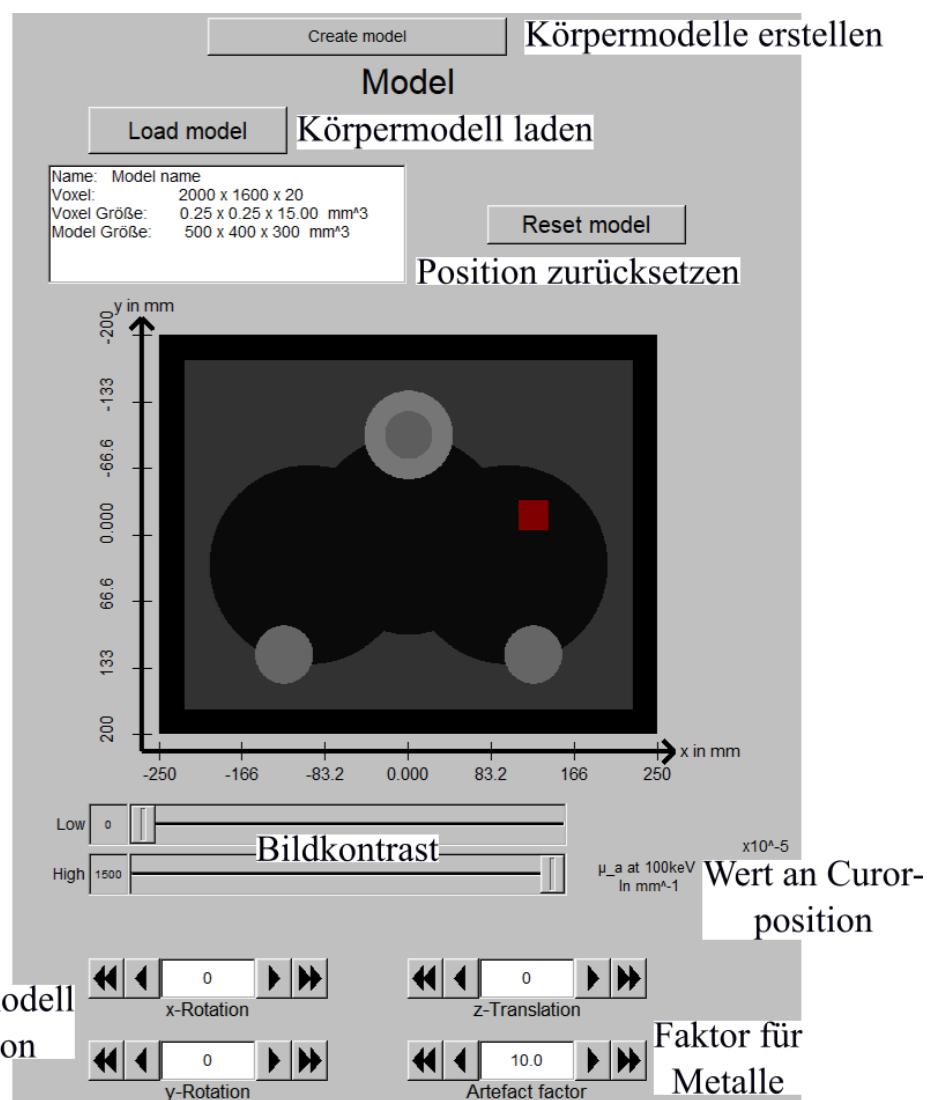
**Unix** Im Terminal in Ordner ct-simulator navigieren. Ordner erstellen mit mkdir build und betreten mit cd build. Mit cmake .. das Erstellen vorbereiten. Mit make kompilieren und linken

11. Ausführbare Datei finden

**Windows** Ausführbare Datei befindet sich in ct-simulator/x64/Release oder Debug

**Unix** Ausführbare Datei befindet sich in ct-simulator/build

### A.1.2. Simulator benutzen



**Abbildung A.1.:** Erstellen, laden und bewegen von Körpermodellen

---

<sup>4</sup><https://sciplot.github.io/installation/>

Bei Start des Simulators kann zunächst ein Körpermodell erstellt oder geladen werden. In der grafischen Benutzeroberfläche wird immer ein Schnittbild durch das Körpermodell angezeigt. Dieses stellt exakt die in den Voxeln gespeicherte Schwächungseigenschaft dar. Der Kontrast beziehungsweise Wertebereich, der auf den Farbverlauf von Schwarz nach Weiß abgebildet wird, ist über die Schieberegler einstellbar. Rechts neben den Reglern wird der Koeffizient an der Cursorposition angezeigt. Das Körpermodell kann um die  $x$ - und  $y$ -Achse rotiert und entlang der  $z$ -Achse verschoben werden. Über einen Faktor ist die Auswirkung von Voxeln mit der Eigenschaft „Metall“ einstellbar. Ist der Faktor gleich Vier, entspricht der Absorptionskoeffizient der Voxel jenem von Titan. Ist der Faktor gleich Null, jenem von Wasser. Der Faktor kann zwischen Null und Zehn eingestellt werden.

Es ist möglich in vorherigen Messungen aufgenommene Projektionen zu laden und den Programmstatus zurückzusetzen. Für die Röntgenröhre lassen sich Röhrenspannung, Röhrenstrom und das Material der Anode einstellen. Außerdem lässt sich ein optionaler Vorfilter, der das Spektrum aufhärtet, aktivieren. Dieser Vorfilter kann durch eine Energie, unterhalb derer der Photonenfluss sehr schnell gegen Null geht, und eine Steigung, die die positive Flanke ab dieser Energie beschreibt, verändert werden. Die spektrale Leistungsdichte, die aus dem Produkt aus Photonenfluss und Photonenenergie berechnet wird. Der Detektor wird auf Basis der gewünschten Abtastung der Projektionen konstruiert. Die Anzahl der Projektionswinkel und der Projektionen je Winkel sowie die Größe des Messfeldes sind dafür maßgeblich. Zusätzlich kann die Detektor-Fokus Distanz eingestellt werden. Ebenfalls ist die Anzahl der Strahlen je Pixel veränderbar und ein Streustrahlungsraster zuschaltbar. Letzteres kann durch Angabe des Akzeptanzwinkels angepasst werden. Bezuglich der Simulation lassen sich bezogen auf die Streuung verschiedene Parameter verändern. Dazu zählen die Anzahl der Iterationen für Streuung, welche Beschreibt, welche Ordnung<sup>5</sup> Strahlen haben können. Zusätzlich kann die Streuwahrscheinlichkeit und ein Faktor, der linear den gestreuten Photonenfluss beeinflusst, anpassen. Über zwei Schaltflächen können die vereinfachte<sup>6</sup> Absorption und Streuung aktiviert oder deaktiviert werden. Streuung kann nur aktiviert werden, wenn die vereinfachte Absorption deaktiviert ist. Es ist möglich und empfohlen einer Messung einen Namen zu geben. Nachdem eine Messung durchgeführt wurde, kann diese gespeichert werden.

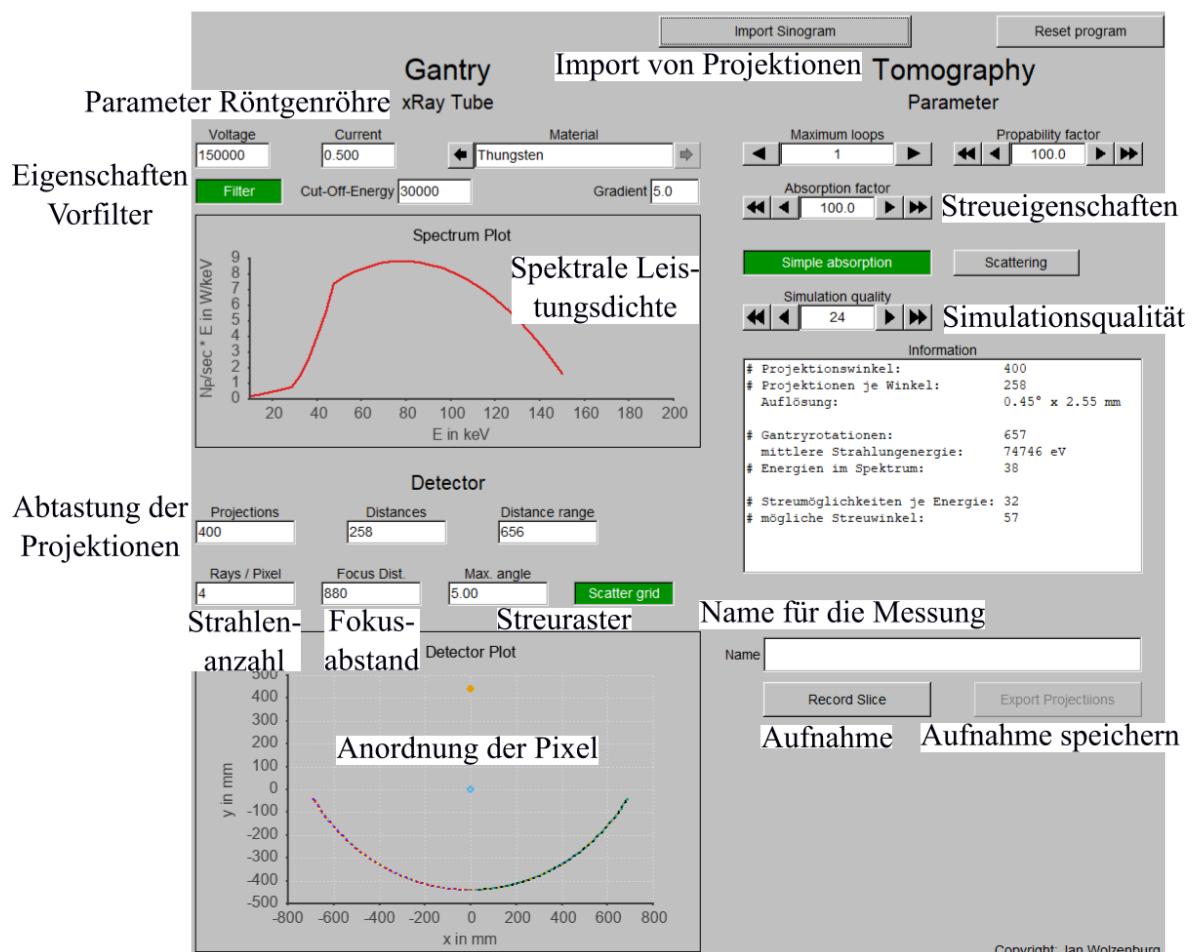
Sobald die Projektionen für ein Schnittbild aufgenommen wurden, öffnet sich ein weiteres Fenster. In diesem werden die Projektionsdaten als Sinogramm angezeigt. Dieses lässt sich auch in dem dargestellten Wertebereich anpassen. Über einen Knopf ist es möglich, den eingesetzten Wertebereich bei der Filterung zu berücksichtigen. Eine obere Grenze entspricht einer endlichen Detektorsensitivität und eine obere Grenze einer Sättigung. Bei jeder Änderung muss das rekonstruierte Bild neu berechnet werden. Der Filterkern, der für die Filterung verwendet werden soll kann ausgewählt werden. Er wird jeweils dargestellt.

---

<sup>5</sup>Ein Strahl erster Ordnung ist jener aus Röhre. Strahlen, die durch Streuung dieses Strahls entstehen, haben eine um eins höhere Ordnung usw.

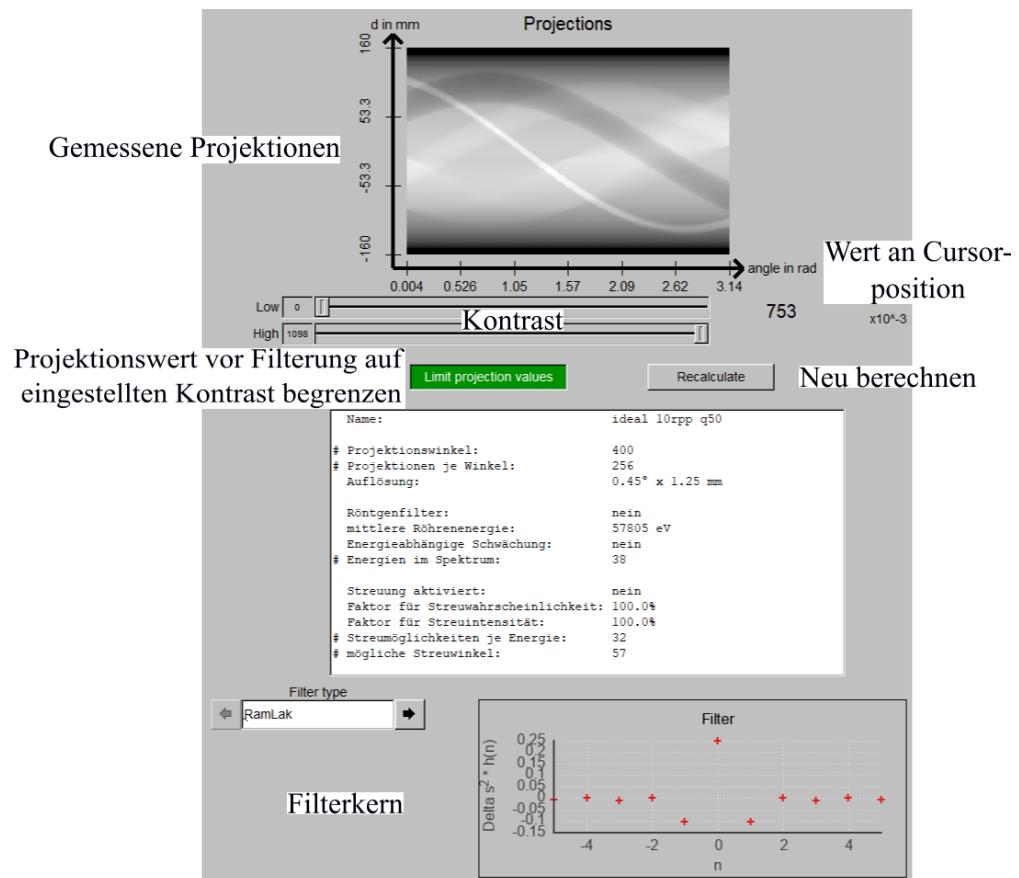
<sup>6</sup>energieunabhängige

Zurücksetzen des  
Programmstatus

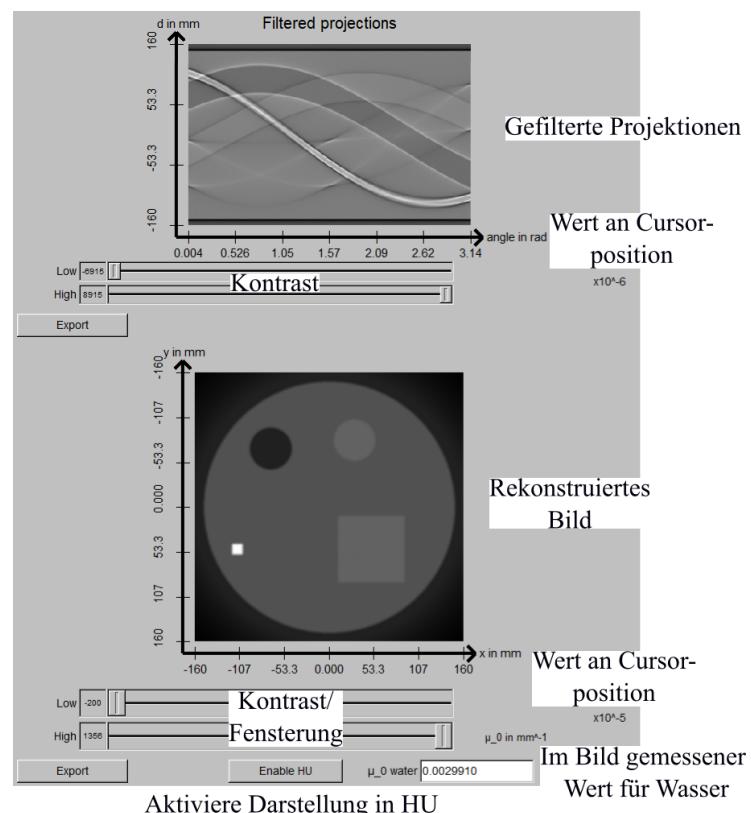


**Abbildung A.2.:** Einstellen der Geräte- und Projektionseigenschaften

Identisch zu den Projektionsdaten werden die gefilterten Projektionsdaten angezeigt. Das aus diesen Daten rekonstruierte Bild wird ebenfalls angezeigt. Auch zu diesem lässt sich der angezeigte Wertebereich ändern. Durch Messen des Wertes an einer Stelle im rekonstruierten Bild, welche Wasser entspricht, und Eintragen des Wertes in das Eingabefeld unten, können statt Schwächungskoeffizienten die Hounsfield-Einheiten angezeigt werden. Dazu muss die entsprechende Schaltfläche aktiviert werden.



**Abbildung A.3.:** Ansicht der Projektionsdaten und einstellen des Filterkerns



**Abbildung A.4.:** Ansicht der gefilterten Projektionsdaten und des rekonstruierten Bildes

## A.2. Ausgewählter Quelltext

In der ausgedruckten Variante der Arbeit ist der Quelltext nicht aufgeführt. Er kann anhand der Dateinamen und Zeilennummern im Repository oder Quelltextverzeichnis eingesehen werden. Unter <https://github.com/janwolzenburg/ct-simulator/tree/v1.0.0> ist der Quelltext zum Zeitpunkt der Abgabe gespeichert.

**Listing A.1:** Berechnung relativer Abweichung für beliebige arithmetische Datentypen.

generelMath.hpp Zeile 76 bis 79

```

1 template <typename T>
2 double RelativeDeviation( const T a, const T b ){
3     return static_cast<double>( std::abs( a - b ) ) / static_cast<double>( std::abs( b ) );
4 }
```

**Listing A.2:** Berechnung relativer Abweichung für beliebige arithmetische Datentypen.

generelMath.cpp Zeile 25 bis 37

```

1 bool IsNearlyEqual( const double a, const double b,
2                     const double tolerance, const ComparisonMode mode ){
3     // fast comparison
4     if( a == b ) return true;
5
6     // absolute comparison
7     if( mode == Absolute ){
8         return std::abs( a - b ) < tolerance;
9     }
10
11    // relative comparison
12    return RelativeDeviation( a, b ) < tolerance;
13 }
```

**Listing A.3:** Rotationsalgorithmus für einen Punkt um einen beliebigen Vektor nach [36,

S. 89]. primitiveVector3.cpp Zeile 109 bis 175

```

1 MathematicalObject::MathError PrimitiveVector3::Rotate(
2                                         const PrimitiveVector3 rotation_vector,
3                                         const double rotation_angle ){
4     // steps for rotation:
5     // 1: rotate around z axis to tilt rot. axis into x-z plane
6     // Could also rotate into y-z plane and rotate around x-axis in next step
7     // 2: rotate around y-axis to align rot. axis with z-axis
8     // 3. rotate this primitive vector by phi around z-axis
9     // 4. Undo previous rotation steps 1 and 2 in reverse order
10
11    // n must have direction
12    if( IsNearlyEqualDistance( rotation_vector.GetLength(), 0 ) )
13        return CheckForAndOutputError( MathError::Input, "rotation_axis_must_have_length!" );
14
15    // create copy and Normalise
16    PrimitiveVector3 rotation_vector_copy( rotation_vector );
17    rotation_vector_copy.Normalise();
18
19    // length of the axis projection on x-y plane
20    const double projection_length = sqrt( pow( rotation_vector_copy.x, 2 )
21                                         + pow( rotation_vector_copy.y, 2 ) );
22
23    // error doing square root?
24    if( errno != 0 )
25        return CheckForAndOutputError( MathError::General, "error_calculation_square_root!" );
26
27    // sine and cosine of angle theta:
28    // angle between rotation axis projection onto x-y plane and x-axis
29    double sine_theta = 0, cosine_theta = 1;
30
31    // avoid division by zero. d = 0 means rotation axis is parallel to z-axis
32    if( projection_length > 0 ){
33        sine_theta = rotation_vector_copy.y / projection_length; // sine of the angle Theta
34        cosine_theta = rotation_vector_copy.x / projection_length; // cosine of the angle Theta
35
36        // clockwise rotation of rotation axis and this vector around z-axis
37        // to align rotation axis to x-z plane
38        rotation_vector_copy.RotateAroundZAxis( -sine_theta, cosine_theta );
39        this->RotateAroundZAxis( -sine_theta, cosine_theta );
40    }
41
42    // gamma is the angle between the rotation axis (aligned to x-z plane) and the z-axis
43    // rotation vector has been normalised - sine of gamma is length / 1
44    // cosine is just the z-component of rotation vector
45    const double sine_gamma = projection_length;
```

```

45     const double cosine_gamma = rotation_vector_copy.z;
46     // clockwise rotation of this vector around y-axis
47     this->RotateAroundYAxis( -sine_gamma, cosine_gamma );
48
49     // the axis rotation vector is now aligned with the z-axis
50
51     // sine and cosine of angle to rotate around
52     const double sine_angle = sin( rotation_angle );
53     const double cosine_angle = cos( rotation_angle );
54
55     // counter-clockwise z-axis rotation of this vector by rotation angle
56     this->RotateAroundZAxis( sine_angle, cosine_angle );
57     // counter-clockwise y-axis rotation of this vector by gamma to reverse step 2
58     this->RotateAroundYAxis( sine_gamma, cosine_gamma );
59
60     if( projection_length > 0 ){
61         // counter-clockwise z-axis rotation of this vector by theta to reverse step 1
62         this->RotateAroundZAxis( sine_theta, cosine_theta );
63     }
64
65     return MathError::Ok;
66 }
```

#### **Listing A.4:** Koordinatentransformation in ein anderes Koordinatensystem.

coordinates.cpp Zeile 85 bis 115

```

1 Coordinates Coordinates::ConvertTo( const CoordinateSystem*
2                                     const target_coordinate_system ) const{
3
4     Coordinates temporary_coordinates{ Tuple3D{ x, y, z }, this->coordinate_system_ };
5
6     if( this->HasSameSystem( target_coordinate_system ) ) return temporary_coordinates;
7
8     // loop until coordinates are in context of global system
9     while( !temporary_coordinates.coordinate_system_->IsGlobal() ){
10        temporary_coordinates = temporary_coordinates.ConvertToParentSystem();
11    }
12
13     // target system is not global system
14     if( !target_coordinate_system->IsGlobal() ){
15        // find path from global system to target system
16        vector<const CoordinateSystem*> path_from_global = target_coordinate_system->
17        GetPathFromGlobal();
18
19        // target system is not the global system
20        if( path_from_global.size() > 0 ){
21            // iterate each coordinate system in path
22            for( auto current_system = path_from_global.cbegin();
23                 current_system < path_from_global.cend(); current_system++ ){
24                temporary_coordinates =
25                    temporary_coordinates.ConvertToChildSystem( *current_system );
26            }
27
28            temporary_coordinates =
29                temporary_coordinates.ConvertToChildSystem( target_coordinate_system );
30        }
31    }
32 }
```

#### **Listing A.5:** Lösung eines eindeutig bestimmten linearem Gleichungssystems.

systemOfEquations.cpp Zeile 204 bis 295

```

1 SystemOfEquationsSolution SystemOfEquations::Solve( void ){
2
3     SystemOfEquationsSolution solution( number_of_variables_ ); // create solution instance
4
5     // system filled with coefficients?
6     if( !IsPopulated() ) return solution;
7
8     // get size of coefficient matrix
9     const size_t rows = number_of_rows(); const size_t columns = number_of_columns();
10
11     GridIndex top_corner{}; // top-left corner of submatrix
12     GridIndex bottom_corner( rows - 1, rows - 1 ); // bottom-right corner of submatrix
13     GridIndex index_of_maximum; // indices of maximum in submatrix
14     double maximum_value; // value of maximum in submatrix
15
16     // matrix with variable indices
17     vector<unsigned int> variable_indices( number_of_variables_ );
18     for( unsigned int i = 0; i < number_of_variables_; i++ )
19         variable_indices.at( i ) = i;
20
21     // iterate all rows to create echelon form of matrix
22     for( size_t k = 0; k < rows; k++ ){
23
24         // top-left corner changes each iteration
25         top_corner.r = k; top_corner.c = k;
```

```

27 // search maximum absolutionate value in quadratic submatrix
28 // from (k, k) to (rows - 1, rows - 1)
29 index_of_maximum = FindMaximum( top_corner, bottom_corner );
30
31 // get value
32 maximum_value = data_.at( number_of_columns_ * index_of_maximum.r + index_of_maximum.c );
33
34 // return if maximum is zero -> no solution
35 if( IsNearlyEqualDistance( maximum_value, 0. ) ) return solution;
36
37 // swap rows and columns to bring the cell with maximum value to (k,k)
38 if( index_of_maximum.r != k ) SwapRows( index_of_maximum.r, k );
39 if( index_of_maximum.c != k ){
40     SwapColumns( index_of_maximum.c, k );
41
42     // variable in columns swapped
43     unsigned int temporary = variable_indices.at( k );
44     variable_indices.at( k ) = variable_indices.at( index_of_maximum.c );
45     variable_indices.at( index_of_maximum.c ) = temporary;
46 }
47
48 ScaleRow( k, 1 / maximum_value );      // scale k-row by reciprocal of (k, k) value
49
50 // subtract k-row from k+1 to n-row to eliminate cells
51 for( size_t row = k + 1; row < rows; row++ ){
52     const double cell_value = data_.at( number_of_columns_ * row + k );
53     // target cell not zero?
54     if( !IsNearlyEqualDistance( cell_value, 0. ) ){
55         // make (row, k) to one by scaling row with reciprocal
56         ScaleRow( row, 1 / cell_value );
57         SubtractRows( k, row );           // subtract k-row from row
58     }
59 }
60
61 // eliminate remaining cells 'above' the diagonal
62 // iterate coefficient columns
63 for( size_t c = columns - 2; c > 0; c-- ){
64
65     // iterate rows above diagonal
66     for( size_t r = 0; r < c; r++ ){
67         const double cell_value = data_.at( number_of_columns_ * r + c );
68         // current cell not already zero?
69         if( !IsNearlyEqualDistance( cell_value, 0. ) ){
70             // calculate result column cell as if current cell is being eliminated
71             double& result_column_cell = data_.at( number_of_columns_ * r + ( columns - 1 ) );
72             const double current_cell = data_.at( number_of_columns_ * c + ( columns - 1 ) );
73             result_column_cell -= ( current_cell * cell_value );
74         }
75     }
76 }
77
78 size_t variable_index;
79 // copy result column entries to var-array
80 for( size_t i = 0; i < rows; i++ ){
81     variable_index = variable_indices.at( i );      // index for swapped variable
82     // check if index is in allowed range
83     if( variable_index < solution.number_of_variables() )
84         solution.SetVariableValue( variable_index,
85                                     data_.at( number_of_columns_ * i + columns - 1 ) );
86     else
87         return solution;
88 }
89 solution.solution_found( true );
90 return solution;
91
92 }

```

**Listing A.6:** Schnittpunktberechnung einer Geraden und Ebene. intersections.hpp

Zeile bis

```

1 template<class L, class S>
2 LineSurfaceIntersection<L, S>::LineSurfaceIntersection( const L& line, const S& surface ) :
3     LineSurfaceIntersection<L, S>{}
4 {
5     // primitive vectors without coordinate system context
6     PrimitiveVector3 surface_origin, surface_direction_1, surface_direction_2;
7     PrimitiveVector3 line_direction = line.direction().GetComponents();
8     PrimitiveVector3 line_origin = line.origin().GetComponents();
9
10    // same system?
11    if( line.direction().GetCoordinateSystem() ==
12        surface.direction_1().GetCoordinateSystem() ){
13        // simply get the components
14        surface_origin = surface.origin().GetComponents();
15        surface_direction_1 = surface.direction_1().GetComponents();
16        surface_direction_2 = surface.direction_2().GetComponents();
17    }
18    else{
19        // convert surface vectors to line's coordinate system

```

```

20     surface_origin = surface.origin().GetComponents( line.direction() );
21     surface_direction_1 = surface.direction_1().GetComponents( line.direction() );
22     surface_direction_2 = surface.direction_2().GetComponents( line.direction() );
23 }
24
25 // create system of equations with three variables and populate
26 // a*v_1 + b*v_2 - c*v = u_line - u_surface
27 SystemOfEquations system_of_equations( 3 );
28 system_of_equations.PopulateColumn( surface_direction_1 );
29 system_of_equations.PopulateColumn( surface_direction_2 );
30 system_of_equations.PopulateColumn( -line_direction );
31 system_of_equations.PopulateColumn( line_origin - surface_origin );
32
33 // solve system
34 SystemOfEquationsSolution solution = system_of_equations.Solve();
35
36 // no solution found
37 if( !solution.solution_found() ) return;
38
39 // copy result
40 intersection_exists_ = true;                                // solution found
41 line_parameter_ = solution.GetVariableValue( 2 );           // line parameter
42 surface_parameter_1_ = solution.GetVariableValue( 0 );       // surface parameter A
43 surface_parameter_2_ = solution.GetVariableValue( 1 );       // surface parameter B
44 intersection_point_ = line.GetPoint( line_parameter_ );      // point of intersection
45
46 // surface parameters outside allowed bounds of surface or line parameter invalid
47 // uses late binding with virtual methods
48 if( !surface.AreParametersInBounds( surface_parameter_1_, surface_parameter_2_ ) ||
49     !line.IsParameterInBounds( line_parameter_ ) )
50     intersection_exists_ = false;
51
52 };

```

**Listing A.7:** Berechnung des Röntgenspektrums aus Röhreneigenschaften.

XRayTube.cpp Zeile 80 bis 167

```

1 XRayTube::XRayTube( CoordinateSystem* const coordinate_system,
2                     const XRayTubeProperties tube_properties ) :
3     coordinate_system_( coordinate_system ),
4     properties_( tube_properties ),
5     anode_material_atomic_number_( ForcePositive(
6         XRayTubeProperties::materials.at( properties_.anode_material ).second ) ),
7     radiation_power_W_(  

8         efficiency_constant_PerV * static_cast<double>( anode_material_atomic_number_ ) *  

9         properties_.anode_current_A * pow( properties_.anode_voltage_V, 2 ) ),
10    max_photon_energy_eV_( ForceToMax(
11        properties_.anode_voltage_V, maximum_energy_in_tube_spectrum ) )
12 {
13
14     // prepare empty spectrum. first vector holds the energies,  

15     // second holds the photonflow in an arbitrary unit  

16     VectorPair energy_spectrum{ CreateLinearSpace( minimum_energy_in_tube_spectrum,
17             max_photon_energy_eV_,  

18             simulation_properties.number_of_points_in_spectrum ),
19             vector<double>( simulation_properties.number_of_points_in_spectrum, 0. ) };
20
21     const double energy_resolution = energy_spectrum.first.at( 1 ) -  

22             energy_spectrum.first.at( 0 );
23
24     // fill value vector with values
25     for( auto current_energy = energy_spectrum.first.begin();
26          current_energy < energy_spectrum.first.end(); current_energy++ ) {
27
28         // current index
29         const size_t energy_index = current_energy - energy_spectrum.first.begin();
30         energy_spectrum.second.at(energy_index) =
31             (energy_spectrum.first.back() - *current_energy + 2 * energy_resolution);
32     }
33
34     double complete_power = 0.;
35     for( size_t energy_index = 0; energy_index < energy_spectrum.first.size();
36          energy_index++ ) {
37
38         // cumulate power
39         complete_power += energy_spectrum.first.at( energy_index ) *
40             energy_spectrum.second.at( energy_index );
41     }
42
43     // only the shape of the spectrum is correct
44     // this is the factor to scale the spectrum by
45     const double correction_factor = radiation_power_W_ / complete_power / J_Per_eV;
46     Scale( energy_spectrum.second, correction_factor ); // scale by factor
47
48     // apply hardening filter
49     if( properties_.has_filter_ ){
50
51         // energy to which the filter dominates spectral behaviors
52         const double change_energy =
53             properties_.filter_cut_of_energy +
54             ( energy_spectrum.first.back() - properties_.filter_cut_of_energy ) /
55             ( 1. + properties_.filter_gradient );
56
57         // gradient of filter
58         const double filter_gradient = properties_.filter_gradient * correction_factor;
59
60         // iterate energies and attenuate according to filter properties
61         for( auto current_energy = energy_spectrum.first.begin();
62              current_energy < energy_spectrum.first.end(); current_energy++ ) {
63
64             // current energy index
65             const size_t energy_index = current_energy - energy_spectrum.first.begin();
66
67             // if filter dominates weaken the photonflow
68             if( *current_energy < change_energy ) {
69                 if( *current_energy < properties_.filter_cut_of_energy ) {
70                     energy_spectrum.second.at( energy_index ) =
71                         0.5 * energy_resolution *
72                         ( *current_energy + properties_.filter_cut_of_energy ) *
73                         ( 1 / properties_.filter_cut_of_energy ) * filter_gradient;
74                 } else{
75                     energy_spectrum.second.at( energy_index ) =
76                         ( *current_energy - properties_.filter_cut_of_energy + energy_resolution ) *
77                         filter_gradient;
78                 }
79             }
80         }
81
82         // write energy and power values to spectrum
83         emitted_spectrum_ = EnergySpectrum{ energy_spectrum };
84         radiation_power_W_ = emitted_spectrum_.GetTotalPower();
85     }

```

**Listing A.8:** Erzeugung der Nadelstrahlen. XRayTube.cpp Zeile 170 bis 229

```

1 vector<Ray> XRayTube::GetEmittedBeam( const vector<DetectorPixel> detector_pixel,
2                                         const double detector_focus_distance ) const {
3
4     const size_t number_of_rays = properties_.number_of_rays_per_pixel_ *
5                               detector_pixel.size();
6
7     // split spectrum into the ray spectra
8     const EnergySpectrum single_ray_spectrum =
9         emitted_spectrum_.GetEvenlyScaled( 1. / static_cast<double>( number_of_rays ) );
10
11    // vector with rays
12    vector<Ray> rays;
13
14    size_t pixel_index = 0;
15
16    // iterate all pixel
17    for( const DetectorPixel& current_pixel : detector_pixel ){
18
19        // properties of created rays for this pixel
20        const RayProperties ray_properties{ single_ray_spectrum, pixel_index++, true };
21
22        // get points on the edge of pixel, the line between them and their distance
23        const Point3D edge_point_1 =
24            current_pixel.GetPoint( current_pixel.parameter_1_min(), 0 );
25        const Point3D edge_point_2 =
26            current_pixel.GetPoint( current_pixel.parameter_1_max(), 0 );
27        const Line connection_line{ edge_point_2 - edge_point_1, edge_point_1 };
28        const double edge_distance = ( edge_point_2 - edge_point_1 ).length();
29
30        // distance on the pixel between rays which hit the pixel
31        const double distance_delta =
32            edge_distance / static_cast<double>( properties_.number_of_rays_per_pixel_ + 1 );
33
34        // iterate all rays hitting current pixel
35        for( size_t ray_index = 0; ray_index < properties_.number_of_rays_per_pixel_;
36             ray_index++ ){
37
38            // offset of current ray origin
39            const double offset = static_cast<double>( ray_index + 1 ) * distance_delta;
40
41            // current ray origin
42            const Point3D ray_origin_on_pixel = connection_line.GetPoint( offset );
43
44            // temporary Line pointing from pixel to tube
45            const Line line_to_tube{
46                current_pixel.GetNormal().ConvertTo( coordinate_system_ ),
47                ray_origin_on_pixel.ConvertTo( coordinate_system_ ) };
48
49            // origin of ray with specific distance to pixel
50            const Point3D ray_origin = line_to_tube.GetPoint( detector_focus_distance );
51
52            // add ray in tube's coordinate system to vector
53            rays.emplace_back( -line_to_tube.direction(), ray_origin, ray_properties );
54
55        }
56    }
57
58    return rays;
59 }

```

**Listing A.9:** Konstruktion des Detektors. xRayDetector.cpp Zeile 25 bis 155

```

1 XRayDetector::XRayDetector( CoordinateSystem* const coordinate_system,
2                             const ProjectionsProperties projection_properties,
3                             const PhysicalDetectorProperties physical_properties ) :
4     coordinate_system_( coordinate_system ),
5     properties_{ projection_properties, physical_properties }
6 {
7
8     // amount of distances or pixel
9     const size_t number_of_distances = properties_.number_of_pixel.c;
10
11    // measure field
12    const double distance_range = static_cast<double>( number_of_distances - 1 ) *
13                                projection_properties.distances_resolution();
14
15    // angle resolution
16    const double delta_theta = projection_properties.angles_resolution();
17
18    // distance resolution
19    const double delta_distance = projection_properties.distances_resolution();
20
21    // distance from middle pixel to origin
22    const double detector_focus_distance = properties_.detector_focus_distance / 2.;
23
24    // y-axis of coordinate system is the middle normal vector
25    const Unitvector3D middle_normal_vector = coordinate_system_->GetEy();
26
27    // pixel normals should lie in xy-plane
28    // the middle normal vector will be rotated around this vector
29    const Unitvector3D rotation_vector = coordinate_system_->GetEz();
30
31
32    // persistent variables
33    Line previous_pixel_normal; // get normal of previous pixel
34    double previous_pixel_size; // size of previous pixel

```

```

30
31
32 // iterate through half of pixel normals. second half is created by symmetry
33 // normals will be created inside to outside
34 for( size_t pixel_index = 0; pixel_index < number_of_distances / 2; pixel_index++ ){
35
36 // angle to rotate the middle normal vector by
37 const double rotation_angle = ( static_cast<double>( pixel_index ) + 0.5 )
38 * delta_theta;
39
40 // middle normal vector rotation by rotation angle around rotation vector
41 const Unitvector3D normal_vector =
42     middle_normal_vector.RotateConstant( rotation_vector, rotation_angle );
43
44
45 // find a point with the distance corresponding to distance in sinogram
46 // the point's origin vector must be perpendicular to the current normal vector
47
48 // the lot is perpendicular to the current normal vector and it lies in xy-plane
49 const Unitvector3D normal_lot = rotation_vector ^ normal_vector;
50
51 // distance from origin to normal. Is the distance in the sinogram
52 const double current_distance = distance_range / 2. -
53 ( static_cast<double>( number_of_distances ) / 2.
54 - static_cast<double>( pixel_index ) - 1 ) * delta_distance;
55
56 // point which lies on the current normal and has the correct distance from the origin
57 const Point3D lot_end_point = Vector3D{ normal_lot } * current_distance;
58
59 // the current normal
60 const Line normal{ normal_vector, lot_end_point };
61
62 Point3D pixel_origin; // origin of current pixel
63 double pixel_size; // size of current pixel
64
65 // "middle" normal
66 if( pixel_index == 0 ){
67     //this is the starting point
68     pixel_origin = normal.GetPoint( detector_focus_distance );
69
70     // first pixel size so that the neighbouring pixel intersects at half angle
71     pixel_size = 2 * tan( delta_theta / 2. ) *
72         ( detector_focus_distance + current_distance / tan( delta_theta / 2. ) );
73 }
74 else{
75     // intersection point of pixel
76     const Point3D intersection = previous_pixel_normal.origin() +
77         ( previous_pixel_normal.direction() ^ rotation_vector ) * previous_pixel_size / 2.;
78
79     // lot vector from current normal to intersection point
80     // vector is pointing to the normal
81     const Vector3D intersection_lot = normal.GetLot( intersection );
82
83     // get the pixel normal's origin which lies on the shortest Line connection the
84     // intersection with current normal
85     pixel_origin = intersection + intersection_lot;
86
87     // pixel size is double the lot length
88     pixel_size = 2 * intersection_lot.length();
89 }
90
91
92 // create current pixel normal pointing to center
93 const Line pixel_normal{ -normal_vector, pixel_origin };
94
95 // store for next pixel
96 previous_pixel_normal = pixel_normal;
97 previous_pixel_size = pixel_size;
98
99 // vector perpendicular to the normal pointing to the next pixel
100 const Unitvector3D surface_vector = -pixel_normal.direction() ^ rotation_vector;
101
102 // add pixel
103 pixel_array_.emplace_back( BoundedSurface{
104     surface_vector, rotation_vector,
105     pixel_normal.origin(),
106     -pixel_size / 2, pixel_size / 2,
107     -properties_.row_width / 2.,
108     properties_.row_width / 2. } );
109
110 // mirror current normal around y-axis
111 const Line mirroredPixelNormal{
112     pixel_normal.direction().NegateXComponent(),
113     pixel_normal.origin().NegateXComponent()
114 };
115
116 // add mirrored pixel
117 const Unitvector3D mirrored_surface_vector = - mirroredPixelNormal.direction()
118     ^ rotation_vector;
119 pixel_array_.emplace_back( BoundedSurface{
120     mirrored_surface_vector, rotation_vector,
121     mirroredPixelNormal.origin(),
122     -pixel_size / 2, pixel_size / 2,
123     -properties_.row_width / 2.,

```

```

124         properties_.row_width / 2. } );
125     }
126
127     // after constructing converted poixel are identical
128     converted_pixel_array_ = pixel_array_;
129 }
130 }
```

**Listing A.10:** Bestimmung des Pixels, der von einem Strahl getroffen wird.

xRayDetector.cpp Zeile 167 bis 243

```

1 optional<size_t> XRayDetector::GetHitPixelIndex( Ray& ray ) const{
2     const size_t expected_pixel_index = ray.properties().expected_detector_pixel_index();
3
4     if( ray.properties().definitely_hits_expected_pixel() ){
5         return expected_pixel_index;
6     }
7
8     size_t pixel_index = expected_pixel_index;
9
10    // iterate all pixel indices. but start with the most likely pixel
11    for( size_t counter = 0; counter < pixel_array_.size() + 1; counter++ ){
12
13        // not first iteration: index is counter - 1 because first iteration was the
14        // most likely pixel
15        if( counter != 0 ){
16            pixel_index = counter - 1;
17
18            // skip detection if expected index would be tested again
19            if( pixel_index == expected_pixel_index )
20                continue;
21        }
22
23        // converted pixel
24        const DetectorPixel& current_pixel = converted_pixel_array_.at( pixel_index );
25
26        // check if ray hit anti scattering structure hit is possible
27        // if detector has anti scattering structure and angle not allowed by structure
28        // goto next pixel
29        if( properties_.has_anti_scattering_structure &&
30            (PI / 2. - ray.GetAngle(current_pixel)) >
31            properties_.max_angle_allowed_by_structure) {
32            continue;
33        }
34
35        // check for intersection of ray with current pixel
36        const RayPixelIntersection pixel_hit{ ray, current_pixel };
37
38        // do they intersect?
39        if( pixel_hit.intersection_exists_ ){
40            ray.SetExpectedPixelIndex( pixel_index, true );
41            return pixel_index;
42        }
43    }
44
45    return {};
46
47
48 }
```

**Listing A.11:** Detektion eines Strahls. xRayDetector.cpp Zeile 216 bis 242

```

1 #ifdef TRANSMISSION_TRACKING // only enabled for verification
2 bool XRayDetector::DetectRay( Ray& ray, mutex& pixel_array_mutex ){
3 #else
4 bool XRayDetector::DetectRay( Ray& ray, mutex& pixel_array_mutex ){
5 #endif
6
7     optional<size_t> pixel_index = GetHitPixelIndex( ray );
8
9     if( pixel_index.has_value() ){
10         pixel_array_mutex.lock();
11         pixel_array_.at( pixel_index.value() ).AddDetectedRayProperties( ray.properties() );
12         pixel_array_mutex.unlock();
13
14 #ifdef TRANSMISSION_TRACKING // only enabled for verification
15     Point3D intersection_point =
16         RayPixelIntersection{ ray,
17             pixel_array_.at( pixel_index.value() ) }.intersection_point_;
18     if( !ray.ray_tracing().tracing_steps.empty() ){
19         ray.ray_tracing().tracing_steps.back().exit = intersection_point;
20         ray.ray_tracing().tracing_steps.back().distance =
21             (intersection_point - ray.ray_tracing().tracing_steps.back().entrance).length();
22     }
23 #endif
24 }
25
26     return pixel_index.has_value();
27 }
```

**Listing A.12:** Bestimmung der Voxelindizes zu einem Punkt. model.cpp Zeile 124 bis 154

```

1 Index3D Model::GetVoxelIndices( const Tuple3D local_coordinates ) const{
2
3     if( local_coordinates.x < 0 || local_coordinates.y < 0 ||
4         local_coordinates.z < 0 ){
5         CheckForAndOutputError( MathError::Input,
6             "only_positive_coordinates_allowed_in_model!" );
7         return Index3D{ 0, 0, 0 };
8     }
9
10    Index3D indices{
11        static_cast<size_t>( local_coordinates.x / voxel_size_.x ),
12        static_cast<size_t>( local_coordinates.y / voxel_size_.y ),
13        static_cast<size_t>( local_coordinates.z / voxel_size_.z )
14    };
15
16
17 // suppress error when index is exactly on the edge
18 if( indices.x == number_of_voxel_3D_.x ) indices.x = number_of_voxel_3D_.x - 1;
19 if( indices.y == number_of_voxel_3D_.y ) indices.y = number_of_voxel_3D_.y - 1;
20 if( indices.z == number_of_voxel_3D_.z ) indices.z = number_of_voxel_3D_.z - 1;
21
22 // are indices too big?
23 if( !AreIndicesValid( indices ) )
24     CheckForAndOutputError( MathError::Input, "coordinates_exceed_model_size!" );
25
26    indices.x = ForceToMax( indices.x, number_of_voxel_3D_.x - 1 );
27    indices.y = ForceToMax( indices.y, number_of_voxel_3D_.y - 1 );
28    indices.z = ForceToMax( indices.z, number_of_voxel_3D_.z - 1 );
29
30    return indices;
31 }
```

**Listing A.13:** Strahlverfolgung durch das Modell. model.cpp Zeile 197 bis 389

```

1 pair<Ray, vector<Ray>> Model::TransmitRay(
2     const Ray& ray,
3     const TomographyProperties& tomography_properties,
4     const RayScattering& scattering_properties,
5     RandomNumberGenerator & dedicated_rng,
6     const bool disable_scattering ) const{
7
8 // current ray in model's coordinate system
9 Ray local_ray = ray.ConvertTo( this->coordinate_system_ );
10
11 // find entrance inside model
12 const RayVoxelIntersection model_intersection{ GetModelVoxel(), local_ray };
13
14 // return if ray does not intersect model
15 if( !model_intersection.entrance_.intersection_exists_ ) return { local_ray, {} };
16
17 // iteration through model
18 /* -----
19 ----- */
20
21 // ray parameter at model entrance plus a tiny bit
22 double current_ray_step = model_intersection.entrance_.line_parameter_ +
23         simulation_properties.ray_step_size_mm;
24
25 // return when the point on the ray is not inside the model meaning that the ray
26 // just barely hit the model
27 if( !IsPointInside( local_ray.GetPoint( current_ray_step ) ) )
28     return { local_ray, {} };
29
30 // current point on the ray is model entrance
31 Point3D current_point_on_ray = local_ray.GetPoint( current_ray_step );
32
33 #ifdef TRANSMISSION_TRACKING
34 if( !IsPointInside( local_ray.origin() ) ){
35     local_ray.ray_tracing().tracing_steps.emplace_back( false,
36         model_intersection.entrance_.line_parameter_, GetModelVoxel().data(),
37         local_ray.origin(), model_intersection.entrance_.intersection_point_,
38         local_ray.properties().simple_intensity(),
39         local_ray.properties().energy_spectrum().GetTotalPower(),
40         local_ray.properties().energy_spectrum() );
41 }
42 #endif
43
44 vector<Ray> all_scattered_rays; // vector for all scattered rays
45
46 // the possible exit faces of voxel
47 const array<bool, ConvertToUnderlying( Voxel::Face::End )> possible_exit_faces =
48     local_ray.GetPossibleVoxelExits();
49
50 // iterate through model while current point is inside model
51 while( IsPointInside( current_point_on_ray ) ){
52
53     // indices of current voxel
54     const Index3D current_voxel_indices = GetVoxelIndices( current_point_on_ray );
55     double distance_to_exit = INFINITY; // the smallest ray parameter
56
57     // iterate all possible faces and get the ray parameter at intersection
58     for( Voxel::Face current_face = Voxel::Face::Begin;
59          current_face < Voxel::Face::End; ++current_face ){
60
61         // distance to the current face. Will become finite if exit found
62         double distance_to_current_face = INFINITY;
63         double exit_face_position; // face position along one axis
64
65         // switch faces. check only the possible faces
66         switch( current_face ){
67
68             case Voxel::Face::YZ_Xp:
69                 if( possible_exit_faces.at( ConvertToUnderlying( current_face ) ) == false )
70                     break;
71                 exit_face_position = ( static_cast<double>( current_voxel_indices.x ) + 1. ) *
72                     this->voxel_size_.x; // position of positive yz-plane
73                 distance_to_current_face = ( exit_face_position - current_point_on_ray.X() ) /
74                     local_ray.direction().X();
75                 break;
76
77             case Voxel::Face::YZ_Xm:
78                 if( possible_exit_faces.at( ConvertToUnderlying( current_face ) ) == false )
79                     break;
80                 exit_face_position = ( static_cast<double>( current_voxel_indices.x ) * -
81                     this->voxel_size_.x; // position of negative yz-plane
82                 distance_to_current_face = ( exit_face_position - current_point_on_ray.X() ) /
83                     local_ray.direction().X();
84                 break;
85
86             case Voxel::Face::XZ_Yp:
87                 if( possible_exit_faces.at( ConvertToUnderlying( current_face ) ) == false )
88                     break;
89                 exit_face_position = ( static_cast<double>( current_voxel_indices.y ) + 1. ) *
90                     this->voxel_size_.y; // position of positive xz-plane
91                 distance_to_current_face = ( exit_face_position - current_point_on_ray.Y() ) /
92                     local_ray.direction().Y();
93                 break;
94
95             case Voxel::Face::XZ_Ym:
96                 if( possible_exit_faces.at( ConvertToUnderlying( current_face ) ) == false )
97                     break;
98                 exit_face_position = ( static_cast<double>( current_voxel_indices.y ) * -
99                     this->voxel_size_.y; // position of negative xz-plane
100                distance_to_current_face = ( exit_face_position - current_point_on_ray.Y() ) /
101                    local_ray.direction().Y();
102                break;
103
104         }
105     }
106 }

```

```

90         break;
91
92     case Voxel::Face::XZ_Ym:
93         if( possible_exit_faces.at( ConvertToUnderlying( current_face ) ) == false )
94             break;
95         exit_face_position = ( static_cast<double>( current_voxel_indices.y ) ) *
96             this->voxel_size_.y; // position of negative xz-plane
97         distance_to_current_face = ( exit_face_position - current_point_on_ray.Y() ) /
98             local_ray.direction().Y();
99         break;
100
101    case Voxel::Face::XY_Zp:
102        if( possible_exit_faces.at( ConvertToUnderlying( current_face ) ) == false )
103            break;
104        exit_face_position = ( static_cast<double>( current_voxel_indices.z ) + 1. ) *
105             this->voxel_size_.z; // position of positive xy-plane
106        distance_to_current_face = ( exit_face_position - current_point_on_ray.Z() ) /
107             local_ray.direction().Z();
108        break;
109
110    case Voxel::Face::XY_Zm:
111        if( possible_exit_faces.at( ConvertToUnderlying( current_face ) ) == false )
112            break;
113        exit_face_position = ( static_cast<double>( current_voxel_indices.z ) ) *
114             this->voxel_size_.z; // position of negative xy-plane
115        distance_to_current_face = ( exit_face_position - current_point_on_ray.Z() ) /
116             local_ray.direction().Z();
117        break;
118
119    default: break;
120 }
121
122 // there may be more than one intersectiones with the unbounded faces
123 // the intersected face with the smallest distance from current point along the ray
124 // is the exit face
125 if( distance_to_current_face < distance_to_exit )
126     distance_to_exit = distance_to_current_face;
127
128 // if exit found
129 if( distance_to_exit < INFINITY ){
130
131     // the distance traveled in this voxel. add step size, because ray origin is
132     // this amount inside the voxel
133     const double distance_in_voxel = distance_to_exit +
134         simulation_properties.ray_step_size_mm;
135
136     // the current voxel's properties
137     const VoxelData current_voxel_data = this->GetVoxelData( current_voxel_indices );
138
139     // update ray's properties with distance traveled in current voxel
140     local_ray.UpdateProperties( current_voxel_data, distance_in_voxel );
141     local_ray.IncrementHitCounter();
142
143 #ifdef TRANSMISSION_TRACKING
144     local_ray.ray_tracing().tracing_steps.emplace_back( true,
145             distance_in_voxel, current_voxel_data, current_point_on_ray,
146             local_ray.GetPointFast( current_ray_step ),
147             local_ray.properties().simple_intensity(),
148             local_ray.properties().energy_spectrum().GetTotalPower(),
149             local_ray.properties().energy_spectrum() );
150 #endif
151
152     // new ray parameter the distance travelled in this voxel
153     current_ray_step += distance_in_voxel;
154
155     // the new point on the ray
156     current_point_on_ray = std::move( local_ray.GetPointFast( current_ray_step ) );
157
158     // scattering. only when enabled, not overriden and current point is inside model
159     if( tomography_properties.scattering_enabled &&
160         !disable_scattering &&
161         IsPointInside( current_point_on_ray ) ){
162
163         // scatter the ray
164         const vector<Ray> scattered_rays =
165             local_ray.Scatter(
166                 scattering_properties,
167                 current_voxel_data, distance_in_voxel,
168                 tomography_properties, current_point_on_ray,
169                 dedicated_rng );
170
171         // append scattered rays
172         all_scattered_rays.insert( all_scattered_rays.end(),
173             make_move_iterator( scattered_rays.begin() ),
174             make_move_iterator( scattered_rays.end() ) );
175     }
176 }
177
178 // ray is now outside the model
179 /* ----- */

```

```

180 // new origin of ray "outside" the model
181 local_ray.origin( current_point_on_ray );
182
183 #ifdef TRANSMISSION_TRACKING
184     local_ray.ray_tracing().tracing_steps.emplace_back( false,
185             -1, GetModelVoxel().data(), current_point_on_ray, current_point_on_ray,
186             local_ray.properties().simple_intensity(),
187             local_ray.properties().energy_spectrum().GetTotalPower(),
188             local_ray.properties().energy_spectrum() );
189 #endif
190
191     return { local_ray, all_scattered_rays };
192 }
193 }
```

**Listing A.14:** Bestimmung der möglichen Randflächen, durch die der Strahl aus ein Voxel austreten kann. ray.cpp Zeile 60 bis 97

```

1 array<bool, ConvertToUnderlying( Voxel::Face::End )> Ray::GetPossibleVoxelExits( void )
2     const{
3
4     array<bool, ConvertToUnderlying( Voxel::Face::End )> face_possibilities{ false };
5
6     // iterate all faces of voxel
7     for( Voxel::Face current_face = Voxel::Face::Begin;
8          current_face < Voxel::Face::End; ++current_face ) {
9
10    // check if face can be an exit face of the ray
11    switch( current_face ) {
12        case Voxel::Face::YZ_Xp:
13            if( direction_.X() > 0 )
14                face_possibilities.at( ConvertToUnderlying( current_face ) ) = true; break;
15        case Voxel::Face::YZ_Xm:
16            if( direction_.X() < 0 )
17                face_possibilities.at( ConvertToUnderlying( current_face ) ) = true; break;
18        case Voxel::Face::XZ_Yp:
19            if( direction_.Y() > 0 )
20                face_possibilities.at( ConvertToUnderlying( current_face ) ) = true; break;
21        case Voxel::Face::XZ_Ym:
22            if( direction_.Y() < 0 )
23                face_possibilities.at( ConvertToUnderlying( current_face ) ) = true; break;
24        case Voxel::Face::XY_Zp:
25            if( direction_.Z() > 0 )
26                face_possibilities.at( ConvertToUnderlying( current_face ) ) = true; break;
27        case Voxel::Face::XY_Zm:
28            if( direction_.Z() < 0 )
29                face_possibilities.at( ConvertToUnderlying( current_face ) ) = true; break;
30        default: break;
31    }
32
33    return face_possibilities;
34 }
35 }
36
37 }
```

**Listing A.15:** Energieabhängige Absorption von Photonen. `energySpectrum.cpp` Zeile 115 bis 125

```

1 void EnergySpectrum::GetAbsorbed( const VoxelData& voxel_data, const double distance ) {
2     double absorption_coefficient;
3
4     for( auto& photonflow: photonflow_per_energy_ ){
5         absorption_coefficient = voxel_data.GetAbsorptionAtEnergy( photonflow.x );
6         photonflow.y *= exp( -absorption_coefficient * distance );
7     }
8
9     mean_energy_valid_ = false;
10 }
11 }
```

**Listing A.16:** Compton Streuung. `ray.cpp` Zeile 120 bis 324

```

1 vector<Ray> Ray::Scatter( const RayScattering& scattering_information,
2                             const VoxelData& voxel_data,
3                             const double distance_traveled_mm,
4                             const TomographyProperties& tomography_properties,
5                             const Point3D& new_origin,
6                             RandomNumberGenerator & dedicated_rng ) {
7
8     // voxel's absorption with respect to water's absorption
9     const double coefficient_factor = voxel_data.GetAbsorptionAtReferenceEnergy() /
10                                absorption_water_Per_mm;
11
12    // attenuation coefficient at reference energy. for simple intensity
13    const double cross_section_at_reference_energy_mm =
14        ScatteringCrossSection ::GetInstance().GetCrossSection( reference_energy_for_mu_eV );
15
16    const double coefficient_at_reference_energy_1Permm =
17        cross_section_at_reference_energy_mm * electron_density_water_1Permm3 *
18        coefficient_factor;
19
20    const double simple_fraction = exp( -distance_traveled_mm *
21                                         coefficient_at_reference_energy_1Permm );
22
23    properties_.simple_intensity_ *= simple_fraction;
24
25    // ----- Scattering -----
26
27    vector<Ray> scattered_rays;
28    // skip if scattering is extremely inprobable
29    if( IsNearlyEqual( coefficient_factor, 0., 1e-6, Relative ) ) return scattered_rays;
30
31    vector<pair<double, pair<double, double>>> scattered_angles;
32
33    // number of scatteres bins over all energies
34    size_t scattered_bins_sum = 0;
35
36    // iterate energies in spectrum
37    size_t energy_index = 0;
38    for( const auto& [ energy, photons ] : properties_.energy_spectrum_.data() ) {
39
40        // no photons at current energy
41        if( IsNearlyEqual( photons, 0., 1e-6, Relative ) ) continue;
42
43        // calculate scattering propability from compton scattering cross section
44        const double cross_section_mm =
45            ScatteringCrossSection ::GetInstance().GetCrossSection( energy );
46
47        const double coefficient_1Permm = cross_section_mm *
48            electron_density_water_1Permm3 *
49            coefficient_factor;
50
51        const double scatter_propability = 1. - exp( -coefficient_1Permm *
52                                         distance_traveled_mm );
53
54        // iterate through energy bins
55        for( size_t bin = 0; bin < simulation_properties.bins_per_energy; bin++ ) {
56
57            // does scattering happen?
58            if( !dedicated_rng.DidARandomEventHappen(
59                scatter_propability * tomography_properties.scatter_propability_correction ) )
60                continue;
61
62            // check if angle is scattered inside scattering plane
63            if( std::abs( scattering_information.GetRandomAngle(
64                energy, dedicated_rng ) ) <=
65                scattering_information.max_angle_to_lie_in_plane() ) {
66
67                // random angle inside scattering plane
68                const double angle = ForceRange( scattering_information.GetRandomAngle(
69                    energy, dedicated_rng ), -PI, PI );
70
71                // if angle is almost zero -> treat as if no scattering happened
72                if( IsNearlyEqual( angle, 0., 1e-3, Relative ) ) continue;
73
74            }
```

```

76     // calculate scattered photons energy via compton-wavelength
77     const double new_energy =
78         1. / ( 1. / ( me_c2_eV ) * ( 1. - cos( angle ) ) + 1. / energy );
79
80     // new photonflow
81     const double new_photonflow =
82         tomography_properties.scattered_ray_absorption_factor * photons /
83         static_cast<double>( simulation_properties.bins_per_energy );
84
85     scattered_angles.emplace_back( angle,
86         pair<double, double>{ new_energy, new_photonflow } );
87 }
88
89 // scalar for energy in incoming ray. only accounts for energy lost to new rays
90 // without considering der angle dependent energy loss (Compton-Aporption).
91 // this is because the compton-absorption is already accounted for in the absorption
92 // routine
93 const double energy_scalar =
94     1. - tomography_properties.scattered_ray_absorption_factor /
95     static_cast<double>( simulation_properties.bins_per_energy );
96
97 properties_.energy_spectrum_.ScaleEnergy( energy_index, energy_scalar );
98 #ifdef TRANSMISSION_TRACKING
99 properties_.only_scattering_spectrum.ScaleEnergy( energy_index, energy_scalar );
100#endif
101
102     scattered_bins_sum++;
103 }
104
105     energy_index++;
106 }
107
108 // no ray scattered in scattering plane -> return empty
109 if( scattered_angles.size() == 0 )
110     return {};
111
112 // sort scattered angles
113 std::sort( scattered_angles.begin(), scattered_angles.end(),
114             [] ( const auto& a, const auto& b ) { return a.first > b.first; } );
115
116 double previous_angle = -1.; // angle currently handled
117 size_t scattered_bins_for_current_angle = 0;
118
119 // photonflows for scattered ray
120 vector<Tuple2D> spectral_photonflows;
121
122 // iterate through sorted angles
123 for( size_t angle_index = 0; angle_index < scattered_angles.size(); angle_index++ ){
124
125     const double angle = scattered_angles.at( angle_index ).first;
126     const double energy = scattered_angles.at( angle_index ).second.first;
127     const double photonflow = scattered_angles.at( angle_index ).second.second;
128     scattered_bins_for_current_angle++;
129
130     // angle already in spectrum or first angle
131     if( previous_angle == angle || angle_index == 0 ){
132
133         // iterate through current spectrum
134         // energy index tracks where to put add photonflow to
135         energy_index = 0;
136         for(; energy_index < spectral_photonflows.size();
137             energy_index++ ){
138
139             // energy is in current spectrum
140             if( energy == spectral_photonflows.at( energy_index ).x )
141                 break; // current index is index of energy
142
143             // energy is not in spectrum -> add
144             if( energy_index == spectral_photonflows.size() ){
145                 spectral_photonflows.emplace_back( energy, photonflow );
146             }
147             else{
148                 spectral_photonflows.at( energy_index ).y += photonflow;
149             }
150         }
151     else{
152         spectral_photonflows.emplace_back( energy, photonflow );
153     }
154
155     // flag to know whether the current angle has power and needs to be created
156     bool build_ray = false;
157
158     // at least one angle left
159     if( angle_index < scattered_angles.size() - 1 ){
160         // next ray has different angle
161         if( scattered_angles.at( angle_index + 1 ).first != angle )
162             build_ray = true;
163     }
164     // last angle
165     else if( angle_index == scattered_angles.size() - 1 )
166         build_ray = true;
167
168     // build scattered ray. When angle is finished or last element is reached

```

```

169     if( build_ray ){
170         // use complete amount of scattered bins to take scattering into account of
171         // simple intensity
172         const double scattered_bins_fraction =
173             static_cast<double>( scattered_bins_for_current_angle ) /
174             static_cast<double>( scattered_bins_sum );
175
176         const EnergySpectrum new_spectrum{ spectral_photonflows };
177
178         // create ray properties
179         RayProperties new_properties{ new_spectrum };
180         new_properties.voxel_hits_ = properties_.voxel_hits_;
181         new_properties.simple_intensity_ = properties_.simple_intensity_ *
182             scattered_bins_fraction *
183             simple_fraction;
184         new_properties.initial_power_ = new_spectrum.GetTotalPower();
185
186         const Unitvector3D new_direction =
187             direction_.RotateConstant( scattering_information.scattering_plane_normal(),
188                                         angle );
189
190         // save scattered ray
191         scattered_rays.emplace_back( new_direction, new_origin, new_properties );
192
193         // reset for next angle
194         scattered_bins_for_current_angle = 0;
195         spectral_photonflows.clear();
196     }
197
198     // remember previous angle
199     previous_angle = angle;
200
201 }
202
203 return scattered_rays;
204
205 }
```

**Listing A.17:** Generierung pseudozufälliger Zahlen nach [32].

propabilityDistribution.cpp Zeile 43 bis 58

```

1 unsigned int RandomNumberGenerator ::GetRandomNumber( void ){
2
3     // algorithm from https://prng.di.unimi.it/xoroshiro128starstar.c
4     const unsigned long long int s0 = generator_state_[0];
5     unsigned long long int s1 = generator_state_[1];
6     const unsigned long long int result = std::rotl(s0 * 5, 7) * 9;
7     s1 ^= s0;
8
9     const unsigned long long int new_state_0 = std::rotl(s0, 24) ^ s1 ^ (s1 << 16);
10    const unsigned long long int new_state_1 = std::rotl(s1, 37);
11
12    generator_state_[0] = new_state_0;
13    generator_state_[1] = new_state_1;
14
15    return static_cast<unsigned int>( result & 0x00000000FFFFFFFF );
16 }
```

**Listing A.18:** Generierung zufälliger Ereignisse mit gegebener Wahrscheinlichkeit.

propabilityDistribution.cpp Zeile 64 bis 77

```

1 bool RandomNumberGenerator ::DidARandomEventHappen( const double event_propability ){
2
3     const unsigned int interval = UINT32_MAX;
4     const double single_value_propability = 1. / static_cast<double>( interval );
5     const unsigned int event_interval =
6         static_cast<unsigned int>( event_propability / single_value_propability );
7
8     const unsigned int random_integer = GetRandomNumber();
9
10    if( random_integer < event_interval )
11        return true;
12
13    return false;
14 }
```

**Listing A.19:** Berechnung der Streuwinkelwahrscheinlichkeiten. rayScattering.cpp

Zeile 23 bis 70

```

1 RayScattering::RayScattering( const size_t number_of_angles,
2                               const NumberRange energy_range,
3                               const size_t number_of_energies,
4                               const Unitvector3D scattering_normal,
5                               const double maximum_angle_to_lie_in_scatter_plane ) :
6   number_of_energies_( ForcePositive( number_of_energies ) ),
7   angle_resolution_( ( 2. * PI ) / static_cast<double>( number_of_angles - 1 ) ),
8   energy_range_( energy_range ),
9   energy_resolution_( ( energy_range_.end() - energy_range_.start() ) /
10                      static_cast<double>( number_of_energies_ - 1 ) ),
11   scattering_plane_normal_( scattering_normal ),
12   max_angle_to_lie_in_scatter_plane_( maximum_angle_to_lie_in_scatter_plane )
13 {
14   // iterate all energies
15   for( size_t energy_index = 0; energy_index < number_of_energies_; energy_index++ ){
16
17     const double energy = energy_range_.start() +
18                           static_cast<double>( energy_index ) * energy_resolution_;
19
20     // calculate pseudo probability distribution
21     vector<Tuple2D> pseudo_distribution;
22
23     // initial photon energy
24     const double a = energy / me_c2_eV;
25
26     // iterate all angles
27     for( size_t angle_index = 0; angle_index < number_of_angles; angle_index++ ){
28
29       const double angle = -PI + static_cast<double>( angle_index ) * angle_resolution_;
30       const double t = angle;
31       const double pseudo_probability =
32         ( 1. + pow( cos( t ), 2 ) ) /
33         ( 2 * pow( 1. + a * ( 1. - cos( t ) ), 2 ) ) *
34         ( 1. +
35           ( pow( a, 2 ) * pow( 1. - cos( t ), 2 ) ) /
36           ( ( 1. + pow( cos( t ), 2 ) ) * ( 1. + a * ( 1. - cos( t ) ) ) ) )
37     );
38
39     pseudo_distribution.emplace_back( t, pseudo_probability );
40
41     if( angle_index == ( number_of_angles - 1 ) / 2 )
42       pseudo_distribution.emplace_back( t, pseudo_probability );
43
44   }
45   scattering_angle_distributions_.emplace_back( energy,
46                                               PropabilityDistribution{ pseudo_distribution } );
46
47 }
48 }
```

**Listing A.20:** Aufnahme der Projektionsdaten für ein Schnittbild. `tomography.cpp`  
Zeile 91 bis 179

```

1 optional<Projections> Tomography::RecordSlice(
2     const ProjectionsProperties projection_properties,
3     Gantry gantry, const Model& model,
4     const double z_position,
5     Fl_Progress_Window* progress_window ) {
6
7     // update simulation properties
8     simulation_properties = SimulationProperties{ properties_.simulation_quality };
9
10    // reset gantry to its initial position
11    gantry.ResetGantry();
12
13    // translate gantry
14    if( z_position != 0. )
15        gantry.TranslateInZDirection( z_position );
16
17    // assign gantry coordinate-system's unit-vectors to radon coordinate system
18    this->radon_coordinate_system_->CopyPrimitiveFrom( gantry.coordinate_system() );
19
20    // create projections
21    Projections projections{ projection_properties, properties_ };
22
23
24    RayScattering scattering_information{
25        simulation_properties.number_of_scatter_angles,
26        gantry.tube().GetEmittedEnergyRange(),
27        simulation_properties.number_of_energies_for_scattering,
28        gantry.coordinate_system()->GetEz(),
29        atan(gantry.detector().properties().row_width /
30              gantry.detector().properties().detector_focus_distance / 2) };
31
32    // radiate the model for each frame
33    for( size_t frame_index = 0;
34         frame_index < projection_properties.number_of_frames_to_fill();
35         frame_index++ ){
36
37        if( progress_window != nullptr )
38            progress_window->ChangeLineText( 0, "Radiating_frame_" +
39                ConvertToString( frame_index ) + "_of_" +
40                ConvertToString( projection_properties.number_of_frames_to_fill() ) );
41
42
43        // radiate
44        gantry.RadiateModel( model, properties_, scattering_information );
45
46        // get the detection result
47        const vector<DetectorPixel> pixel_array = gantry.pixel_array();
48
49        // iterate all pixel
50        for( const DetectorPixel& pixel : pixel_array ){
51
52            // get coordinates for pixel
53            const RadonCoordinates radon_coordinates{ this->radon_coordinate_system_,
54                pixel.NormalLine() };
55
56            optional<double> line_integral =
57                pixel.GetProjectionValue( properties_.use_simple_absorption,
58                    gantry.tube().number_of_rays_per_pixel(), gantry.tube().GetEmittedBeamPower() /
59                    ( static_cast<double>( pixel_array.size() ) *
60                      static_cast<double>( gantry.tube().number_of_rays_per_pixel() ) )
61                );
62
63
64            // if no value no ray was detected by pixel, the line integral would be infinite.
65            // set current_byte to a high value instead
66            if( !line_integral.has_value() ){
67                line_integral = 25.; // is like ray's energy is 1 / 10^11 of its start energy
68            }
69
70            // get the radon point
71            const RadonPoint radon_point{ radon_coordinates, line_integral.value() };
72
73            // assign the data to sinogram
74            projections.AssignData( radon_point );
75        }
76
77        // rotate gantry
78        gantry.RotateCounterClockwise( projection_properties.angles_resolution() );
79
80        Fl::check();
81
82        if( progress_window != nullptr ){
83            if( !progress_window->visible() )
84                return {};
85        }
86
87
88        return projections;
89    }

```

**Listing A.21:** Durchstrahle das Körpermodell mit der aktuellen Gantryposition.

gantry.cpp Zeile 136 bis 218

```

1 void Gantry::RadiateModel( const Model& model,
2                             TomographyProperties tomography_properties,
3                             const RayScattering& scattering_information ) {
4
5     // current rays. start with rays from source
6     vector<Ray> rays =
7         tube_.GetEmittedBeam(
8             detector_.pixel_array(),
9             detector_.properties().detector_focus_distance ) ;
10
11    // convert rays to model's coordinate system
12    for( Ray& current_ray : rays ){
13        current_ray = std::move( current_ray.ConvertTo( model.coordinate_system() ) );
14    }
15
16    // convert pixel
17    detector_.ConvertPixelArray( model.coordinate_system() );
18
19    // scattered rays should lie in the same plane as the detector
20    // const Unitvector3D scattering_rotation_axis =
21    // this->coordinate_system_->GetEz().ConvertTo( model.coordinate_system() );
22
23
24    detector_.ResetDetectedRayPorperties(); // reset all pixel
25
26    size_t shared_current_ray_index = 0; // index of next ray to iterate
27    mutex current_ray_index_mutex; // mutual exclusion for ray index
28    mutex rays_for_next_iteration_mutex; // mutual exclusion for ray storage
29    mutex detector_mutex; // mutual exclusion for detector
30
31    // loop until maximum loop depth is reached or no more rays are left to transmit
32    for( size_t current_iteration = 0;
33         current_iteration <= tomography_properties.max_scattering_occurrences &&
34         rays.size() > 0;
35         current_iteration++ ){
36
37        // no scattering in last iteration
38        tomography_properties.scattering_enabled =
39            current_iteration < tomography_properties.max_scattering_occurrences &&
40            tomography_properties.scattering_enabled;
41
42        const bool second_to_last_iteration =
43            ( current_iteration == tomography_properties.max_scattering_occurrences - 1 );
44
45        // store for information output
46        tomography_properties.mean_energy_of_tube = this->tube_.GetMeanEnergy();
47
48        vector<Ray> rays_for_next_iteration; // rays to process in the next iteration
49        shared_current_ray_index = 0; // reset current ray index
50
51        const size_t number_of_threads = std::thread::hardware_concurrency();
52
53        vector<RandomNumberGenerator> dedicated_rngs( number_of_threads );
54
55        // start threads
56        vector<std::thread> threads;
57        for( size_t thread_index = 0;
58             thread_index < number_of_threads; thread_index++ ){
59
60            // transmit rays
61            threads.emplace_back( TransmitRaysThreaded,
62                                  cref( model ), cref( tomography_properties ),
63                                  cref( scattering_information ),
64                                  cref( rays ), second_to_last_iteration,
65                                  ref( shared_current_ray_index ),
66                                  ref( current_ray_index_mutex ), ref( rays_for_next_iteration ),
67                                  ref( rays_for_next_iteration_mutex ),
68                                  ref( detector_ ), ref( detector_mutex ),
69                                  ref( dedicated_rngs.at( thread_index ) ) );
70
71            // for debugging
72            if( thread_index == 0 ) first_thread_id = threads.back().get_id();
73        }
74
75        // wait for threads to finish
76        for( std::thread& currentThread : threads ) currentThread.join();
77
78        rays = std::move( rays_for_next_iteration );
79
80    }
81
82 }
```

**Listing A.22:** Funktion für die parallele Verfolgung einzelner Strahlen durch das Körpermodell. `gantry.cpp` Zeile 67 bis 134

```

1 void Gantry::TransmitRaysThreaded(
2     const Model& model, const TomographyProperties& tomography_properties,
3     const RayScattering& ray_scattering,
4     const vector<Ray>& rays, const bool second_to_last_iteration,
5     size_t& shared_current_ray_index, mutex& current_ray_index_mutex,
6     vector<Ray>& rays_for_next_iteration, mutex& rays_for_next_iteration_mutex,
7     XRayDetector& detector, mutex& detector_mutex,
8     RandomNumberGenerator & dedicated_rng ) {
9
10    size_t local_ray_index;
11    Ray current_ray;
12    pair<Ray, vector<Ray>> rays_to_return;
13
14    // loop while rays are left
15    while( shared_current_ray_index < rays.size() ){
16
17        // get the ray which should be transmitted next and increment index
18        current_ray_index_mutex.lock();
19        local_ray_index = shared_current_ray_index++;
20        current_ray_index_mutex.unlock();
21
22        // no more rays left
23        if( local_ray_index >= rays.size() ) break;
24
25        // get current ray
26        current_ray = rays.at( local_ray_index );
27
28        // transmit ray through model
29        rays_to_return = std::move(
30            model.TransmitRay( cref( current_ray ), cref( tomography_properties ),
31                               cref( ray_scattering ),
32                               ref( dedicated_rng ) ) );
33
34        // detect the ray
35        detector.DetectRay( ref( rays_to_return.first ), ref( detector_mutex ) );
36
37        if( rays_to_return.second.empty() ){
38            continue;
39        }
40
41        // check if this is last iteration and delete rays which can not be detected
42        if( second_to_last_iteration ){
43
44            // check each ray
45            for( auto& ray : rays_to_return.second ){
46                // detection possible. ray's expected pixel index is updated in TryDetection()
47                if( detector.TryDetection( ray ) ){
48                    // only rays which can be detected are in the next iteration
49                    rays_for_next_iteration_mutex.lock();
50                    rays_for_next_iteration.emplace_back( std::move( ray ) );
51                    rays_for_next_iteration_mutex.unlock();
52                }
53            }
54        }
55        else{
56            // add all rays for next iteration
57            rays_for_next_iteration_mutex.lock();
58            rays_for_next_iteration.insert(
59                rays_for_next_iteration.end(),
60                make_move_iterator( rays_to_return.second.begin() ),
61                make_move_iterator( rays_to_return.second.end() ) );
62            rays_for_next_iteration_mutex.unlock();
63        }
64    }
65
66    return;
67 }
68 }
```

**Listing A.23:** Bestimme die Radon-Koordinaten einer Geraden nach dem Ordnungsschema. `radonCoordinates.cpp` Zeile 26 bis 66

```

1 RadonCoordinates::RadonCoordinates( const CoordinateSystem* const radon_coordinate_system,
2                                     const Line line )
3 {
4     // project ray on xy-plane
5     const Line projected_line = line.ProjectOnXYPlane( radon_coordinate_system );
6
7     // get perpendicular to projected ray through coordinate system's origin
8     Vector3D lot = projected_line.GetLot( radon_coordinate_system->GetOriginPoint() );
9
10    // ray intersects origin
11    if( IsNearlyEqualDistance( lot.length(), 0. ) ){
12        // lot vector only for angle calculation
13        lot = projected_line.direction() ^ radon_coordinate_system->GetEz();
14        distance = 0.;
```

```

15    }
16    // no intersection -> distance from perpendicular
17    else{
18        // y component is zero
19        if( IsNearlyEqualDistance( lot.Y(), 0 ) ){
20            // x component is less than zero
21            if( lot.X() < 0 ) distance = -lot.length();
22            else distance = lot.length();
23        }
24        // y component is not zero
25        else{
26            // y component is less than zero
27            if( lot.Y() < 0 ) distance = -lot.length();
28            else distance = lot.length();
29        }
30    }
31
32    // calculate radon angle
33
34    // y component is zero
35    if( IsNearlyEqualDistance( lot.Y(), 0 ) ) theta = 0;
36    // y component is greater than zero
37    else if( lot.Y() > 0. ) theta = radon_coordinate_system->GetEx().GetAngle( lot );
38    // y component is less than zero
39    else theta = PI - radon_coordinate_system->GetEx().GetAngle( lot );
40
41 }

```

**Listing A.24:** Berechnung der Projektionsdaten aus den Strahleigenschaften.

detectorPixel.cpp Zeile 26 bis 54

```

1 optional<double> DetectorPixel::GetProjectionValue(
2     const bool use_simple_absorption,
3     const size_t expected_ray_hits,
4     const double start_intensity ) const{
5
6     double sum_of_end_intensity = 0.;
7     double sum_of_simple_end_intensity = 0. ;
8
9     // iterate all detected ray properties
10    for( const RayProperties& currentRay : detected_ray_properties_ ){
11        sum_of_end_intensity += currentRay.energy_spectrum_.GetTotalPower();
12        sum_of_simple_end_intensity += currentRay.simple_intensity();
13    }
14
15    // check no rays were detected return empty
16    if( detected_ray_properties_.size() == 0 )
17        return {};
18
19    // calculate line integral
20    const double line_integral_spectrum = log( start_intensity *
21                                              static_cast<double>( expected_ray_hits ) /
22                                              sum_of_end_intensity ) ;
23
24    const double line_integral_simple = log( 1. *
25                                              static_cast<double>( expected_ray_hits ) /
26                                              sum_of_simple_end_intensity );
27
28    return !use_simple_absorption ? line_integral_spectrum : line_integral_simple;
29 }

```

**Listing A.25:** Berechnung der gefilterten Projektionen. filteredProjections.cpp

Zeile 28 bis 103

```

1  FilteredProjections :: FilteredProjections (
2      const Projections& projections,
3      const BackprojectionFilter ::TYPE filter_type,
4      Fl_Progress_Window* progress_window ) :
5
6  DataGrid{ projections.data().size(), projections.data().start(),
7      projections.data().resolution(), 0. },
8  filter_{ NaturalNumberRange{ -static_cast<signed long long>( size().r + 1 ),
9      static_cast<signed long long>( size().r - 1 ) },
10     resolution().r, filter_type } :
11 {
12     const size_t number_of_angles = size().c; // number of angles
13     const size_t number_of_distances = size().r; // number of distances
14
15     const double distance_resolution = resolution().r; // distance resolution
16
17     // local copy of projection data
18     const DataGrid<> projections_data = projections.data();
19
20     // iterate all angles
21     for( size_t angle_index = 0; angle_index < number_of_angles; angle_index++ ){
22
23         if( progress_window != nullptr )
24             progress_window->ChangeLineText( 0, "Filtering_angle_"
25                                         + ConvertToString( angle_index + 1 )
26                                         + "_of_" + ConvertToString( number_of_angles ) );
27
28         // iterate all distances
29         for( size_t distance_index = 0;
30              distance_index < number_of_distances;
31              distance_index++ ){
32
33             if( filter_.type() == BackprojectionFilter ::TYPE::constant ){
34                 this->SetData(
35                     GridIndex{ angle_index, distance_index },
36                     projections_data.GetData( GridIndex{ angle_index, distance_index } ) );
37                 continue;
38             }
39
40             // convolute filter with with projections
41             double convolution_sum = 0;
42
43             for( signed long long kernel_index = filter_.points_range().start();
44                  kernel_index <= filter_.points_range().end();
45                  kernel_index++ ){
46
47                 // index of current row in projection data
48                 signed long long projection_row_index =
49                     static_cast<signed long long>( distance_index ) - kernel_index;
50
51                 // index out of bounds of input data ->
52                 // add nothing is like padding input data with zero
53                 if( projection_row_index < 0 ||
54                     projection_row_index >= static_cast<signed long long>( number_of_distances ) ){
55                     continue;
56                 }
57
58                 // projection data
59                 const double projection_value = projections_data.GetData(
60                     GridIndex{ angle_index, static_cast<size_t>( projection_row_index ) } );
61
62                 // filter value
63                 const double kernel_value = filter_( kernel_index );
64
65                 // multiply
66                 convolution_sum += kernel_value * projection_value;
67             }
68             convolution_sum *= distance_resolution;
69             this->SetData( GridIndex{ angle_index, distance_index }, convolution_sum );
70
71         }
72     }
73     Fl::check();
74 }
75 }
```

**Listing A.26:** Parallel Rekonstruktion jeweils einer Bildspalte. backprojection.cpp

Zeile 29 bis 93

```

1 void Backprojection::ReconstructImageColumn(
2     size_t& current_angle_index, mutex& current_angle_index_mutex,
3     Backprojection& image, mutex& image_mutex,
4     Fl_Progress_Window* progress, mutex& progress_mutex,
5     const FilteredProjections& projections ) {
6
7     const size_t number_of_distances = image.size().r;           // number of distances
8
9     const size_t number_of_angles = projections.size().c;          // number of angles
10    const double angle_resolution = projections.resolution().c; // angle resolution
11
12    while( current_angle_index < number_of_angles ){
13
14        size_t angle_index;
15
16        current_angle_index_mutex.lock();
17        angle_index = current_angle_index++;
18        current_angle_index_mutex.unlock();
19
20        if( angle_index >= number_of_angles ) break;
21
22        if( progress != nullptr ){
23            progress_mutex.lock();
24            progress->ChangeLineText( 1, "Backprojecting_projection_"
25                + ConvertToString( angle_index + 1 ) + "_of_"
26                + ConvertToString( number_of_angles ) );
27            progress_mutex.unlock();
28        }
29
30        // current angle value
31        const double angle = static_cast<double>( angle_index ) * angle_resolution +
32                                         angle_resolution / 2.;
33        const double cos_angle = cos( angle );
34        const double sin_angle = sin( angle );
35
36        // iterate image columns
37        for( size_t column_index = 0; column_index < number_of_distances; column_index++ ){
38            const double column_coordinate = image.GetColumnCoordinate( column_index );
39
40            // iterate image rows
41            for( size_t row_index = 0; row_index < number_of_distances; row_index++ ){
42                const double row_coordinate = image.GetRowCoordinate( row_index );
43
44                // pixel's "distance" or magnitude in polar coordinates
45                const double s = column_coordinate * cos_angle + row_coordinate * sin_angle;
46
47                // projection value for given angle and distance
48                const double projection_value = projections.GetValue( angle_index, s );
49
50                // value to add to pixel value
51                double new_value = projection_value * PI / static_cast<double>( number_of_angles );
52
53                image_mutex.lock();
54                new_value += image.GetData( GridIndex{ column_index, row_index } );
55                image.SetData( GridIndex{ column_index, row_index }, new_value );
56                image_mutex.unlock();
57            }
58        }
59
60        progress_mutex.lock();
61        Fl::check();
62        progress_mutex.unlock();
63    }
64 }
```

**Listing A.27:** Interpolation der gefilterten Projektionen. filteredProjections.cpp

Zeile 105 bis 136

```

1 double FilteredProjections::GetValue( const size_t angle_index,
2                                     const double distance ) const{
3
4     const double distance_resolution = resolution().r; // distance resolution
5     const size_t number_of_distances = size().r;         // number of distances
6
7     // exact 'index' of distance. ideally we want to get the value at this index
8     const double exact_index = ForceRange(
9         distance / distance_resolution +
10        static_cast<double>( number_of_distances - 1 ) / 2.,
11        static_cast<double>( number_of_distances ) - 1. );
12
13     // if the exact index is a whole number skip interpolation
14     if( IsNearlyEqualDistance( round( exact_index ), exact_index ) ){
15         // return value at distance index
16         return (*this)( GridIndex{ angle_index, static_cast<size_t>( exact_index ) } );
17     }
18
19     // interpolate
```

```
20     const size_t index_floor = static_cast<size_t>( floor( exact_index ) ); // lower index
21     const size_t index_ceil   = static_cast<size_t>( ceil( exact_index ) ); // upper index
22
23     const double value_floor = this->operator()( 
24         GridIndex{ angle_index, index_floor } ); // value at floor index
25     const double value_ceil  = this->operator()( 
26         GridIndex{ angle_index, index_ceil } ); // value at ceil index
27
28 // return the interpolated value
29 return value_floor + ( value_ceil - value_floor ) *
30             ( exact_index - static_cast<double>( index_floor ) );
31
32 }
```

# **Eigenständigkeitserklärung**

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Altena, den 16. Mai 2024

Jan Wolzenburg

---