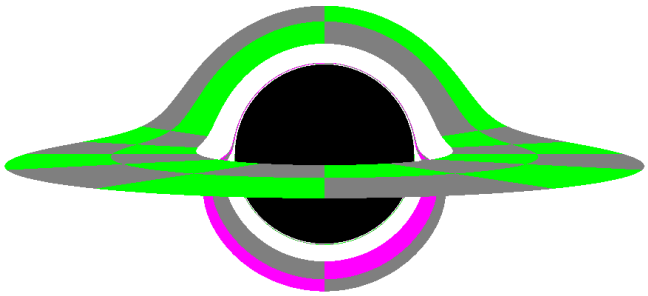
cuRRay: CUDA-Raytracer für Lichtstrahlen in relativistischer Kerr-Newman-Raumzeit

Wettbewerbsarbeit Nationaler Wettbewerb SJF 2018



Sébastien C. Garmier

23. März 2018

Experte SJF

Marcel Balsiger

Ursprünglich der Kantonsschule Wohlen AG
als Maturaarbeit eingereicht

Betreuung Kantonsschule Wohlen

Dr. Adrien Cornaz
Dr. Jan-Mark Iniotakis

Inhaltsverzeichnis

Vorwort **iv**

Abstract **v**

Einleitung **vi**

Notation **viii**

1 Allgemeine Relativitätstheorie **1**
1.1 Spezielle Relativitätstheorie 1
1.2 Äquivalenzprinzip 2
1.3 Gekrümmte Raumzeit 4
1.4 Tensoren 6
1.5 Koordinaten 7
1.6 Metrischer Tensor 8
1.7 Geodäten 9
1.8 Riemann-Tensor 12
1.9 Energie-Impuls-Tensor 13
1.10 Einsteinsche Feldgleichungen 14

2 Schwarze Löcher **15**
2.1 Schwarzschild-Metrik 15
2.2 Kerr-Newman-Metrik 17
2.3 Killing-Vektorfelder und Erhaltungsgrössen 22
2.3.1 Killing-Vektorfelder der Kerr-Newman-Metrik 22
2.3.2 Geodätengleichungen der Kerr-Newman-Metrik 23

3 Licht **24**
3.1 Elektromagnetisches Feld 24
3.2 Elektromagnetische Wellen 25
3.3 Photonen 26
3.4 Lichtablenkung 26
3.5 Gravitative Rotverschiebung 27

4 Übersicht `cuRRay` **28**
4.1 Funktionen 28
4.2 Systemvoraussetzungen 30

4.3 Verwendete Code-Bibliotheken 30
4.4 Kurzanleitung 31

5 Programmstruktur **32**
5.1 Ein- und Ausgabe 32
5.2 Parallele Ausführung von Code 33
5.2.1 CPU 33
5.2.2 GPU 34

6 Raytracing **36**
6.1 Gitter-Layout 36
6.2 Abarbeitung der Frames 38
6.3 Anfangswertproblem 38
6.4 Berechnung der Anfangsgeschwindigkeit 39
6.5 Runge-Kutta-Algorithmus vierter Ordnung 41
6.6 Schrittweitensteuerung 44
6.7 Überwachung des Geschwindigkeitsvektors 44
6.8 Abbruchbedingungen 44
6.9 Rotverschiebung 45
6.10 Sternenhimmel 46
6.11 Ausführliche Pixeldaten 47

7 Ergebnisse **49**
7.1 Schwarzschild-Löcher 49
7.2 Kerr-Löcher 57
7.3 Reissner-Nordström-Löcher 60

8 Diskussion **62**
8.1 Zuverlässigkeit von `cuRRay` 62
8.2 Effizienz von `cuRRay` 63
8.3 Ausblick 64
8.4 Bestehende Software 65

A Geodätengleichungen erster Ordnung **66**

B Christoffelsymbole in Kerr-Newman-Raumzeit **68**

C Gravitative Rotverschiebung in Kerr-Newman-Raumzeit **71**

D Herleitung der Kartesisch-BL-Koordinatentransformation **73**

E `cuRRay`: Bedienungsanleitung **78**
E.1 Kommandozeile 78
E.2 Systemkonfigurationsdatei (Sysconfig) 81
E.3 Raytracing 83
E.4 Szenendatei 84
E.5 TDR-Timer unter Windows 87

E.6 Quellcode-Dokumentation 88

F Git-Repository **89**

G Kompilieren **90**
G.1 Allgemeines 90
G.2 Kompilieren unter Windows (Visual Studio) 90
G.3 Kompilieren unter Linux (g++-5) 91

Abbildungsverzeichnis **93**

Listingverzeichnis **94**

Literaturverzeichnis **95**

Vorwort

Albert Einstein gilt als einer der grössten Physiker aller Zeiten. Alle kennen seinen Namen, doch nur die wenigsten verstehen ihn. Seine beiden Relativitätstheorien stellten innerhalb von zehn Jahren zweimal unser Weltbild auf den Kopf. Ich erhielt im Physikunterricht bereits Einblick in die spezielle Relativitätstheorie, die weitaus einfachere der beiden, doch die allgemeine Relativitätstheorie würde erst im Physikstudium besprochen. Natürlich war ich neugierig, die schwierigere der Relativitätstheorien zumindest ansatzweise im Rahmen der Maturaarbeit zu lernen. Ich musste eine Anwendung der allgemeinen Relativitätstheorie finden, welche für eine Maturaarbeit geeignet wäre.

Die Anwendung – Lichtstrahlen, welche durch ein Gravitationsfeld abgelenkt werden – fiel mir spontan ein. Ich hatte bereits Bilder von schwarzen Löchern gesehen, in welchen die Sterne im Hintergrund zu langen Kreisbögen verzogen waren, weil ihr Licht abgelenkt wurde. Solche Bilder sind, wie ich nach kurzer Recherche herausfand, durchaus physikalisch. Ich setzte mir für die Maturaarbeit das Ziel, die allgemeine Relativitätstheorie so gut zu verstehen, dass ich mit einem selbst programmierten Computerprogramm Bilder von schwarzen Löchern erzeugen kann. Ich stand vor einem Berg, der Weg zum Gipfel war mir unbekannt.

Nach rund neun Monaten erreichte ich den Gipfel, der höher war als alle Schätzungen am Fuss des Berges. Das Produkt der Arbeit, die Software `cuRRay`, übertraf alle meine Erwartungen. Ich hatte in drei Quartalen mehr gelernt als in vier Schuljahren.

Doch weitaus höhere Berge bilden das Gebirge der Physik und insbesondere der allgemeinen Relativitätstheorie. Die spezifische Anwendung der Theorie für diese Arbeit ist nur ein kleiner Teil eines grossen Ganzen. Ich bin zuversichtlich, dass ich in Zukunft im Studium von diesem Ganzen immer mehr erarbeiten kann und mich vielleicht eines Tages zu den Menschen zählen darf, die Einstein verstehen.

Abstract

`cuRRay` ist ein CUDA-beschleunigter Raytracer für Lichtstrahlen in Kerr-Newman-Raumzeit. `cuRRay` wurde in C++ programmiert und kann auf Systemen mit NVIDIA-Grafikkarten kompiliert und ausgeführt werden. Die Software ermöglicht die Konfiguration von Szenen, welche einen Beobachter, ein schwarzes Loch und optionale Objekte enthalten; ausgehend von Szenen können Bilder der Objekte und dem schwarzen Loch mittels Raytracing aus der Perspektive des Beobachters erzeugt werden. Das schwarze Loch ist ein Kerr-Newman-Loch, welches durch die drei Parameter Masse, Drehimpuls und elektrische Ladung definiert wird. Neben der Konfiguration einer Akkretionsscheibe und eines Sternenhimmels können auch mehrere Kugeln in der Raumzeit platziert werden.

Die mit `cuRRay` erzeugten Bilder ermöglichen eine Visualisierung von zwei Aspekten der Raumzeitkrümmung: Ablenkung von Lichtstrahlen und Rotverschiebung. Neben dem Aufbau und den Algorithmen der Software zeigen wir Bilder, welche mit `cuRRay` erstellt wurden. Anhand der Bilder diskutieren wir die Unterschiede zwischen verschiedenen Arten von schwarzen Löchern.

Einleitung

Die *allgemeine Relativitätstheorie* ist eine Gravitationstheorie, welche die klassische Newtonsche Theorie bei sehr hohen Massen-Energiedichten ersetzt, wenn die Newtonsche Theorie ungenau wird. Die allgemeine Relativitätstheorie beschreibt die Gravitation als Folge der *gekrümmten Raumzeit* und bedient sich dazu der *Differentialgeometrie*. Die Theorie wurde 1915 von Albert Einstein begründet.

Sie sagt *Schwarze Löcher* voraus, Regionen in der Raumzeit, aus denen wegen starker Gravitation nichts entweichen kann. Schwarze Löcher können nur indirekt beobachtet werden, da nicht einmal Licht aus ihnen entkommen kann. Sie erscheinen komplett schwarz.

Licht von anderen Körpern wird durch das extreme Gravitationsfeld um ein Schwarzes Loch abgelenkt. Die daraus entstehenden, verzerrten Bilder dieser Körper können verwendet werden, um die Eigenschaften des schwarzen Lochs und der Raumzeit indirekt zu diskutieren. D. Psaltis et al. [PD11.1] verwenden zum Beispiel simulierte Bilder von schwarzen Löchern, um die Emissionsspektren von Gas, welches um dieselben rotiert, vorherzusagen.

Gravitation von C. Misner et al. [MC73.1] und *General Relativity* von R. Wald [WR84.1] sind relativ moderne Werke, welche als ausführliche Einleitung in die allgemeine Relativitätstheorie geeignet sind. Sie bieten die notwendigen Grundlagen, um die Ausbreitung von Licht im relativistischen Gravitationsfeld zu verstehen.

Die Software `cuRRay`, Akronym für ***CU**DA **r**elativistic **Ray**tracer*, wurde im Rahmen dieser Arbeit entwickelt. Sie simuliert die Flugbahnen von einzelnen Photonen, Lichtstrahlen, in relativistischer *Kerr-Newman-Raumzeit*. Kerr-Newman-Raumzeit beschreibt das Gravitationsfeld um ein rotierendes und elektrisch geladenes schwarzes Loch. Neben dem schwarzen Loch kann eine *Akkretionsscheibe* (Ring um das schwarze Loch), verschiedene frei platzierbare Kugeln und ein Sternenhimmel konfiguriert werden. `cuRRay` sendet alle Photonen vom Beobachter in verschiedene Richtungen aus und berechnet ihre Flugbahnen rückwärts, bis sie ein Objekt treffen. So kann das Bild, welches vom Beobachter gesehen wird, rekonstruiert werden. `cuRRay` verwendet *NVIDIA® CUDA*, um mehrere Photonen parallel und hardwarebeschleunigt zu simulieren.

Als Einführung in parallele Berechnungen mit CUDA eignen sich *CUDA by example* von J. Sanders et al. [SJ11.1] und *Professional CUDA C Programming*

von J. Cheng et al. [CJ14.1].

Folgende Ziele wurden verfolgt: die Erarbeitung der allgemeinen Relativitätstheorie und das Programmieren der Software `cuRRay`, welche Bilder von Objekten in der Nähe von schwarzen Löchern erzeugen kann. Es soll nicht eine bessere oder schnellere Software, als bestehende Software entwickelt werden; im Zentrum steht die Anwendung der allgemeinen Relativitätstheorie.

Im Hauptteil wird die notwendige Theorie erarbeitet und die Funktionsweise der Software erläutert. Abschliessend visualisieren wir Raumzeiten mit `cuRRay` ausserhalb verschiedener schwarzer Löcher und diskutieren diese.

Notation

Vorzeichenkonvention Wir verwenden die metrische Signatur $-+++$.

Ereignisse und Objekte Wir bezeichnen Ereignisse mit Skript-Buchstaben; zum Beispiel: $\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$, usw. Objekte, wie zum Beispiel Beobachter oder Himmelskörper werden mit grossen, lateinischen Buchstaben bezeichnet.

Tensoren Für Tensoren verwenden wir zwei verschiedene Notationen: die *koordinatenunabhängige Schreibweise* und die *Indexnotation*.

Fette Zeichen kennzeichnen Tensoren einschliesslich Vektoren und Kovektoren in koordinatenunabhängiger Schreibweise. Zum Beispiel, *Divergenz des Energie-Impuls-Tensor*:

$$\nabla \cdot \boldsymbol{T} = 0. \tag{1}$$

Indizes in dieser Notation werden ebenfalls fett geschrieben. Sie unterscheiden zwischen verschiedener Tensoren einer Menge. Zum Beispiel: *Basisvektoren*:

$$\boldsymbol{e_0}, \boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}. \tag{2}$$

Spielen die Komponenten eines Tensors eine wichtige Rolle, verwenden wir die Indexnotation. Zum Beispiel, *Kontraktion von Riemann-Tensor*:

$$R_{\alpha\beta\gamma}{}^{\beta} = R_{\alpha\gamma}. \tag{3}$$

Indizes von koordinatenabhängigen Tensoren geben an, welche Komponente eines Tensors gemeint ist. Zum Beispiel: *Massen-Energie im lokalen Lorentz-System*:

$$E = T_{00}\,. \tag{4}$$

Indizes in Indexnotation In Indexnotation nehmen griechische Indizes alle Werte von 0 bis 3 an. Der Index 0 ist ein zeitlicher Index, die Indizes 1 bis 3 sind räumliche Indizes. Gelegentlich bezeichnen wir spezifische Werte von Indizes mit den Buchstaben für die Koordinatenbasen. Je nach Index, bzw. Kombination von Indizes, ist eine Komponente *zeitlich*, *räumlich* oder eine Mischung aus den beiden Varianten.

$$g_{tt} = g_{00} \tag{5}$$

ist zum Beispiel die Zeit-Zeit-Komponente des metrischen Tensors. Lateinische

Indizes beziehen sich nur auf die räumlichen Komponenten, sie nehmen Werte von 1 bis 3 an.

Kompakte Indexnotation Für Ableitungen verwenden wir gelegentlich die *kompakte Indexnotation*:

$$\partial_\alpha S \equiv S_{,\alpha} \tag{6}$$

und

$$\nabla_\alpha V^\beta \equiv V^\beta_{;\alpha}\,. \tag{7}$$

Nach allen Indizes, welche hinter einem Komma bzw. Semikolon stehen, wird abgeleitet. Ist ein Index, nach welchem abgeleitet wird, angehoben, bedeutet dies, dass nach dem Ableiten mit dem metrischen Tensor multipliziert wird.

Einsteinsche Summenkonvention Wenn nicht anders gekennzeichnet, verwenden wir die Einsteinsche Summenkonvention. Über Indizes, die innerhalb eines Produkts je einmal *kovariant* (tiefgestellt) und *kontravariant* (hochgestellt) vorkommen, wird eine Summe impliziert:

$$\sum_\alpha v_\alpha v^\alpha = v_\alpha v^\alpha. \tag{8}$$

4er-Vektoren Wenn nicht anders gekennzeichnet, verwenden wir folgende speziell relativistische *4er-Vektoren* anstelle der klassischen dreidimensionalen Vektoren:

4er-Position $\boldsymbol{x} = (ct, x, y, z)$	*4er-Impuls* $\boldsymbol{p} = m\boldsymbol{u}$
4er-Geschwindigkeit $\boldsymbol{u} = d\boldsymbol{x}/d\tau$	*4er-Kraft* $\boldsymbol{F} = d\boldsymbol{p}/d\tau$
4er-Beschleunigung $\boldsymbol{a} = d\boldsymbol{v}/d\tau$	*4er-Stromdichte* $\boldsymbol{j} = (c\rho, j_x, j_y, j_z)$
	ρ = Ladungsdichte
(weitere Ableitungen von $\boldsymbol{x}$)	j = klassische Stromdichte

Geometrische Einheiten Wir verwenden generell geometrische Einheiten. Die Lichtgeschwindigkeit c und die Gravitationskonstante G nehmen den Wert 1 an:

$$\begin{aligned} c &\equiv 1, \\ G &\equiv 1. \end{aligned} \tag{9}$$

Gausssche Einheiten In Kapitel 3 verwenden wir das *Gausssche Einheitensystem* für elektromagnetische Felder. Dadurch verschwinden alle Hinweise auf ϵ_0 und μ_0.

Zitate Wir kennzeichnen Zitate nach folgendem Schema: Die beiden grossen Buchstaben sind die Anfangsbuchstaben des Namens und Vornamens des erstgenannten Autors. Die zweistellige Zahl sind die zwei letzten Stellen des Publikationsjahrs, eine laufende Nummerierung unterscheidet verschiedene Publikationen des gleichen Autors im gleichen Jahr. Beispiel: [EA16.1] für Einstein, Albert (1916), erste verwendete Quelle dieses Jahres.

Kapitel 1

Allgemeine Relativitätstheorie

In diesem Kapitel werden wir die allgemeine Relativitätstheorie darlegen. Wir beginnen mit einer kurzen Diskussion der *speziellen Relativitätstheorie*, welche in der allgemeinen Relativitätstheorie ihre Gültigkeit beibehält. Wir fahren mit dem Grundprinzip der allgemeinen Relativitätstheorie fort, dem *Äquivalenzprinzip*. Anschliessend werden wir auf die gekrümmten Weltlinien von Körpern in der Raumzeit schliessen und diese mathematisch beschreiben. Am Ende des Kapitels verbinden wir Raumzeitkrümmung mit Masse und Energie, um die Einsteinschen Feldgleichungen zu erhalten.

1.1 Spezielle Relativitätstheorie

Die *spezielle Relativitätstheorie* beschreibt, wie sich die Physik für verschiedene relativ zueinander bewegte Beobachter verhält. Sie wurde 1905 von Albert Einstein begründet (siehe [EA05.1]).

Postulate Zentral sind zwei Postulate. Das erste, das *Relativitätsprinzip*, besagt, dass die Gesetze der Physik für alle unbeschleunigten Beobachter gleich sind. Das zweite besagt, dass die Lichtgeschwindigkeit c für alle Beobachter gleich ist:

$$c = 299792458 \text{ m/s}. \tag{1.1.1}$$

Lorentz-Transformation Die Auswirkung der speziellen Relativitätstheorie auf die Physik wird besonders in der *Lorentz-Transformation* sichtbar. Die Lorentz-Transformation rechnet 4er-Vektoren im kartesischen Koordinatensystem eines ruhenden Beobachters in die kartesischen Koordinaten eines relativ mit Geschwindigkeit v entlang der x-Achse bewegten Beobachters um. Die

Lorentz-Transformation kann als Matrix folgendermassen geschrieben werden:

$$\Lambda = \begin{pmatrix} \gamma & -\gamma v & 0 & 0 \\ -\gamma v & \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.1.2}$$

γ ist der Lorentz-Faktor:

$$\gamma = \frac{1}{\sqrt{1 - v^2}}. \tag{1.1.3}$$

Lorentz-Transformationen entlang einer beliebigen Achse sind natürlich ebenfalls möglich. Wir wenden dazu vor der Lorentz-Transformation eine Rotation an, die die gewünschte Achse auf die x-Achse abbildet. Nach der Lorentz-Transformation wenden wir die umgekehrte Rotation an, um wieder die ursprünglichen Achsen zu erhalten.

Eine detaillierte und illustrierte Einführung in die spezielle Relativitätstheorie liefert zum Beispiel [TE91.1].

1.2 Äquivalenzprinzip

Schwaches Äquivalenzprinzip Das *schwache Äquivalenzprinzip* war bereits seit Galilei bekannt. Es besagt, dass alle Körper, unabhängig ihrer Masse, Form oder Struktur im luftleeren Raum gleich fallen: mehrere fallende Körper bewegen sich in einem fallenden System geradlinig und unbeschleunigt, obwohl sie sich von aussen betrachtet auf Parabeln bewegen. Das Prinzip beruht auf der Äquivalenz der *schweren Masse* und *trägen Masse* jedes Körpers. [MC73.1, S. 13-19].

Starkes Äquivalenzprinzip Kraftfelder der Physik können allgemein durch Beobachtung von Testteilchen nachgewiesen werden. So kann zum Beispiel von der Beschleunigung eines geladenen Teilchens auf die elektromagnetische Kraft geschlossen werden. Ein frei fallender Beobachter hingegen kann keine Testteilchen verwenden, um das Gravitationsfeld nachzuweisen, da, wie wir im letzten Abschnitt ausarbeiteten, alle komplett unbeschleunigt erscheinen. Das Gravitationsfeld kann aus diesem Grund lokal nicht gemessen werden [1].

Das *starke Äquivalenzprinzip* besagt, dass ein fallender Beobachter äquivalent zu einem Beobachter abseits aller Gravitationsfelder ist, weil das Gravitationsfeld lokal nicht gemessen werden kann. Das starke Äquivalenzprinzip ist die Basis der allgemeinen Relativitätstheorie. [WR84.1, S. 66-67].

Abbildung 1.2.1 zeigt Gedankenexperimente über die Äquivalenz zweier Beobachter B_1 und B_2, welche ausgehend vom starken Äquivalenzprinzip durchgeführt werden.

Wir wenden uns zuerst dem oberen Teil der Abbildung zu: B_1 befindet

[1] Durch die relative Beschleunigung voneinander entfernter Testteilchen können die *Gezeitenkräfte* des Gravitationsfeld gemessen werden; diese Messung ist aber nicht lokal.

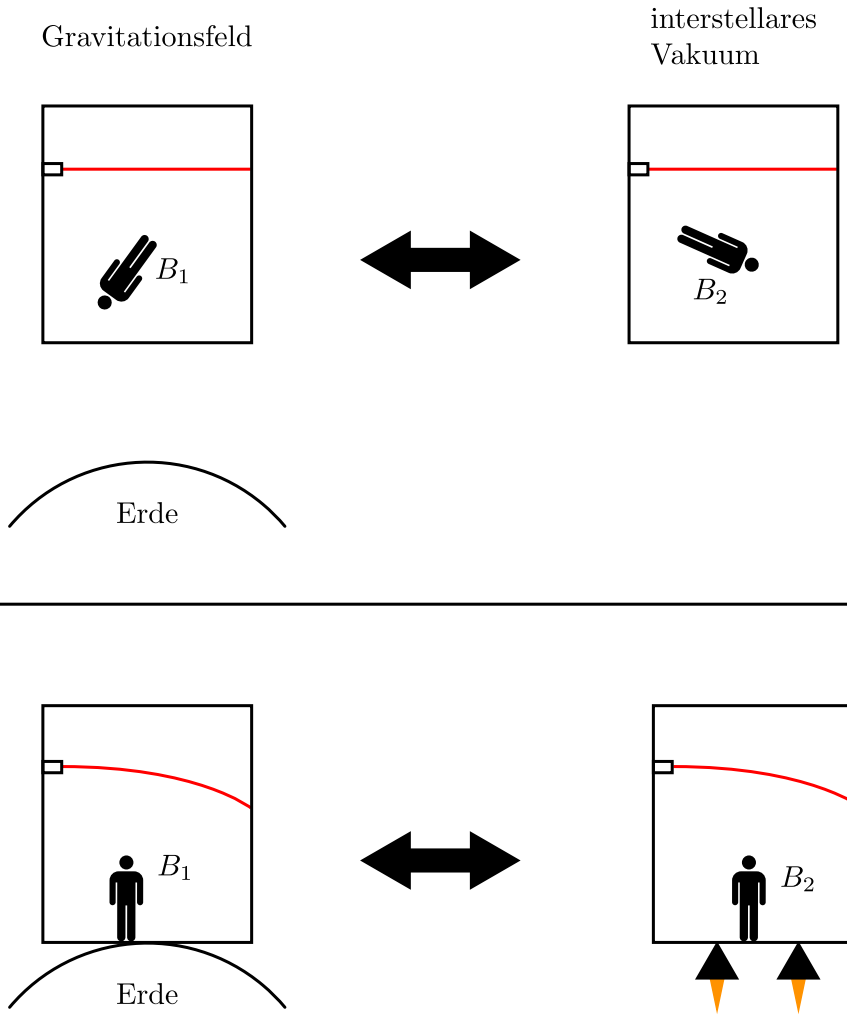


Abbildung 1.2.1: Beobachter B_1 ist äquivalent zu Beobachter B_2. Oben sind beide Beobachter kräftefrei, unten wird B_1 durch die Normalkraft und B_2 durch einen Raketenantrieb beschleunigt. Die rote Linie stellt einen Lichtstrahl dar, wie er von den Beobachtern gesehen wird.

sich im freien Fall nahe der Erde, B_2 im interstellaren Vakuum, wo wir die Gravitation vernachlässigen. Wegen dem starken Äquivalenzprinzip sind beide Beobachter äquivalent und keiner kann ein Gravitationsfeld nachweisen. Sendet B_2 einen Lichtstrahl aus, bewegt sich dieser geradlinig, ohne jegliche Krümmung von ihm aus. B_1 muss den gleichen Effekt beobachten. Sendet nun B_1 orthogonal zu seiner Fallrichtung einen Lichtstrahl aus, muss dieser ebenfalls geradlinig verlaufen; die Photonen des Strahls müssen von einem relativ zum Gravitationsfeld ruhenden Beobachter gesehen mit B_1 mitfallen.

Im unteren Teil der Abbildung sehen wir, dass B_1 auf der Erde steht. Er verspürt eine Normalkraft, die ihn am Fall durch den Erdboden hindert. Diese Normalkraft ist die einzige Änderung zur vorherigen Situation. Beschleunigt nun B_2, respektive setzt er sich einer Kraft aus, sind die beiden Beobachter

wieder äquivalent. Betrachten wir den Lichtstrahl von B_2: er scheint von B_2 aus gebogen zu sein, da das Licht nicht vom Raketenantrieb beschleunigt wird. Weil aber B_1 und B_2 äquivalent sind, muss B_1 seinen Lichtstrahl ebenfalls gebogen sehen. Daraus sehen wir bereits, dass Lichtstrahlen durch Gravitation gekrümmt werden.

1.3 Gekrümmte Raumzeit [2]

Raumzeit In den Relativitätstheorien bezeichnet *Raumzeit* die vierdimensionale Struktur, welche alle Punkte des dreidimensionalen Raums zu jedem Zeitpunkt umfasst. Punkte in der Raumzeit werden *Ereignisse* genannt. In der klassischen Physik ist die Raumzeit *flach*, Weltlinien von unbeschleunigten Teilchen sind stets Geraden.

Inertialsysteme Das Äquivalenzprinzip führt dazu, dass alle fallenden Beobachter gleichwertig sind, genau wie in der klassischen Physik alle unbeschleunigten Beobachter gleichwertig sind. Wir nennen die Systeme solcher Beobachter *lokale Inertialsysteme* oder *lokale Lorentz-Systeme*.

Im Gegensatz zu den Inertialsystemen der speziellen Relativitätstheorie sind die lokalen Inertialsysteme der allgemeinen Relativitätstheorie nur lokal inertial; dies bedeutet hauptsächlich, dass über endliche Distanzen Gezeitenkräfte messbar sind. Lokale Inertialsysteme haben die Eigenschaft, dass in ihnen keine Beschleunigung des Systems gemessen werden kann. Somit kann auch keine Gravitationskraft, welche für die Beschleunigung der Inertialsysteme verantwortlich wäre, gemessen werden: in der allgemeinen Relativitätstheorie betrachten wir Gravitation nicht als Kraft. Damit ein Beobachter an der Oberfläche einer grossen Masse trotzdem die Bahnen von fallenden Körpern als gekrümmt auffasst, schlug Einstein vor, dass die Raumzeit gekrümmt ist.

Die spezielle Relativitätstheorie [3] gilt in allen Inertialsystemen und lokal daher auch in gekrümmter Raumzeit.

Geodäten Ein fallender Beobachter ist wie eine Ameise, die sich auf der Oberfläche eines Apfels bewegt: obwohl sie sich immer geradeaus bewegt, folgt sie von aussen betrachtet einer gekrümmten Bahn [4]. Analog dazu fällt ein Beobachter lokal in einer geraden Linie, weil wegen dem Äquivalenzprinzip keine Beschleunigung gemessen werden kann. Global bewegt er sich hingegen auf einer gekrümmten Bahn.

Die *möglichst geraden Weltlinien* in gekrümmter Raumzeit heissen *Geodäten*. Alle frei fallenden Objekte bewegen sich auf Geodäten.

Massen-Energie Die Raumzeitkrümmung wird durch die Massen-Energie im Universum erzeugt, analog zum Newtonschen Gravitationsfeld, welches eben-

[2] Dieses Kapitel basiert auf [MC73.1, Kapitel 1].

[3] Siehe [EA05.1]

[4] Siehe [MC73.1, S. 3-5] für die Kurzgeschichte „*The Parable of the Apple*“, welche dieses Konzept verbildlicht.

falls durch Massen erzeugt wird. Abbildung 1.3.1 illustriert die Krümmung der Raumzeit durch einen Stern und die daraus resultierende gekrümmte Bahn eines Planeten.

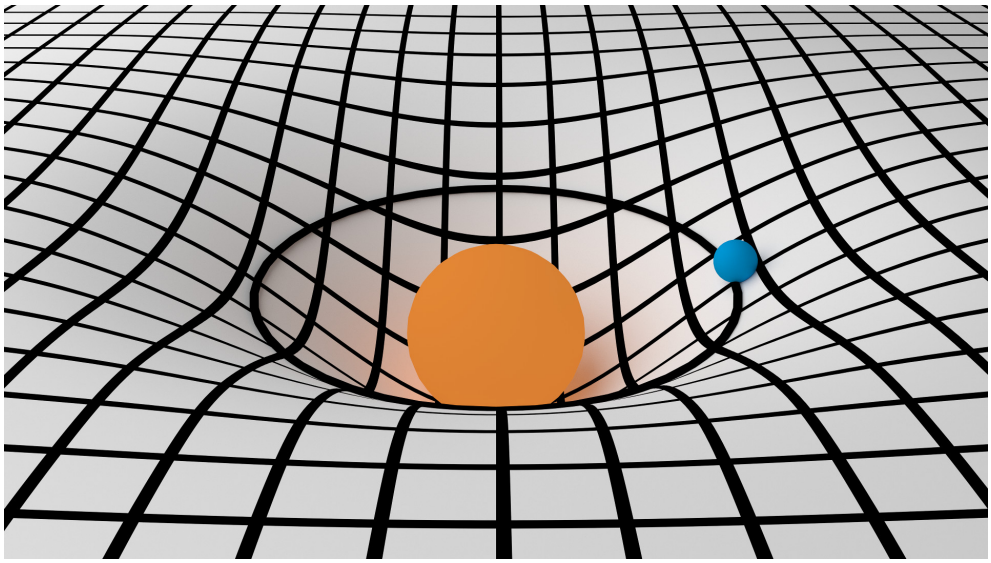


Abbildung 1.3.1: Die Raumzeit, dargestellt als zweidimensionale Fläche, wird durch die grosse Masse des orangen Sterns in der Mitte gekrümmt. Aus diesem Grund ist die Bahn des Planeten, dargestellt durch die kleinere blaue Kugel, gekrümmt. Die Krümmung der Raumzeit durch den Planeten wird vernachlässigt.

Die *Einsteinschen Feldgleichungen* (siehe Kapitel 1.10) setzen die Massen-Energie-Dichte gleich der Krümmung der Raumzeit und beschreiben somit die Wechselwirkung zwischen Massen-Energie und der Raumzeitkrümmung: (1) Materie und Energie krümmen die Raumzeit, (2) die Raumzeitkrümmung beeinflusst die Bewegung von Materie und Energie.

Die allgemeine Relativitätstheorie, die geometrische Betrachtung der Gravitation, setzt das Äquivalenzprinzip um. Sie ist extrem erfolgreich: Unzählige Messungen und Experimente bestätigen heute die Theorie – zum Beispiel die Messung von Gravitationswellen im September 2015 durch den LIGO-Detektor, einem Phänomen, das hundert Jahre zuvor von Einstein vorhergesagt wurde (siehe [AB16.1]) [5].

Verschiedene Ansätze, die Gravitation in der flachen Raumzeit der speziellen Relativitätstheorie zu beschreiben, wurden unternommen, doch alle scheiterten oder führten zur allgemeinen Relativitätstheorie (siehe [MC73.1, Kapitel 7] für eine ausführliche Übersicht über verschiedene gescheiterte Ansätze).

[5] Viele weitere Messungen, wie die Perihelpräzession des Merkur (siehe [WR84.1, S. 143]) oder Lichtablenkung durch die Sonne (siehe [WR84.1, S. 146]), bestätigen die Theorie.

1.4 Tensoren

In der allgemeinen Relativitätstheorie verwenden wir *Tensoren* für die Darstellung von physikalischen Grössen. Tensoren sind für unsere Zwecke vor allem eine Verallgemeinerung von Vektoren und Kovektoren. Sie wurden bereits 1916 von Einstein als passend vorgeschlagen (siehe [EA16.1]).

Definition Ein Rang-(k, l)-Tensor ist eine multilineare Funktion $\boldsymbol{A}$, die k Kovektoren $\boldsymbol{\omega}$ und l Vektoren $\boldsymbol{v}$ auf eine reelle Zahl s abbildet:

$$s = \boldsymbol{A}(\boldsymbol{\omega_1}, \ldots, \boldsymbol{\omega_k}, \boldsymbol{v^1}, \ldots, \boldsymbol{v^l}). \tag{1.4.1}$$

Tensoren sind an ein Ereignis gebunden. Wegen der Raumzeitkrümmung ist es nicht trivial, Tensoren von verschiedenen Ereignissen zu vergleichen.

Tangentialraum Alle Tensoren ausser Rang-$(0, 0)$-Tensoren (Skalaren) operieren auf Elementen des *Tangentialraums*, welcher in gekrümmter Raumzeit an jedem Ereignis unterschiedlich ist. Nur Tensoren, welche auf den Vektoren und Kovektoren des gleichen Tangentialraums operieren, können verglichen oder miteinander verrechnet werden. Vgl. [WR84.1, S. 14-18].

Streng genommen ist der Tangentialraum ein Vektorraum, das heisst, Kovektoren sind keine Elemente des Tangentialraums, sondern besitzen ihren eigenen *Kovektorraum*, den *Kotangentialraum*. Beide Räume können jedoch durch die *Metrik* (siehe Kapitel 1.6) ineinander übergeführt werden. Aus diesem Grund sprechen wir von nun an nur noch vom Tangentialraum.

Der Tangentialraum kann als eine Art lineare Approximation zur gekrümmten Raumzeit verstanden werden: Verschiebungen um den Ursprung des Tangentialraums entsprechen zur ersten Ordnung Verschiebungen in der gekrümmter Raumzeit um das Ereignis, zu dem der Tangentialraum gehört. Diese Idee zeigt sich in der Definition des Tangentialraums: Die Elemente des Tangentialraums sind Tangentenvektoren zu Kurven in gekrümmter Raumzeit. Siehe [WR84.1, S. 14-18].

Indizes Sobald bekannt ist, welche Werte ein Rang-(k, l)-Tensor für alle (k, l)-Tupel der *Basisvektoren* und *Basis-1-Formen* eines Koordinatensystems annimmt, können die Komponenten des Tensors in derselben Basis berechnet werden (vgl. [MC73.1, S. 53]):

$$A^{\alpha_1 \cdots \alpha_k}{}_{\beta_1 \cdots \beta_l} = \boldsymbol{A}(\boldsymbol{e_{\beta_1}}, \ldots, \boldsymbol{e_{\beta_k}}, \boldsymbol{\epsilon^{\alpha_1}}, \ldots, \boldsymbol{\epsilon^{\alpha_l}}). \tag{1.4.2}$$

Wegen der Linearität von Tensoren kann (1.4.1) als Produkt von Komponenten geschrieben werden:

$$s = A^{\alpha_1 \cdots \alpha_k}{}_{\beta_1 \cdots \beta_l} \omega_{\alpha_1} \cdots \omega_{\alpha_k} v^{\beta_1} \cdots v^{\beta_l}. \tag{1.4.3}$$

Vgl. [MC73.1, S. 75].

Tensorprodukt Wir können zwei Tensoren miteinander multiplizieren, um einen neuen Tensor von höherem Rang zu erhalten. Diese Operation wird

Tensorprodukt genannt. Zum Beispiel das Tensorprodukt aus einem Vektor $\boldsymbol{v}$ und einem Kovektor $\boldsymbol{\omega}$:

$$\boldsymbol{v} \otimes \boldsymbol{\omega} = v^{\alpha}\omega_{\beta} = A^{\alpha}{}_{\beta}. \tag{1.4.4}$$

Kontraktion Wir sprechen von *Kontraktion* (auch *Verjüngung*), wenn in einem Tensorprodukt ein oder mehrere Indizes durch Aufsummierung verschwinden. Zum Beispiel:

$$B_{\beta} = A_{\alpha\beta}v^{\alpha}. \tag{1.4.5}$$

Gradient, Richtungsableitung und Divergenz Diese drei Operationen werden durch den *Ableitungsoperator* ∇ ermöglicht:

1. Gradient:
$$\nabla \boldsymbol{T} = \nabla_{\alpha} A^{\beta_1 \ldots \beta_k}{}_{\gamma_1 \ldots \gamma_l}. \tag{1.4.6}$$

2. Richtungsableitung
$$\nabla_{\boldsymbol{k}} \boldsymbol{A} = k^{\alpha} \nabla_{\alpha} A^{\beta_1 \ldots \beta_k}{}_{\gamma_1 \ldots \gamma_l}. \tag{1.4.7}$$

3. Divergenz im Bezug zum Index β_1
$$\nabla \cdot \boldsymbol{A} = \nabla_{\beta_1} A^{\beta_1 \ldots \beta_k}{}_{\gamma_1 \ldots \gamma_l}. \tag{1.4.8}$$

Es soll angemerkt werden, dass ∇ kein Vektor oder Kovektor ist; der angehängte Index zeigt nur an, dass das Ergebnis des ∇-Operators eine kovariante Komponente mehr als der Operand hat.

1.5 Koordinaten

Ereignisse und Koordinatensysteme Wir halten verschiedene Ereignisse in der Raumzeit auseinander, indem wir ihnen eindeutige Namen zuweisen. Sind die Namen systematisch verteilt, sprechen wir von einem *Koordinatensystem*. Wir verwenden Zahlentupel als Koordinaten. Ein Koordinatensystem ist *kontinuierlich*, wenn zwei infinitesimal nebeneinander liegende Ereignisse sich durch ein infinitesimales Zahlentupel unterscheiden. Vgl. [MC73.1, S. 5-13].

Koordinaten machen allgemein keine Aussage über die Distanz zwischen zwei Ereignissen. In flacher Raumzeit können zwar solche Koordinaten einfach gewählt werden, in gekrümmter Raumzeit ist dies jedoch nicht generell möglich. Es ist sogar möglich, dass nicht die gesamte Raumzeit mit kontinuierlichen Koordinaten abgedeckt werden kann und dass *Koordinatensingularitäten*, Ereignisse mit keinen oder mehreren Koordinaten, entstehen. Vgl. [WR84.1, S. 11-14].

Lokale Koordinaten Wir können stets Koordinaten wählen, welche nur lokal gültig sind. Natürlich können wir in einem lokalen Inertialsystem die bekannten Koordinaten der speziellen Relativitätstheorie wählen.

Spezielle Kovarianz Physikalische Gesetze müssen in allen Koordinaten und jedem Inertialsystem gelten. Dieses Prinzip heisst *Prinzip der speziellen Kovarianz*. Vgl. [WR84.1, S. 58].

Koordinatentransformationen Tensoren verändern sich unter einer Koordinatentransformation $\tilde{x}^\alpha(x^\beta)$ folgendermassen:

$$\tilde{A}^\alpha{}_\beta = \sum_{\gamma,\delta} A^\gamma{}_\delta \frac{\partial \tilde{x}^\alpha}{\partial x^\gamma} \frac{\partial x^\delta}{\partial \tilde{x}^\beta}. \tag{1.5.1}$$

Für jeden weiteren ko- oder kontravarianten Index werden entsprechende Terme angehängt. Gleichung (1.5.1) sorgt dafür, dass Skalare von der Form

$$s = A_{\alpha_1 \dots \alpha_k} B^{\alpha_1 \dots \alpha_k} \tag{1.5.2}$$

durch die Koordinatentransformation unverändert bleiben. Daraus folgt, dass, wenn physikalische Gesetze als Äquivalenzen von Tensoren geschrieben werden, die spezielle Kovarianz garantiert ist. Vgl. [EA16.1, S. 779-780] und [WR84.1, S. 56-59].

1.6 Metrischer Tensor

In diesem Kapitel führen wir den *metrischen Tensor* ein, welcher uns ermöglichen wird, die Krümmung der Raumzeit zu quantifizieren.

Geometrie der Raumzeit Sind die infinitesimalen Distanzen zwischen einem zentralen Ereignis und seinen Nachbarereignissen bekannt, ist lokal die Form der Raumzeit definiert. Gleichermassen können wir für die gesamte Raumzeit vorgehen: die infinitesimalen Distanzen zwischen allen Ereignissen bestimmen die Geometrie der Raumzeit. Vgl. [MC73.1, S. 309].

Metrischer Tensor Der *metrische Tensor* $\boldsymbol{g}$ ist ein Rang-(0,2)-Tensor und definiert an jedem Ereignis in der Raumzeit das Skalarprodukt zweier Vektoren $\boldsymbol{v}$ und $\boldsymbol{w}$ im Tangentialraum (vgl. [MC73.1, S. 305]). Er ist ein Längen- und Winkelmass; besonders sichtbar wird dies in euklidischer Geometrie:

$$\begin{aligned} \boldsymbol{v} \cdot \boldsymbol{v} &= |\boldsymbol{v}|^2 \qquad \text{(Längenmass)}, \\ \boldsymbol{v} \cdot \boldsymbol{w} &= |\boldsymbol{v}||\boldsymbol{w}| \cdot \cos \angle(\boldsymbol{v}, \boldsymbol{w}) \qquad \text{(Winkelmass)}. \end{aligned} \tag{1.6.1}$$

Metrik Die Vorschrift, wie das Skalarprodukt berechnet werden muss, heisst *Metrik* und wird häufig in Form der quadrierten Länge ds^2 eines infinitesimalen Vektors $\boldsymbol{dx}$ angegeben:

$$ds^2 = g_{\alpha\beta} dx^\alpha dx^\beta, \tag{1.6.2}$$

wobei $g_{\alpha\beta}$ die Komponenten des metrischen Tensors sind. $\boldsymbol{g}$ ist symmetrisch, das heisst, dass in $g_{\alpha\beta}$ α und β vertauscht werden können, ohne dass sich der Wert der Komponente ändert. Der metrische Tensor hat 16 Komponenten.

Wegen der Symmetrie sind jedoch maximal zehn Komponenten voneinander unabhängig. Vgl. [MC73.1, S. 310].

Metrik in flacher Raumzeit In flacher Raumzeit sind die Komponenten von $\boldsymbol{g}$ relativ zu einem orthonormalen Koordinatensystem überall $g_{\alpha\beta} = \text{diag}(-1, 1, 1, 1)$. Die Metrik hat eine *Lorentz-Signatur*; das heisst, dass das Vorzeichen für die zeitliche Komponente unterschiedlich zu jenem der Raum-Komponenten ist. ds^2 wird dann zum bekannten infinitesimalen Raumzeitintervall dI^2 der speziellen Relativitätstheorie.

Gravitationspotential In der allgemeinen Relativitätstheorie ist der Metrische Tensor an jedem Ereignis unterschiedlich. Die flache Raumzeit der speziellen Relativitätstheorie ist bloss ein Spezialfall. Die Komponenten $g_{\alpha\beta}$ des Tensorfelds $\boldsymbol{g}$ können als *Potentiale*, welche Gravitationseffekte erzeugen, verstanden werden (vgl. z.B. [MC73.1, S. 436]).

Anheben und Absenken von Indizes Indizes von Vektoren, Kovektoren und Tensoren im Allgemeinen können durch Multiplikation mit dem metrischen Tensor angehoben und abgesenkt werden:

$$\omega_\alpha = g_{\alpha\beta} v^\beta, \tag{1.6.3}$$

$$v^\alpha = g^{\alpha\beta} \omega_\beta. \tag{1.6.4}$$

$g^{\alpha\beta}$ ist der *inverse metrische Tensor* und wird über das *Kroenecker-Delta* $\delta^\alpha{}_\gamma$ definiert:

$$g^{\alpha\beta} g_{\beta\gamma} = \delta^\alpha{}_\gamma. \tag{1.6.5}$$

Vgl. [WR84.1, S. 25].

1.7 Geodäten

In diesem Kapitel werden wir die *Geodätengleichung* aufstellen. Wir verwenden diese in der Software, um die Geodäten von Lichtteilchen zu berechnen.

Definition Ein Beobachter befindet sich auf einer Geodäte, wenn er sich lokal unbeschleunigt auf einer Geraden bewegt. Dabei verändert sich sein Geschwindigkeitsvektor lokal nicht. Entlang der Geodäten, sprich an den Ereignissen, welche die Geodäte bilden, gilt für den Tangentenvektor $\boldsymbol{u} = \boldsymbol{dx}/d\lambda$ der Geodäte

$$\nabla_{\boldsymbol{u}} \boldsymbol{u} = \nabla_\lambda \boldsymbol{u} = 0. \tag{1.7.1}$$

Die Parametrisierung λ der Geodäten ist willkürlich. Vgl. [MC73.1, S. 262-263]. Gleichung (1.7.1) ist die abstrakte Form der Geodätengleichung.

Ableitungsoperator und paralleler Transport Die Geodätengleichung (1.7.1) wird über den *Ableitungsoperator* ∇ definiert. Die Richtungsableitung $\nabla_{\boldsymbol{u}}$ berechnet die Änderungsrate eines beliebigen Tensorfelds $\boldsymbol{A}$ entlang $\boldsymbol{u}$ an einem Ereignis $\boldsymbol{x^0}$. Dazu muss $\boldsymbol{A}$ an den Positionen $\boldsymbol{x} = \boldsymbol{x^0}$ und $\boldsymbol{x} = \boldsymbol{x^0} + \boldsymbol{dx}$

ausgewertet werden:

$$\nabla_{\boldsymbol{u}} \boldsymbol{A}(\boldsymbol{x}^{\mathbf{0}}) = \lim_{k \to 0} \frac{\boldsymbol{A}(\boldsymbol{x} = \boldsymbol{x}^{\mathbf{0}} + \boldsymbol{u}k) - \boldsymbol{A}(\boldsymbol{x} = \boldsymbol{x}^{\mathbf{0}})}{k}, \tag{1.7.2}$$

wobei $\lim_{k \to 0} \boldsymbol{u}k = \boldsymbol{dx}$. Weil aber $\boldsymbol{A}(\boldsymbol{x} = \boldsymbol{x}^{\mathbf{0}})$ und $\boldsymbol{A}(\boldsymbol{x} = \boldsymbol{x}^{\mathbf{0}} + \boldsymbol{dx})$ nicht auf dem gleichen Tangentialraum operieren, muss zuerst einer der beiden Tensoren *entlang der Krümmung der Raumzeit*, sprich auf einer Geodäten, verschoben werden. Dieser Vorgang wird *paralleler Transport* genannt. Zwischen den zwei benachbarten Ereignissen existiert genau eine Verbindungsgeodäte, der parallele Transport entlang einer infinitesimalen Strecke ist eindeutig definiert. Ereignisse, welche nicht benachbart sind, können hingegen mit mehreren Geodäten verbunden werden. Vgl. [MC73.1, S. 249].

Abbildung 1.7.1 zeigt den parallelen Transport eines Vektors $\boldsymbol{v}$. Bevor der Vektor an den Positionen x und $x + dx$ verglichen werden kann, wird $\boldsymbol{v}(x)$ parallel zu $x + dx$ transportiert und wird zu $\boldsymbol{v}(x)_{||}$.

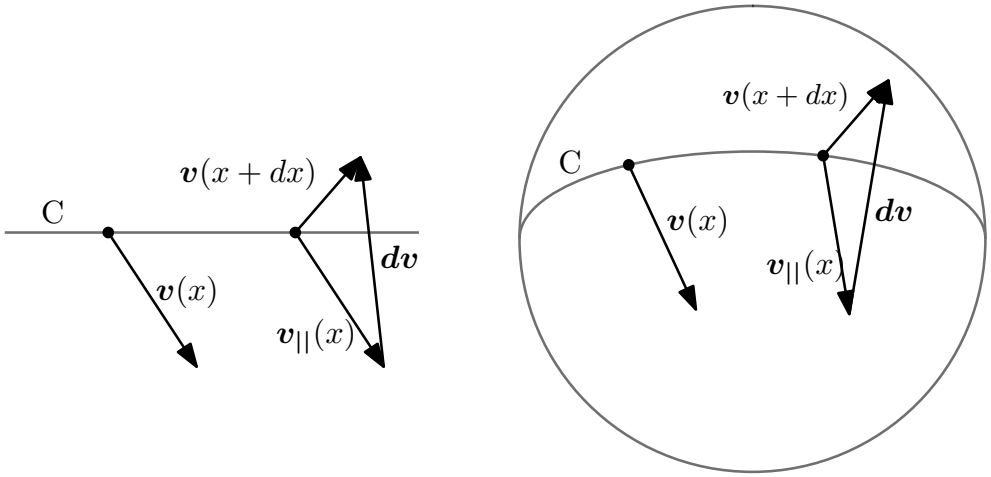


Abbildung 1.7.1: Geometrische Darstellung des parallelen Transports eines Vektors $\boldsymbol{v}(x)$ entlang der Geodäte C in flacher und gekrümmter Raumzeit

Rechenregeln Für den Ableitungsoperator gelten verschiedene Rechenregeln, insbesondere die *Summenregel*

$$\nabla(\boldsymbol{A} + \boldsymbol{B}) = \nabla \boldsymbol{A} + \nabla \boldsymbol{B}, \tag{1.7.3}$$

die *Produktregel*

$$\nabla(\boldsymbol{A} \otimes \boldsymbol{B}) = (\nabla \boldsymbol{A}) \otimes \boldsymbol{B} + \boldsymbol{A} \otimes (\nabla \boldsymbol{B}) \tag{1.7.4}$$

und die *Kettenregel* (hier nicht aufgeführt) für beliebige Tensoren $\boldsymbol{A}$ und $\boldsymbol{B}$. Die Summenregel gilt natürlich nur, wenn die Ränge von $\boldsymbol{A}$ und $\boldsymbol{B}$ gleich sind. Vgl. [MC73.1, S. 257-258]

Äquivalenzprinzip Durch das Äquivalenzprinzip gilt

$$\nabla \boldsymbol{g} = 0. \tag{1.7.5}$$

Dies ist die mathematische Formulierung, dass lokal kein Gravitationsfeld gemessen werden kann. Vgl. [WR84.1, S. 35].

Paralleler Transport von Vektoren Das Skalarprodukt zweier parallel transportierten Vektoren $\boldsymbol{v}$ und $\boldsymbol{w}$ verändert sich durch den parallelen Transport nicht:

$$\begin{aligned}\nabla_{\boldsymbol{u}}(\boldsymbol{v}\cdot\boldsymbol{w}) &= [\nabla_\gamma(g_{\alpha\beta}v^\alpha w^\beta)]u^\gamma \\ &= [(\nabla_\gamma g_{\alpha\beta})v^\alpha w^\beta + (\nabla_\gamma v^\alpha)g_{\alpha\beta}w^\beta + (\nabla_\gamma w^\beta)g_{\alpha\beta}v^\alpha]u^\gamma \\ &= 0. \end{aligned} \tag{1.7.6}$$

Der erste Term verschwindet wegen dem Äquivalenzprinzip (Gleichung (1.7.5)), die restlichen Terme durch die Forderung, dass $\boldsymbol{v}$ und $\boldsymbol{w}$ parallel transportiert werden.

Insbesondere verändert sich die Magnitude eines parallel transportierten Vektors $\boldsymbol{v}$ nicht:

$$\nabla_\gamma(\boldsymbol{v}\cdot\boldsymbol{v}) = 0. \tag{1.7.7}$$

Christoffelsymbole In Indexnotation ist der Ableitungsoperator über die *Christoffelsymbole* und den *gewöhnlichen Ableitungsoperator* ∂ definiert:

$$\nabla_\gamma A^\alpha{}_\beta = \partial_\gamma A^\alpha{}_\beta + \Gamma^\alpha{}_{\mu\gamma}A^\mu{}_\beta - \Gamma^\mu{}_{\beta\gamma}A^\alpha{}_\mu. \tag{1.7.8}$$

Für jeden ko- oder kontravarianten Index wird ein entsprechender Term angehängt. $\Gamma^\alpha{}_{\beta\gamma}$ sind *Christoffelsymbole* zweiter Art. Sie zeigen, wie sich die Basisvektoren eines Koordinatensystems entlang sich selber verändern. Da die Christoffelsymbole von der Wahl der Koordinaten abhängen, sind sie keine Tensoren und erfüllen Gleichung (1.5.1) nicht. Christoffelsymbole werden aus ersten Ableitungen der Metrik konstruiert:

$$\Gamma^\alpha{}_{\beta\gamma} = g^{\alpha\delta}\frac{1}{2}(\partial_\gamma g_{\delta\beta} + \partial_\beta g_{\delta\gamma} - \partial_\delta g_{\beta\gamma}). \tag{1.7.9}$$

Die Christoffelsymbole sind das Pendant zum Gravitations-Kraftfeld in der klassischen Theorie. Vgl. [WR84.1, S. 35-36].

Geodätengleichungen Die Geodätengleichungen der einzelnen Geschwindigkeitskomponenten in Indexschreibweise sind:

$$\frac{d^2x^\alpha}{d\lambda^2} + \sum_{\beta,\gamma}\Gamma^\alpha{}_{\beta\gamma}\frac{dx^\beta}{d\lambda}\frac{dx^\gamma}{d\lambda} = 0. \tag{1.7.10}$$

Vgl. [MC73.1, S. 263]. Die Gleichungen (1.7.10) werden von `cuRRay` für Photonen in Kerr-Newman-Raumzeit gelöst. Die dazu notwendigen Christoffelsymbole werden im Anhang B hergeleitet.

Arten Mit einem gegeben Tangentenvektor u^α einer Geodäte unterscheiden wir je nach Vorzeichen von $g_{\alpha\beta}u^\alpha u^\beta$ verschiedene Arten von Geodäten. Eine Geodäte kann niemals ihre Art wechseln. Ist das Produkt negativ, handelt

es sich um eine *zeitartige* Geodäte; massereiche Teilchen bewegen sich auf solchen. Ist das Produkt positiv, ist die Geodäte *raumartig*; keine bekannten Teilchen bewegen sich auf solchen Geodäten. Ist das Produkt null, ist die Geodäte *lichtartig*; masselose Teilchen wie Photonen bewegen sich auf solchen Geodäten. Vgl. [WR84.1, S. 44].

1.8 Riemann-Tensor

In diesem Kapitel betrachten wir den Riemann-Tensor, ein weiteres Mass für die Raumzeitkrümmung. Er wird ausserdem für das Aufstellen der Feldgleichungen benötigt.

Relative Beschleunigung von Geodäten Der *Riemann-Tensor* $\boldsymbol{R}$ ist ein Mass für die relative Beschleunigung von benachbarten, ursprünglich parallelen Geodäten. Betrachten wir Abbildung 1.8.1: eine Familie $\gamma_n(\lambda)$ von Geodäten durchziehen die Raumzeit. Geodäten unterscheiden sich durch den Auswahlparameter n. Für jede Geodäte ist $\boldsymbol{t} = \partial/\partial\lambda$ der Tangentenvektor und $\boldsymbol{\xi} = \partial/\partial n$ der *Ablenkungsvektor*.

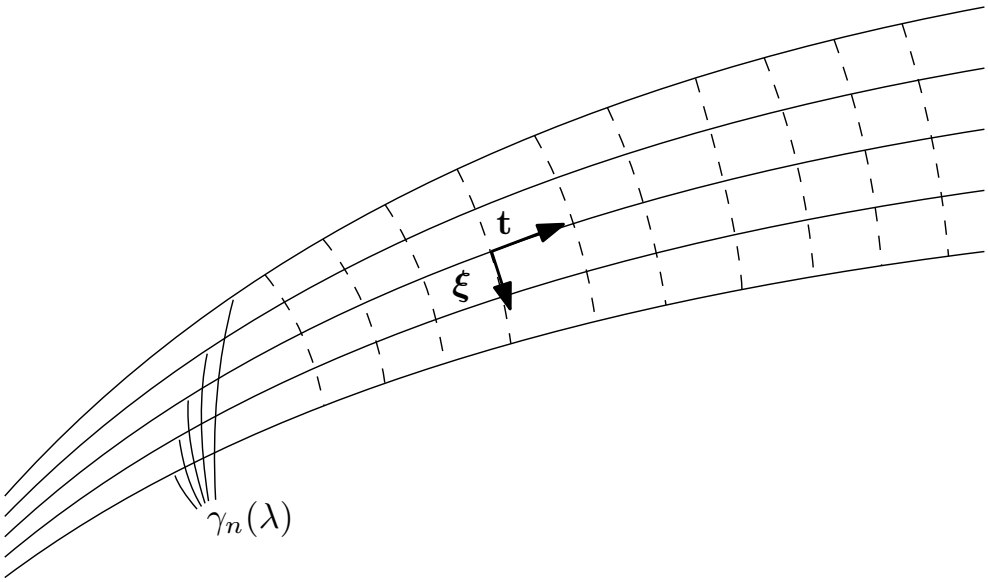


Abbildung 1.8.1: Eine Familie $\gamma_n(\lambda)$ von Geodäten: der Tangentenvektor $\boldsymbol{t} = \partial/\partial\lambda$ und Ablenkungsvektor $\boldsymbol{\xi} = \partial/\partial n$ sind an jedem Punkt auf jeder Geodäte definiert. Die Änderung zweiter Ordnung von $\boldsymbol{\xi}$ ist die relative Beschleunigung von benachbarten Geodäten.

Die Änderung von $\boldsymbol{\xi}$ zweiter Ordnung entspricht der gesuchten Beschleunigung von benachbarten Geodäten. Sie kann mit dem Rang-(1,3) Riemann-Tensor $\boldsymbol{R}$ ausgedrückt werden:

$$\nabla_{\boldsymbol{t}} \nabla_{\boldsymbol{t}} \boldsymbol{\xi} + \boldsymbol{R}(\boldsymbol{t}, \boldsymbol{\xi}, \boldsymbol{t}, \cdot\,) = 0. \tag{1.8.1}$$

Vgl. [MC73.1, S. 265-270].

Pfadabhängigkeit von parallelem Transport Neben der relativen Geodätenbeschleunigung quantifiziert der Riemann-Tensor auch die Pfadabhängigkeit von parallelem Transport. Der *Kommutator* der Ableitung entlang zweier Koordinatenvektorfelder $\boldsymbol{A}$ und $\boldsymbol{B}$ ist gegeben durch:

$$\boldsymbol{R}(\boldsymbol{C}, \boldsymbol{A}, \boldsymbol{B}, \cdot\,) = [\nabla_{\boldsymbol{A}}, \nabla_{\boldsymbol{B}}]\boldsymbol{C}. \tag{1.8.2}$$

Gleichung (1.8.2) erzeugt einen Vektor, welcher angibt, wie stark sich $\nabla_{\boldsymbol{A}}\nabla_{\boldsymbol{B}}\boldsymbol{C}$ von $\nabla_{\boldsymbol{B}}\nabla_{\boldsymbol{A}}\boldsymbol{C}$ unterscheidet. Vgl. [MC73.1, S. 277-281].

Komponenten Die Komponenten des Riemann-Tensors sind

$$R_{\alpha\beta\gamma}{}^{\delta} = \partial_\beta \Gamma^{\delta}{}_{\alpha\gamma} - \partial_\alpha \Gamma^{\delta}{}_{\beta\gamma} + \Gamma^{\epsilon}{}_{\alpha\gamma}\Gamma^{\delta}{}_{\epsilon\beta} - \Gamma^{\epsilon}{}_{\beta\gamma}\Gamma^{\delta}{}_{\epsilon\alpha}. \tag{1.8.3}$$

Vgl. [WR84.1, S. 47-48].

1.9 Energie-Impuls-Tensor

In diesem Kapitel werden wir den *Energie-Impuls-Tensor* einführen, welcher die Massen-Energie an einem bestimmten Ereignis in der Raumzeit beschreibt. Am Ende des Kapitels werden wir in der Lage sein, die Einsteinschen Feldgleichungen aufzustellen.

Aufbau des Energie-Impuls-Tensors Der symmetrische Rang-$(0, 2)$ Energie-Impuls-Tensor $\boldsymbol{T}$ beschreibt Masse und Energie an einem Ereignis in der Raumzeit. Die Energie des Gravitationsfeldes ist nicht im Energie-Impuls-Tensor enthalten.

$$\boldsymbol{T}(\boldsymbol{v}, \boldsymbol{w}) = T_{\alpha\beta}\, v^\alpha w^\beta \tag{1.9.1}$$

beschreibt je nach Wahl der Vektoren $\boldsymbol{v}$ und $\boldsymbol{w}$ verschiedene Aspekte von Masse, Energie, Impuls, Druck und mechanische Spannung. In einem lokalen Lorentz-System hat $\boldsymbol{T}$ folgende Komponenten:

$$\begin{aligned} T_{00} &\equiv \text{Massen-Energiedichte} \\ T_{0a} = T_{a0} &\equiv a\text{-Komponente der Impulsdichte} \\ T_{ab} = T_{ba} &\equiv ab\text{-Komponente des Maxwellschen Spannungstensors.} \end{aligned} \tag{1.9.2}$$

Vgl. [MC73.1, S. 131].

Energie- und Impulserhaltung Damit die Energie- und Impulserhaltung gilt, muss die *Divergenz* von $\boldsymbol{T}$ null sein:

$$\nabla \cdot \boldsymbol{T} = \nabla^\alpha T_{\alpha\beta} = 0. \tag{1.9.3}$$

Dadurch ist gewährleistet, dass keine Quellen bzw. Senken von Massen-Energie existieren. Vgl. [MC73.1, S. 146].

1.10 Einsteinsche Feldgleichungen

Die *Einsteinschen Feldgleichungen* beschreiben die Wechselwirkung zwischen Raumzeitkrümmung einerseits und Massen-Energie andererseits. Durch Lösen der Gleichungen können die Komponenten $g_{\alpha\beta}$ des metrischen Tensors gefunden werden.

Ricci-Tensor und Ricci-Krümmungsskalar Wenn wir den Riemann-Tensor über den zweiten und letzten Index kontrahieren, erhalten wir den *Ricci-Tensor*:

$$R_{\alpha\gamma} = R_{\alpha\beta\gamma}{}^{\beta}. \tag{1.10.1}$$

Durch erneute Kontraktion erhalten wir den *Ricci-Krümmungsskalar*:

$$R = g^{\alpha\beta} R_{\alpha\beta} = R_{\alpha}{}^{\alpha}. \tag{1.10.2}$$

Vgl. [WR84.1, S. 40].

Einstein-Tensor Der *Einstein-Tensor* $\boldsymbol{G}$ wird aus dem Ricci-Tensor und dem Ricci-Skalar gebildet:

$$G_{\alpha\beta} = R_{\alpha\beta} - \frac{1}{2} R g_{\alpha\beta}. \tag{1.10.3}$$

$\boldsymbol{G}$ ist symmetrisch. Er erfüllt die *Bianchi-Relation*:

$$\nabla \cdot \boldsymbol{G} = \nabla^{\alpha} G_{\alpha\beta} = 0. \tag{1.10.4}$$

Feldgleichungen Die Einsteinschen Feldgleichungen sind

$$\boldsymbol{G} = \frac{8\pi G}{c^4} \boldsymbol{T} \tag{1.10.5}$$

beziehungsweise

$$R_{\alpha\beta} - \frac{1}{2} R g_{\alpha\beta} = \frac{8\pi G}{c^4} T_{\alpha\beta}\,. \tag{1.10.6}$$

Vgl. [WR84.1, S. 72].

Energie-Impuls-Erhaltung Durch die Bianchi-Relation $\nabla \cdot \boldsymbol{G} = 0$ folgt $\nabla \cdot \boldsymbol{T} = 0$; dies führt zur erwünschten Erkenntnis, dass Energie und Impuls erhalten bleiben. Vgl. [MC73.1, S. 475].

Lösen der Feldgleichungen Die Feldgleichungen sind Differentialgleichungen zweiter Ordnung. Sie müssen gleichzeitig nach dem metrischen Tensor und dem Energie-Impuls-Tensor aufgelöst werden, weil die beiden Grössen eng miteinander verbunden sind. Wegen ihrer Komplexität sind die Feldgleichungen nur in wenigen Fällen mit hoher Symmetrie analytisch lösbar (vgl. [WR84.1, S. 73]).

Kapitel 2

Schwarze Löcher

In diesem Kapitel diskutieren wir zwei Lösungen der Feldgleichungen, die *Schwarzschild-Metrik* und die *Kerr-Newman-Metrik*, und verwenden sie, um schwarze Löcher zu beschreiben.

2.1 Schwarzschild-Metrik

Die *Schwarzschild-Metrik* ist eine der wenigen algebraischen Lösungen der Feldgleichungen. Durch einen hohen Grad an Symmetrie ist sie relativ einfach herzuleiten. Sie beschreibt die Raumzeit um *sphärisch symmetrische* und *statische* Massenverteilungen und kann auf Planeten und Sternen angewandt werden, da diese häufig hinreichend symmetrisch und statisch sind.

Vakuum- und innere Metrik Die Schwarzschildmetrik wurde 1916 sowohl für das Vakuum ausserhalb des Körpers [SK16.1] als auch für die homogene Masse innerhalb des Körpers [SK16.2] von Karl Schwarzschild hergeleitet. Wir sind jedoch nur an der Vakuum-Metrik interessiert; wir bezeichnen mit *Schwarzschild-Metrik* von nun an die *Vakuum-Metrik*.

Koordinaten Die Schwarzschild-Metrik ist in sogenannten *Schwarzschildkoordinaten* definiert. Die Koordinaten eines Ereignis $\mathscr{P}$ sind

$$x^\alpha(\mathscr{P}) = (t, r, \theta, \phi). \tag{2.1.1}$$

θ und ϕ sind Winkelkoordinaten: θ ist der Zenitwinkel und ϕ der Azimut. Die Polachse des Koordinatensystems kann frei gewählt werden. t ist die Eigenzeit eines unendlich weit entfernten Beobachters. r ist die *radiale Schwarzschildkoordinate*; sie misst *nicht* die Distanz vom Zentrum der Masse, sondern ist folgendermassen definiert:

$$r = (\text{[Fläche der Kugel um } r = 0 \text{ durch } \mathscr{P}]/4\pi)^{1/2}. \tag{2.1.2}$$

Vgl. [MC73.1, S. 596].

Metrik Die Schwarzschildmetrik hängt von der Masse M des zentralen Kör-

pers und der radialen Koordinate r ab. Sie ist gegeben durch

$$ds^2 = -\left(1 - \frac{2M}{r}\right) dt^2 + \left(1 - \frac{2M}{r}\right)^{-1} dr^2 + r^2 \left(d\theta^2 + \sin^2\theta \; d\phi^2\right). \quad (2.1.3)$$

Vgl. [MC73.1, S. 820].

Ereignishorizont Bei $r = 2M$ wird g_{tt} null und g_{rr} unendlich, vorausgesetzt, die Oberfläche der zentralen Masse befindet sich bei $r < 2M$. Es kann gezeigt werden, dass dies keine physikalische Singularität ist, da keine Komponente des Riemann-Tensors bei $r = 2M$ unendlich wird; $r = 2M$ ist eine *Koordinatensingularität*. Vgl. [WR84.1, S. 124].

$r = 2M$ ist der *Schwarzschildradius*; die Erde hat zum Beispiel einen Schwarzschildradius von $(2\ G \cdot 5.97 \cdot 10^{24}\ \text{kg})/c^2 \approx 9\text{mm}$. Ein fallendes Objekt benötigt von $r > 2M$ eine unendliche Koordinatenzeit t, um den Schwarzschildradius zu erreichen, aber nur eine endliche Eigenzeit.

Die Region innerhalb des Schwarzschildradius wird *schwarzes Loch* genannt. Diese Raumzeitregion ist durch den *Ereignishorizont* bei $r = 2M$ vom Rest des Universums abgeschnitten. Alle zeitartigen und lichtartigen Geodäten, welche $r = 2M$ unterschreiten, werden nie nach $r > 2M$ entweichen können und bei $r = 0$ enden. Dies bedeutet, dass Objekte, deren Radius kleiner als ihr Schwarzschildradius ist, unaufhaltsam in einem *Gravitationskollaps* zu einem Punkt unendlicher Dichte bei $r = 0$ zusammenstürzen. Weil kein Licht aus schwarzen Löchern entkommen kann, sind sie komplett schwarz. Vgl. [WR84.1, S. 154-155].

Photonensphäre Bei $r = 3M$ existieren labile Orbits von Photonen. Die Kugel $r = 3M$ wird deshalb *Photonensphäre* genannt. Photonen im Orbit entweichen mit einem kleinen Stoss nach aussen in die Unendlichkeit oder fallen mit einem kleinen Stoss nach innen ins schwarze Loch. Vgl. [WR84.1, S. 143].

Singularität Nach einem Gravitationskollaps ist die gesamte Masse des Körpers bei $r = 0$ in einem Punkt unendlicher Dichte konzentriert [1]. $r = 0$ ist gleichzeitig eine Koordinatensingularität und eine *physikalische Singularität*, da dort Komponenten des Riemann-Tensors unendlich werden. Vgl. [WR84.1, S. 124].

Schwarzschild-Löcher Die Schwarzschild-Metrik beschreibt die Raumzeit $r > 2M$ um ein schwarzes Loch mit Masse M. Solche schwarze Löcher heissen *Schwarzschild-Löcher*. Die Software `cuRRay` kann Bilder von Schwarzschild-Löchern erstellen, wenn der Drehimpuls und die elektrische Ladung auf null gesetzt werden. Schwarzschild-Löcher sind für die Visualisierung von vielen Lichtablenkungsphänomenen geeignet.

[1] Es wird angenommen, dass Quantengravitationseffekte bei sehr hohen Dichten eine wichtige Rolle spielen. Diese sind jedoch bisher unbekannt.

2.2 Kerr-Newman-Metrik

Die *Kerr-Newman-Metrik* beschreibt die axialsymmetrische Raumzeit einer Masse M mit Drehimpuls L und elektrischer Ladung Q. Sie wurde 1965 von Ezra T. Newman [NE65.1] gefunden und ist eine Verallgemeinerung der *Reissner-Nordström-Metrik* [RH16.1] (1916), [NG18.1] (1918) und der *Kerr-Metrik* [KR63.1] (1963), welche wiederum Verallgemeinerungen der Schwarzschild-Metrik sind.

Keine-Haare-Theorem Ähnlich wie in der Schwarzschild-Metrik existieren in der Kerr-Newman-Metrik schwarze Löcher, wir nennen sie *Kerr-Newman-Löcher*. Das *Keine-Haare-Theorem* „Black holes have no hair“ [2] (engl. „schwarze Löcher haben keine Haare“) ist eine Metapher dafür, dass schwarze Löcher maximal drei Eigenschaften besitzen: die Masse M, den Drehimpuls L und die elektrische Ladung Q. Ist das Theorem wahr, können alle Arten von stationären schwarzen Löchern durch die Kerr-Newman-Metrik beschrieben werden. Aus diesem Grund wurde die Kerr-Newman-Metrik für die Verwendung in `cuRRay` ausgewählt.

Koordinaten Wir verwenden die Kerr-Newman-Metrik in *Boyer-Lindquist-Koordinaten* (kurz: BL-Koordinaten). BL-Koordinaten haben den Vorteil, dass die Killing-Vektorfelder den Koordinatenvektorfelder entsprechen (Siehe Kapitel 2.3). t und ϕ haben die gleiche Bedeutung wie in der Schwarzschild-Metrik. Die Drehrichtung des schwarzen Lochs ist in Richtung zunehmender ϕ-Koordinate. Die Kerr-Newman-Metrik ist zwar nicht statisch, da die zentrale Masse rotiert, aber *stationär*, das heisst, dass sich die Komponenten des metrischen Tensors mit der Koordinatenzeit t nicht verändern (vgl. [WR84.1, S. 119]).

Diese Symmetrie ermöglicht uns, für Geodäten in Kerr-Newman-Raumzeit den Parameter λ beliebig zu skalieren. Beim Raytracing kehrt `cuRRay` die Zeit um, da die Geodäten rückwärts berechnet werden. Das schwarze Loch rotiert nun in die entgegengesetzte Richtung (vgl. [HS73.1, S. 161]). Aus diesem Grund muss bei der Interpretation von Bildern mit $a \neq 0$ vorsichtig vorgegangen werden. Wir werden in Kapitel 7.2 diese Problematik genauer untersuchen.

r und θ sind eine Art elliptische Koordinaten. Wir werden sie weiter unten genauer definieren.

Metrik Die Kerr-Newman-Metrik ist

$$ds^2 = -\left(\frac{\Delta - a^2 \ \sin^2\theta}{\Sigma}\right) dt^2 - \frac{2a \ \sin^2\theta(r^2 + a^2 - \Delta)}{\Sigma} dt d\phi + \left(\frac{(r^2 + a^2)^2 - \Delta a^2 \ \sin^2\theta}{\Sigma}\right) \sin^2\theta \ d\phi^2 + \frac{\Sigma}{\Delta} dr^2 + \Sigma d\theta^2, \quad (2.2.1)$$

$$\Sigma = r^2 + a^2 \ \cos^2\theta, \quad (2.2.2)$$

$$\Delta = r^2 + a^2 + Q^2 - 2Mr, \quad (2.2.3)$$

[2] John A. Wheeler in [MC73.1, S. 876]

$$a = L/M. \tag{2.2.4}$$

Die Metrik reduziert sich zur Schwarzschild-Metrik, wenn $a = Q = 0$, zur Kerr-Metrik, wenn $Q = 0, a \neq 0$ und zur Reissner-Nordström-Metrik, wenn $a = 0, Q \neq 0$. Vgl. [WR84.1, S. 313-314].

Lense-Thirring-Effekt In der Nähe des rotierenden schwarzen Lochs wird die Raumzeit durch den Kreuzterm $dtd\phi$ mitgerissen (vgl. [MC73.1, S. 879]). Dieses Phänomen wird *Lense-Thirring-Effekt* genannt. Geodäten von Teilchen werden dadurch beeinflusst.

Ereignishorizonte Die Kerr-Newman-Metrik besitzt allgemein *zwei* Ereignishorizonte an den Positionen $r_\pm$. Sie ergeben sich aus der Gleichung $\Delta = 0$:

$$r_\pm = M \pm \sqrt{M^2 - a^2 - Q^2}. \tag{2.2.5}$$

Sie heissen *äusserer Ereignishorizont* (r_+) und *innerer Ereignishorizont* (r_-) (vgl. [WR84.1, S. 315]). Falls die Wurzel in (2.2.5) null ist, ist die Metrik *extrem* und beide Ereignishorizonte fallen zusammen: $r_+ = r_- = M$. Hat Gleichung (2.2.5) keine Lösung, weil $M^2 \leq a^2 + Q^2$, existiert *kein* Ereignishorizont und die Singularität ist *nackt*. Wegen der Vermutung der *kosmischen Zensur* wird jedoch angenommen, dass aus plausiblen Anfangsbedingungen keine nackten Singularitäten entstehen können (siehe [WR84.1, S. 299-308]).

Ergosphären Eine weitere interessante Eigenschaft ist der Wechsel des Vorzeichens von g_{00}, das heisst $(a^2 \ \sin^2\theta - \Delta)/\Sigma = 0$, bei $r_{E\pm}$:

$$r_{E\pm} = M \pm \sqrt{M^2 - Q^2 - a^2 \ \cos^2\theta}. \tag{2.2.6}$$

r_{E+} wird *äussere statische Grenze* genannt, weil für $r_{E+} > r > r_+$ jeder Beobachter immer in Richtung der Rotation des schwarzen Lochs mitgerissen wird; dort existieren keine statischen Beobachter. Der Bereich zwischen r_+ und r_{E+} heisst *äussere Ergosphäre*. Je näher der Bobachter an r_+ gelangt, desto schneller muss sein Orbit sein, um bei $r =$ konst. zu bleiben. Für $r_- > r > r_+$ ist es für einen Beobachter unmöglich, r konstant zu halten. Gleichermassen heisst r_{E-} *innere statische Grenze* und der Bereich zwischen r_- und r_{E-} *innere Ergosphäre*. Vgl. [MC73.1, S. 894].

Orbits von Photonen Auch in Kerr-Newmann-Raumzeit existieren labile Orbits von Photonen (vgl. [CC13.1]). Wir betrachten diese jedoch nicht im Detail.

Singularität Bei

$$r^2 + a^2 \ \cos^2\theta = 0 \tag{2.2.7}$$

existiert eine physikalische Singularität (siehe [WR84.1, S. 314]). Interpretierten wir r und θ als sphärische Koordinaten, würde die Singularität nur für $r = 0$ und $\theta = \pi/2$ existieren, nicht aber für alle anderen Werte von θ, ein Widerspruch, da ein Ereignis in der Raumzeit „aus jeder Richtung“ den gleichen metrischen Tensor haben muss. Betrachten wir r und θ als Boyer-Lindquist-

Koordinaten, wird klar, dass die Singularität ein *Ring* um die Drehachse des schwarzen Lochs ist. Vgl. [WR84.1, S. 314-315].

Kartesische Koordinaten Die Kerr-Newman-Metrik ist *asymptotisch flach* (vgl. [HS73.1, S. 161]), das bedeutet, dass die Raumzeit immer flacher wird, je weiter entfernt sich ein Beobachter vom schwarzen Loch befindet. Wir können die Raumzeit mit einem kartesischen Koordinatensystem überziehen, welches zwar in der Nähe des Ereignishorizonts ungenau wird, aber weiter entfernt wegen der asymptotischen Fläche tatsächliche Längen und Winkel in der Raumzeit widerspiegelt:

$$\begin{aligned} x &= \sqrt{r^2 + a^2} \sin\theta \cos\phi, \\ y &= \sqrt{r^2 + a^2} \sin\theta \sin\phi, \\ z &= r \cos\theta \end{aligned} \tag{2.2.8}$$

(vgl. [VM08.1, S. 15]). In kartesischen Koordinaten ist die Ring-Singularität um den Koordinatenursprung zentriert und hat Radius a (vgl. [HS73.1, S. 162-163] und [WR84.1, S. 314-315]).

Berechnungen in kartesischen Koordinaten Wir werden in `cuRRay` kartesische Koordinaten verwenden, um Kollisionen von Lichtstrahlen mit Kugeln mittels Vektorgeometrie zu berechnen. Dies vereinfacht einerseits die Berechnungen, da sie im gewohnten dreidimensionalen, Euklidischen Raum durchgeführt werden; andererseits werden die Berechnungen verfälscht, da wie oben besprochen, die kartesischen Koordinaten nur unendlich weit vom schwarzen Loch die Struktur der Raumzeit genau beschreiben.

Wir nehmen die Ungenauigkeit in Kauf, da die genaue Konstruktion von Kugeln in gekrümmter Raumzeit den Rahmen dieser Arbeit sprengen würde. Obwohl die Abweichungen in den meisten Fällen (r-Position der Kugeln $> 10M$) klein sein werden, sollten Bilder vorsichtig interpretiert werden.

Nur die Kollisionserkennung zwischen Lichtstrahlen und Kugeln ist von der Ungenauigkeit betroffen; die Lichtstrahlen selber sowie deren Kollisionen mit der Akkretionsscheibe werden in BL-Koordinaten berechnet.

Visualisierung Wir können kartesische Koordinaten ebenfalls zur Visualisierung der Kerr-Newman-Metrik verwenden. Abbildung 2.2.1 zeigt einen Schnitt ($t = \phi = \text{konst}$) durch die Kerr-Newman-Metrik mit $M = 0.9$, $Q = 0$ und $a = 0.85$ eines schwarzen Lochs, welches sich um die z-Achse dreht. Abbildung 2.2.2 zeigt einen ähnlichen Schnitt für den Extremfall $M = a = 0.9$ und $Q = 0$. In beiden Abbildungen sind die grauen Linien Kurven für $r = \text{konst}$ und $\theta = \text{konst}$. An den Rändern sind zusätzlich die kartesischen Koordinaten notiert. Die beiden schwarzen Punkte zeigen den Schnitt durch die unendlich dünne Ringsingularität. Die roten Kurven markieren die Ereignishorizonte, die schwarzen die statischen Grenzen.

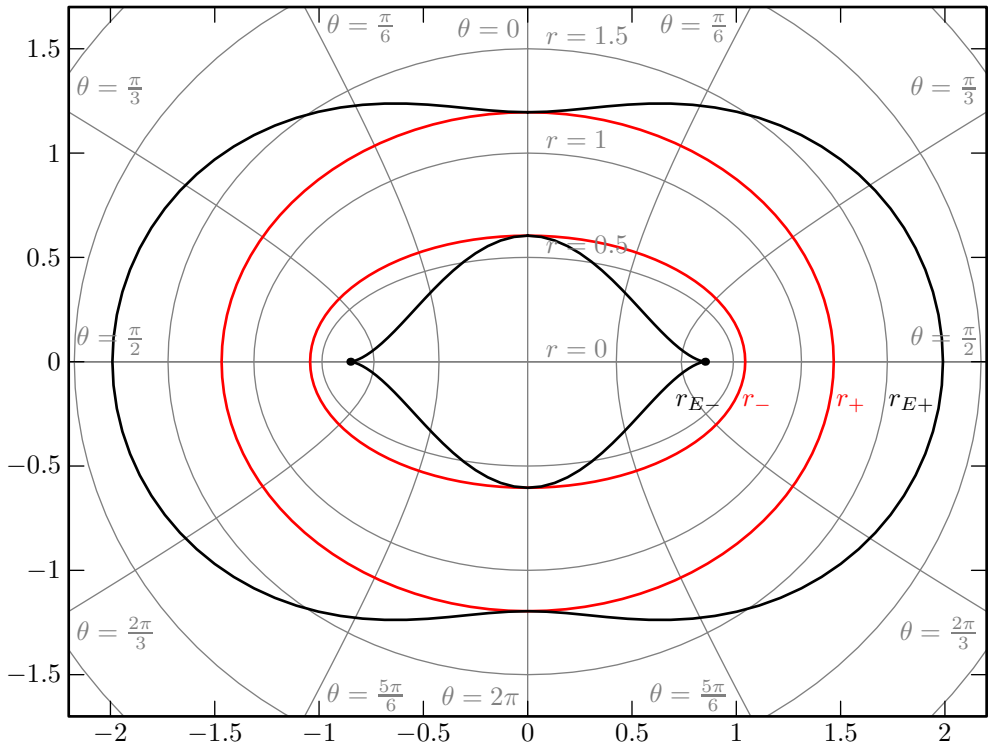


Abbildung 2.2.1: Kerr-Newman-Metrik in kartesischen Koordinaten für $M = 0.9$, $Q = 0$ und $a = 0.85$. Die Ellipse $r = 0$ wird zu einer Linie zwischen zwei Punkten auf der ringförmigen Singularität (dargestellt durch schwarze Punkte).

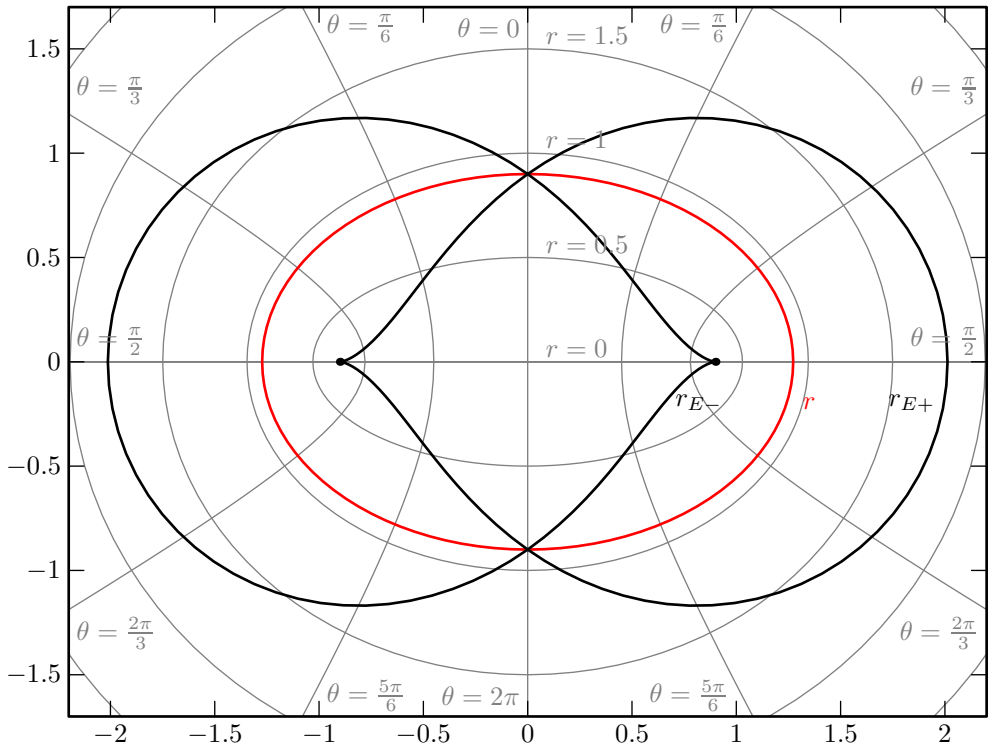


Abbildung 2.2.2: Extreme Kerr-Metrik in kartesischen Koordinaten für $M = 0.9$, $Q = 0$ und $a = 0.9$. Die Koordinatenlinien sind die gleichen wie in Abbildung 2.2.1. Da die Metrik extrem ist, existiert nur ein Ereignishorizont, es sind jedoch nach wie vor zwei statische Grenzen vorhanden.

2.3 Killing-Vektorfelder und Erhaltungsgrössen

Killing-Vektorfelder Wenn wir jeden Punkt $\mathscr{P}$ in der Raumzeit um $\boldsymbol{dx}(\mathscr{P})$ infinitesimal verschieben können, ohne dass die Metrik verändert wird, existiert ein *Killing-Vektorfeld* $\boldsymbol{\xi}$, welches *Generator* der infinitesimalen Verschiebungen ist; es gilt dann:

$$\boldsymbol{\xi} = \frac{\partial}{\partial \boldsymbol{x}}. \tag{2.3.1}$$

Jedes Vektorfeld, welches die *Killing-Gleichung*

$$\xi_{\alpha;\beta} + \xi_{\beta;\alpha} = 0 \tag{2.3.2}$$

erfüllt, ist ein Killing-Vektorfeld. Killing-Vektorfelder verkörpern die Symmetrien der Raumzeit. Vgl. [MC73.1, S. 652].

Bewegungskonstanten Das Skalarprodukt zwischen dem Tangentenvektor $\boldsymbol{u}$ und dem Killing-Vektor entlang einer beliebigen Geodäte bleibt konstant; dieses Produkt wird *Bewegungskonstante* genannt. Da die Wahl der Geodäte frei ist, erhalten wir überall

$$\boldsymbol{u} \cdot \boldsymbol{\xi} = \text{konst.} \tag{2.3.3}$$

Wenn das Killing-Vektorfeld einem Koordinatenvektorfeld ∂_K entspricht, gilt zudem

$$\xi^\alpha = \delta^\alpha{}_K. \tag{2.3.4}$$

Vgl. [MC73.1, S. 651].

In einem passenden Koordinatensystem können die Geodätengleichungen mithilfe von Bewegungskonstanten vereinfacht werden. Wir werden uns dies in `cuRRay` zu Nutzen machen.

2.3.1 Killing-Vektorfelder der Kerr-Newman-Metrik [3]

Die Kerr-Newman-Metrik besitzt zwei Killing-Vektorfelder, welche der Zeittranslationssymmetrie und der Rotationssymmetrie entsprechen.

Energie Das Killing-Vektorfeld $t^\alpha = (1, 0, 0, 0)$ in BL-Koordinaten liefert als Erhaltungsgrösse

$$E = g_{tt} u^t + g_{t\phi} u^\phi. \tag{2.3.5}$$

E ist proportional zur Energie des Teilchens, die ein unendlich weit entfernter Beobachter messen würde.

Drehimpuls Das Killing-Vektorfeld $\phi^\alpha = (0, 0, 0, 1)$ in BL-Koordinaten liefert als Erhaltungsgrösse

$$L = g_{\phi\phi} u^\phi + g_{t\phi} u^t. \tag{2.3.6}$$

L ist proportional zum Drehimpuls des Teilchens, der ein unendlich weit entfernter Beobachter messen würde.

[3] Dieses Kapitel basiert auf [WR84.1, S. 313, 320]

2.3.2 Geodätengleichungen der Kerr-Newman-Metrik

In diesem Kapitel definieren wir die Geodätengleichungen in Kerr-Newman-Raumzeit, welche von `cuRRay` integriert werden. Wir beginnen mit den Geodätengleichungen in der Form von (1.7.10):

$$\frac{d^2x^\alpha}{d\lambda^2} + \Gamma^\alpha{}_{\beta\gamma}\frac{dx^\beta}{d\lambda}\frac{dx^\gamma}{d\lambda} = 0. \tag{2.3.7}$$

Die Geodätengleichungen sind ein *gekoppeltes System gewöhnlicher, partieller Differentialgleichungen zweiter Ordnung.*

Gleichungen erster Ordnung Die Killing-Vektorfelder t^α und ϕ^α ermöglichen uns, die Bewegungsgleichungen der t- und ϕ-Komponenten der Position zu vereinfachen. Wir erhalten für t

$$\frac{dt}{d\lambda} = \frac{E \cdot g_{\phi\phi} - L \cdot g_{t\phi}}{g_{tt}g_{\phi\phi} - {g_{t\phi}}^2} \tag{2.3.8}$$

und für ϕ

$$\frac{d\phi}{d\lambda} = \frac{L \cdot g_{tt} - E \cdot g_{t\phi}}{g_{tt}g_{\phi\phi} - {g_{t\phi}}^2}. \tag{2.3.9}$$

Alleine E, L und die Anfangsposition x_0^α bestimmen die Bewegungsgleichungen der t- und ϕ-Komponenten. Die Geodätengleichungen (2.3.8) und (2.3.9) werden in Anhang A hergeleitet.

Gleichungen zweiter Ordnung Die Bewegungsgleichungen für r und θ bleiben zweier Ordnung. Wir können jedoch alle Γ-Terme, welche null sind, wegstreichen. Für r erhalten wir:

$$\begin{aligned}\frac{d^2r}{d\lambda^2} =& -\Gamma^r{}_{tt}(u^t)^2 - \Gamma^r{}_{rr}(u^r)^2 - \Gamma^r{}_{\theta\theta}(u^\theta)^2 - \Gamma^r{}_{\phi\phi}(u^\phi)^2 \\ & - 2\Gamma^r{}_{t\phi}u^t u^\phi - 2\Gamma^r{}_{r\theta}u^r u^\theta,\end{aligned} \tag{2.3.10}$$

und für θ

$$\begin{aligned}\frac{d^2\theta}{d\lambda^2} =& -\Gamma^\theta{}_{tt}(u^t)^2 - \Gamma^\theta{}_{rr}(u^r)^2 - \Gamma^\theta{}_{\theta\theta}(u^\theta)^2 - \Gamma^\theta{}_{\phi\phi}(u^\phi)^2 \\ & - 2\Gamma^\theta{}_{t\phi}u^t u^\phi - 2\Gamma^\theta{}_{r\theta}u^r u^\theta.\end{aligned} \tag{2.3.11}$$

Wobei wie oben $u^\alpha = dx^\alpha/d\lambda$. Eine Liste mit den Ausdrücken für die Christoffelsymbole der Gleichungen (2.3.10) und (2.3.11) wird im Anhang B gegeben.

Kapitel 3

Licht

In diesem Kapitel betrachten wir elektromagnetische Strahlung und insbesondere Photonen in gekrümmter Raumzeit.

3.1 Elektromagnetisches Feld

Das *elektromagnetische Feld* beeinflusst elektrisch geladene Teilchen und wird durch sie erzeugt. Es ist für die *elektromagnetische Kraft*, eine der vier Fundamentalkräfte der Physik, zuständig. Das elektromagnetische Feld lässt Wellen zu, welche sich mit Lichtgeschwindigkeit ausbreiten: Licht.

Farady-Tensor Der Rang-(1,1) *Farady-Tensor* $\boldsymbol{F}$ beschreibt das elektromagnetische Feld. In einem Inertialsystem können seine Komponenten aus dem elektrischen Vektorfeld $\boldsymbol{E}$ und dem magnetischen Vektorfeld $\boldsymbol{B}$ gewonnen werden:

$$F^{\alpha}{}_{\beta} = \begin{matrix} \\ \alpha \downarrow \end{matrix} \begin{matrix} \qquad\quad \beta \rightarrow \\ \end{matrix} \begin{pmatrix} 0 & \boldsymbol{E}_x & \boldsymbol{E}_y & \boldsymbol{E}_z \\ \boldsymbol{E}_x & 0 & \boldsymbol{B}_z & -\boldsymbol{B}_y \\ \boldsymbol{E}_y & -\boldsymbol{B}_z & 0 & \boldsymbol{B}_x \\ \boldsymbol{E}_z & \boldsymbol{B}_y & -\boldsymbol{B}_x & 0 \end{pmatrix}. \tag{3.1.1}$$

Vgl. [MC73.1, S. 71-73].

Lorentz-Kraft Die Kraft, welche vom elektromagnetischen Feld auf ein Teilchen der Ladung q ausgeübt wird, ist die *Lorentz-Kraft*:

$$\frac{d\boldsymbol{p}}{d\tau} = q\boldsymbol{F}(\boldsymbol{u}) = qF^{\alpha}{}_{\beta}\, u^{\beta}, \tag{3.1.2}$$

wobei $\boldsymbol{u}$ die Geschwindigkeit und τ die Eigenzeit des Teilchens ist. Vgl. [MC73.1, S. 73].

Maxwellgleichungen Die *Maxwellgleichungen* nehmen, ausgedrückt durch den Farady-Tensor, folgende Form an:

$$F_{\alpha\beta;\gamma} + F_{\beta\gamma;\alpha} + F_{\gamma\alpha;\beta} = 0, \tag{3.1.3}$$

$$F^{\alpha\beta}{}_{;\beta} = 4\pi j^{\alpha}. \tag{3.1.4}$$

Vgl. [MC73.1, S. 568].

Wellengleichung Das elektromagnetische Feld wird durch ein *Vektorpotential* $\boldsymbol{A}$ erzeugt:

$$F_{\alpha\beta} = A_{\beta;\alpha} - A_{\alpha;\beta}. \tag{3.1.5}$$

Gleichung (3.1.4) kann mit $\boldsymbol{A}$ umgeschrieben werden, so dass wir die *Wellengleichung* [1] erhalten:

$$-A^{\alpha;\beta}{}_{\beta} + R^{\alpha}{}_{\beta}A^{\beta} = 4\pi j^{\alpha}. \tag{3.1.6}$$

Die Wellengleichung (3.1.6) lässt Potentiale zu, welche sich periodisch ändern und somit eine Welle bilden. Der Term $R^{\alpha}{}_{\beta}A^{\beta}$ trägt der Raumzeitkrümmung Rechnung. Er erschwert das Finden einer Lösung der Wellengleichung. Vgl. [MC73.1, S. 569].

3.2 Elektromagnetische Wellen

Mögliche Lösungen der Wellengleichung (3.1.6) sind elektromagnetische Wellen, die sich mit Lichtgeschwindigkeit ausbreiten. Diese Wellen gleichen denen, welche aus der Elektrodynamik in flacher Raumzeit bekannt sind, mit dem Unterschied, dass in gekrümmter Raumzeit der Ricci-Term der Wellengleichung zu Abweichungen führt. Wählen wir jedoch die Wellenlänge im Vergleich zur Raumzeitkrümmung klein genug, kann die Abweichung vernachlässigt werden. Siehe [MC73.1, S. 571].

Unter der Annahme einer genügend kleinen Wellenlänge lassen sich folgende Schlüsse ziehen [2]:

Wellenvektor Für einen Beobachter mit Geschwindigkeit $\boldsymbol{u}$ ist die *Kreisfrequenz* ω der Welle

$$\omega = -\varphi_{;\alpha}u^{\alpha} = -k_{\alpha}u^{\alpha}. \tag{3.2.1}$$

$\boldsymbol{k}$ heisst *Wellenvektor*; er zeigt in die Ausbreitungsrichtung der Welle.

Null-Geodäten Für den Wellenvektor gilt insbesondere

$$\boldsymbol{k} \cdot \boldsymbol{k} = 0, \tag{3.2.2}$$

$$\nabla_{\boldsymbol{k}}\boldsymbol{k} = 0. \tag{3.2.3}$$

$\boldsymbol{k}$ ist ein *Null-Vektor* und wird entlang sich selber parallel transportiert. Da $\boldsymbol{k}$ mit der Ausbreitungsgeschwindigkeit $\boldsymbol{u}$ der Welle kollinear ist, gilt ebenfalls

$$\boldsymbol{u} \cdot \boldsymbol{u} = 0, \tag{3.2.4}$$

$$\nabla_{\boldsymbol{u}}\boldsymbol{u} = 0. \tag{3.2.5}$$

Elektromagnetische Strahlung breitet sich demnach entlang von Null-Geodäten aus. Wir nennen die Geodäten von elektromagnetischen Wellen *Lichtstrahlen*.

[1] Gleichzeitig wird die Lorenz-Eichung $A^{\alpha}{}_{;\alpha} = 0$ angewandt.

[2] Für eine vollständige Herleitung, siehe [MC73.1, Kapitel 22.5]

3.3 Photonen [3]

Da sich Lichtstrahlen entlang von Geodäten bewegen, ist es möglich, Licht durch Teilchen zu beschreiben, welche sich auf den gleichen Geodäten bewegen. Diese Lichtteilchen heissen *Photonen*. `cuRRay` löst die Geodätengleichungen von einzelnen Photonen.

Ruhemasse Photonen haben keine Ruhemasse, weil sie sich auf Null-Geodäten bewegen.

Impuls Der Impuls $\boldsymbol{p}$ eines Photons ist gegeben durch

$$\boldsymbol{p} = \hbar \boldsymbol{k}. \tag{3.3.1}$$

$\boldsymbol{p}$ zeigt in die gleiche Richtung wie die Geschwindigkeit $\boldsymbol{u}$.

Energie Die Energie eines Photons ist

$$E = \hbar\omega. \tag{3.3.2}$$

3.4 Lichtablenkung

Abbildung 3.4.1 zeigt schematisch, wie die Lichtstrahlen γ_1 und γ_2 einer punktförmigen Quelle Q von einem Schwarzen Loch L um die Winkel α_1 und α_2 abgelenkt werden. Für den Beobachter B entstehen in der Ebene zwei Bilder (I_1 und I_2) der Quelle; in drei Dimensionen entsteht ein Ring, wenn die Quelle genau hinter dem Schwarzen Loch ist.

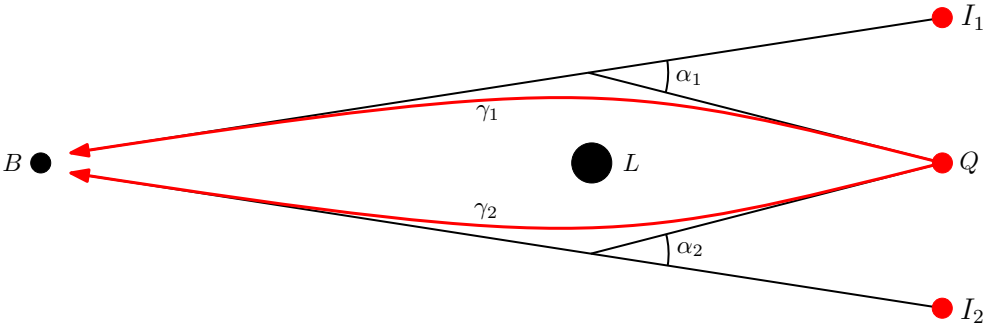


Abbildung 3.4.1: Das schwarze Loch L lenkt die Lichtstrahlen γ_1 und γ_2 der Quelle Q ab, so dass der Bobachter B die Bilder I_1 und I_2 sieht.

Weil die Lichtstrahlen unterschiedlicher Teile einer ausgedehnten Quelle im Allgemeinen unterschiedlich stark oder in unterschiedliche Richtungen abgelenkt werden, ist das Gesamtbild der Quelle häufig verzerrt.

Diese Phänomene erwarten wir in den von `cuRRay` erstellten Bildern.

[3] Dieses Kapitel basiert auf [SP99.1, S. 98-100].

3.5 Gravitative Rotverschiebung

Lichtsignale, welche zwischen zwei Beobachtern B_S (Sender) und B_E (Empfänger) ausgetauscht werden, unterliegen der *gravitativen Rotverschiebung*. Unterschiede in der Metrik an den Positionen von B_S und B_E führen zu einer Veränderung der Periode T der elektromagnetischen Wellen und somit einer Verschiebung der Kreisfrequenz ω. Je nachdem, ob die Wellenlänge zunimmt oder abnimmt, sprechen wir von gravitativer *Rot-* oder *Blauverschiebung*.

Verhältnis der Frequenzen In cuRRay sind wir am Verhältnis der Kreisfrequenzen ω_E/ω_S interessiert. Aus Gleichung (2.2.1) und Gleichung (3.2.1) erhalten wir in Kerr-Newman-Raumzeit

$$\frac{\omega_E}{\omega_S} = \frac{u^t\sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\Big|_E}{u^t\sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\Big|_S}. \tag{3.5.1}$$

Siehe Anhang C für die Herleitung von Gleichung (3.5.1).

`cuRRay` berechnet $\boldsymbol{u}$ rückwärts entlang der Geodäte und benutzt den Startwert $\boldsymbol{u}_E$; am Ende der Geodäte ist $\boldsymbol{u}_S$ bekannt. Die Software verwendet Gleichung (3.5.1), um die gravitative Rotverschiebung zu berechnen.

Gleichung (3.5.1) gilt nur für stationäre Quellen (Sender). Das bedeutet, dass sie für Quellen innerhalb der Ergosphäre nicht angewandt werden kann. Da wir uns nur für die gravitative Rotverschiebung interessieren, ist dies auch nicht notwendig; Quellen innerhalb der Ergosphäre können nie stationär sein und sind deshalb nie nur von gravitativer Rotverschiebung, sondern auch vom relativistischen Dopplereffekt betroffen. `cuRRay` berechnet Rotverschiebung nur für Quellen ausserhalb der Ergosphäre.

Kapitel 4

Übersicht `cuRRay`

In diesem Kapitel soll eine Übersicht der Software `cuRRay` (CUDA relativistic raytracer) gegeben werden. `cuRRay` ist Hauptgegenstand dieser Arbeit und wurde von Grund auf programmiert.

4.1 Funktionen

`cuRRay` kann Bilder, sogenannte *Frames*, von Objekten in Kerr-Newman-Raumzeit generieren, wie zum Beispiel das Bild der *Akkretionsscheibe* (siehe weiter unten) um ein schwarzes Loch.

Es wurden zwei Versionen von `cuRRay` programmiert: Die CUDA-Version und die CPU-Version. Die CUDA-Version benötigt eine unterstützte NVIDIA-Grafikkarte, verwendet CUDA für die Berechnungen und ist schneller als die CPU-Version. Die CPU-Version benötigt keine Grafikkarte und verwendet die CPU für Berechnungen.

Raytracing Raytracing (deutsch: *Strahlenverfolgung*) bezeichnet das Zeichnen eines Bildes, indem einzelne Lichtstrahlen simuliert werden. `cuRRay` verfolgt für jeden Pixel des zu erstellenden Bildes den dazugehörigen Lichtstrahl rückwärts durch die Kerr-Newman-Raumzeit, bis ein Objekt getroffen wird, der Strahl in die Unendlichkeit entwischt oder in den Ereignishorizont fällt.

Szene Die Software liest Informationen über die Szene aus einer YAML-Datei [1], der Szenendatei. Die Parameter der Metrik und die Position, die Orientierung und das Sichtfeld des Beobachters werden darin aufgeführt. Ausserdem können Kugeln in der Raumzeit platziert, eine Akkretionsscheibe konfiguriert und ein Sternenhimmel eingestellt werden. Schliesslich ermöglicht die Szenendatei die Wahl verschiedener Farben für spezielle Pixel: Die Farbe des Ereignishorizonts, die Farbe des Himmels, wenn kein Sternenhimmel konfiguriert wird, und eine Farbe für fehlerhafte Pixel.

Kugeln `cuRRay` erlaubt zwei Arten von Objekten, die in der Szenendatei kon-

[1] YAML ist eine gut lesbare Auszeichnungssprache (markup language), welche ebenfalls gut von Computern interpretiert werden kann. Siehe `http://yaml.org/`.

figuriert werden können; Kugeln sind eine davon. Kugeln erhalten ein Schachbrettmuster, welches in zwei Farben eingefärbt wird: Grau und eine wählbare Akzentfarbe. Es können bis zu acht Kugeln frei platziert werden. Der Radius der Kugeln ist ebenfalls einstellbar.

Akkretionsscheibe Die zweite Art von Objekt ist die *Akkretionsscheibe*. In Realität sind Akkretionsscheiben Scheiben um den Äquator des schwarzen Lochs, welche durch einfallendes Gas gebildet werden. In `cuRRay` kann eine Akkretionsscheibe in der Ebene $\theta = \pi/2$ modelliert werden. Um die Akkretionsscheibe zu beschreiben, wird der innere und äussere Radius in der Szenendatei angegeben. Ausserdem können zwei Akzentfarben gewählt werden, eine für die Oberseite, eine für die Unterseite. Wird nur eine Farbe gewählt, erhält die Unterseite die Komplementärfarbe der Oberseite. Ähnlich der Kugeln wird die Akkretionsscheibe mit einem Schachbrettmuster überzogen

Sternenhimmel In der Szenendatei kann eine Bilddatei angegeben werden, welche auf den Himmel der Szene projiziert wird. So ist es zum Beispiel möglich, einen Sternenhimmel im Hintergrund des schwarzen Lochs zu zeichnen.

Frame-Informationen Frame-Informationen werden über die Kommandozeile an `cuRRay` gesendet. Sie umfassen die Masse der resultierenden Frames in Pixeln, der Pfad der Szenendatei, das Ausgabeverzeichnis, die Art der Ausgabe und die Anzahl zu zeichnender Frames. Viele Parameter aus der Szenendatei lassen sich über mehrere Frames hinweg linear animieren, indem ein Anfangs- und Endwert angegeben wird.

Systemkonfigurationsdatei Ebenfalls kann in der Kommandozeile eine *Systemkonfigurationsdatei* (Sysconfig) angegeben werden. Wird keine explizit angegeben, verwendet `cuRRay` eine Standarddatei. Die Sysconfig-Datei ist ebenfalls eine YAML-Datei. Sie definiert, welche GPUs von der CUDA-Version und wie viele CPU-Threads von der CPU-Version verwendet werden. Ausserdem kann definiert werden, wie Information ins Logbuch und in die Konsole geschrieben wird. Einstellungen des Raytracers können ebenfalls konfiguriert werden. Einige der Optionen in der Sysconfig können in der Kommandozeile überschrieben werden.

Ausgabe `cuRRay` erzeugt je nach gewähltem Ausgabetyp verschiedene Ausgabedateien im Ausgabeverzeichnis. Erstens kann `cuRRay` eine Textdatei erstellen, in der die Szeneninformationen sowie die Frame-Informationen nochmals aufgeführt sind. Zweitens können pro Frame die Pixelfarben in einer PNG-Bilddatei gespeichert werden. Drittens kann `cuRRay` auch die Rotverschiebungen der Pixel in einer PNG-Datei speichern. Schliesslich kann pro Frame eine CSV-Datei mit den detaillierten Pixeldaten angelegt werden, welche später in einem Tabellenkalkulationsprogramm eingesehen werden kann. Alle Dateien, die pro Frame erstellt werden, nummeriert `cuRRay` mit der Frame-Nummer.

4.2 Systemvoraussetzungen

Plattform `cuRRay` wurde in C++ [2] geschrieben und verwendet nur Code-Bibliotheken, welche plattformübergreifend verfügbar sind. Aus diesem Grund kann `cuRRay` bei Bedarf auf den meisten 64-bit-Betriebssystemen kompiliert und verwendet werden. Siehe Anhang G für Informationen zur Kompilation von `cuRRay` unter Windows und Linux. Das Git-Repository (siehe Anhang F) enthält den Quellcode und, zum Zeitpunkt des Drucks, kompilierte Versionen für Windows 10 und Linux Debian 8.

Prozessor- und OS-Architektur Ein 64-bit-Prozessor sowie ein 64-bit Betriebssystem sind notwendig.

Arbeitsspeicher `cuRRay` benötigt ein Minimum von 1 GB Arbeitsspeicher. Je nach Grösse der zu zeichnenden Bilder wird mehr Arbeitsspeicher benötigt.

CUDA-Grafikkarte [3] Eine NVIDIA® Grafikkarte mit Unterstützung für CUDA 8.0 und *compute capability* 3.0 ist für die CUDA-Version von `cuRRay` notwendig. Weiter sollte der Grafikprozessor über mindestens 500 MB Grafikspeicher verfügen. [4]

Die CPU-Version von `cuRRay` benötigt keine Grafikkarte und verwendet die CPU des Systems.

Festplattenspeicher Der verwendete Festplattenspeicher ist vom Betriebssystem abhängig. Es wird ein minimaler Speicher von 1 GB empfohlen.

4.3 Verwendete Code-Bibliotheken

Folgende Code-Bibliotheken wurden verwendet. Die angegebenen Versionen der Bibliotheken sind die, welche bei der Entwicklung verwendet wurden. Es kann sein, dass ältere Versionen ebenfalls funktionieren.

Boost 1.61 [5] Boost ist eine umfassende Sammlung von Erweiterungen der C++-Standardbibliothek. Die Bibliothek ist frei verfügbar.

C++ 14 Standardbibliothek [6] Die Standardbibliothek, Version 14, ist mit C++ zusammen frei verfügbar.

libpng 1.6.26 [7] libpng ist die offizielle Bibliothek zur Handhabung von PNG-Bilddateien. Sie ist frei verfügbar.

NVIDIA® CUDA 8.0 runtime [8] Die CUDA 8.0 runtime wird benötigt,

[2] `isocpp.org/`

[3] Siehe Kapitel 5.2.2

[4] Für eine ausführliche Liste aller Grafikkarten, welche compute capability 3.0 unterstützen, siehe `https://developer.nvidia.com/cuda-gpus`.

[5] `boost.org`

[6] `cplusplus.com/reference/` und `cppreference.com/`

[7] `libpng.org`

[8] `developer.nvidia.com/cuda-toolkit`

um auf die Funktionen von CUDA-Geräten, wie zum Beispiel eine Grafikkarte, zuzugreifen. Die Bibliothek ist frei verfügbar, setzt aber mindestens ein NVIDIA® CUDA-Gerät voraus.

pngIO 1.1 [9] pngIO ist eine vom Autor entwickelte Bibliothek, welche die wichtigsten Funktionen von libPNG vereinfacht anbietet. Die Bibliothek ist frei verfügbar und bereits im Quellcode von `cuRRay` inbegriffen.

TCLAP 1.2.1 [10] Die Templatized C++ Command Line Parser Library (TCLAP) ermöglicht das Einlesen von Kommandozeileneingaben. Die Bibliothek ist frei verfügbar und bereits im Quellcode von `cuRRay` inbegriffen.

yaml-cpp 0.6.0 [11] yaml-cpp ermöglicht das Lesen und Schreiben von YAML-Dateien. Die Bibliothek ist frei verfügbar.

4.4 Kurzanleitung

Eine Anleitung zum Bedienen der Software wird in Anhang E gegeben.

[9] https://gitlab.com/misc-util/pngIO
[10] `http://tclap.sourceforge.net/`
[11] https://github.com/jbeder/yaml-cpp

Kapitel 5

Programmstruktur

In diesem Kapitel behandeln wir die Programmstruktur von `cuRRay`. Dazu gehören die Designentscheide, welche bezüglich Ein- und Ausgabe und paralleler Ausführung von Code getroffen wurden.

5.1 Ein- und Ausgabe

`cuRRay` muss in erster Linie die Funktionen, welche in Kapitel 4.1 aufgelistet sind, implementieren. Die Software soll ausserdem einfach zu bedienen sein, aber trotzdem viele Konfigurationsmöglichkeiten besitzen. Des Weiteren soll sie möglichst betriebssystemunabhängig sein. Folgende Entscheidungen wurden getroffen:

Benutzerinteraktion Der Benutzer interagiert über die Systemkonsole mit der Software; dies hat den Vorteil, dass keine grafische Oberfläche benötigt wird und `cuRRay` auf vielen Betriebssystemen kompiliert werden kann. `cuRRay` wird über Kommandozeilenargumente gesteuert. YAML-Dateien werden verwendet, um grosse Mengen an Informationen in die Software zu laden. Siehe *Systemkonfigurationsdatei* und *Szenendatei* in Kapitel 4.1.

Berichterstattung Die Software kopiert die Konsolenausgabe mitsamt Fehlermeldungen in eine Logdatei, welche nach der Ausführung eingesehen werden kann. Die Ausgabe in die Konsole und in die Logdatei kann, wie weiter oben besprochen, konfiguriert werden.

Dateiformate Die Software verwendet YAML (`.yml`) für Konfigurationsdateien, da dieses Dateiformat einfach von Hand bearbeitet werden kann und es das Speichern von strukturierter Information ermöglicht. Die Informationsdatei, welche beim Raytracing erstellt werden kann, ist eine ASCII-Textdatei (`.txt`). Bilder werden als PNG-Dateien (`.png`) abgespeichert. Genaue Pixeldaten werden in CSV-Dateien (`.csv`) gesichert.

5.2 Parallele Ausführung von Code

`cuRRay` verwendet zwei Arten von Prozessoren: *CPUs* und *GPUs*. Beide Prozessorarten ermöglichen eine parallele Ausführung von Code, welche wir in `cuRRay` verwenden. `cuRRay` verwendet NVIDIA® CUDA, um neben der CPU auf die Funktionen der GPU(s) des Systems zuzugreifen.

5.2.1 CPU [1]

CPU steht für *central processing unit* (zentrale Recheneinheit). Jeder Computer hat mindestens eine CPU. CPUs sind darauf spezialisiert, komplexe, heterogene Berechnungen durchzuführen.

Kerne Die meisten modernen CPUs besitzen mehrere *Kerne* (englisch: Cores) und werden daher *Multicore-CPUs* genannt. Die Kerne können Code unabhängig von den anderen ausführen. CPUs sind vor allem auf *Funktionsparallelität* ausgelegt, sprich Problemstellungen, in welchen verschiedene Arten von Aufgaben gleichzeitig ausgeführt werden müssen.

Threads In den meisten modernen Betriebssystemen schreibt der Programmierer nicht direkt Code für die einzelnen Kerne, sondern für sogenannte *Threads*. Das Betriebssystem kümmert sich um die parallele Ausführung der Threads: Sie werden häufig nur für kurze Zeit am Stück ausgeführt, dies erlaubt dem Betriebssystem, zwischen mehreren Threads zu wechseln und so mehr Threads auszuführen, als die Anzahl der Kerne zulässt. Die schnell abwechselnde serielle Ausführung der Threads, kombiniert mit der physikalischen Parallelität moderner CPUs, erweckt den Anschein, dass beliebig viele Threads parallel ausgeführt werden können.

Prozesse Threads gehören einem *Prozess* an. Meistens wird jedem ausgeführten Programm ein Prozess zugewiesen [2]. Jeder Prozess hat anfänglich einen einzelnen Thread, den *Hauptthread*, weitere können zur Laufzeit erzeugt werden. Abbildung 5.2.1 zeigt eine mögliche Kombination von Prozessen und Threads.

CUDA-Version Die CUDA-Version verwendet einen *Hauptthread*, welcher die Kommandozeile liest, Konfigurationsdateien lädt und Grafikkarten verwaltet. Der Hauptthread startet pro Grafikkarte einen weiteren Thread. Diese weiteren Threads sorgen dafür, dass mehrere CUDA-Grafikkarten parallel arbeiten können. Welche Grafikkarten verwendet werden, kann in der Systemkonfiguration angegeben werden.

CPU-Version Die CPU-Version verwendet ebenfalls einen Hauptthread wie die CUDA-Version, ausser, dass keine Grafikkarten verwaltet werden. Zudem werden Threads gestartet, um Raytracing parallel auf der CPU durchzuführen. Die Anzahl dieser Berechnungsthreads kann in der Systemkonfiguration angegeben werden.

[1] Die ersten vier Abschnitte dieses Kapitels basieren auf [LD10.1, Kapitel 37].
[2] Programme haben aber häufig die Möglichkeit, weitere Prozesse zu starten.

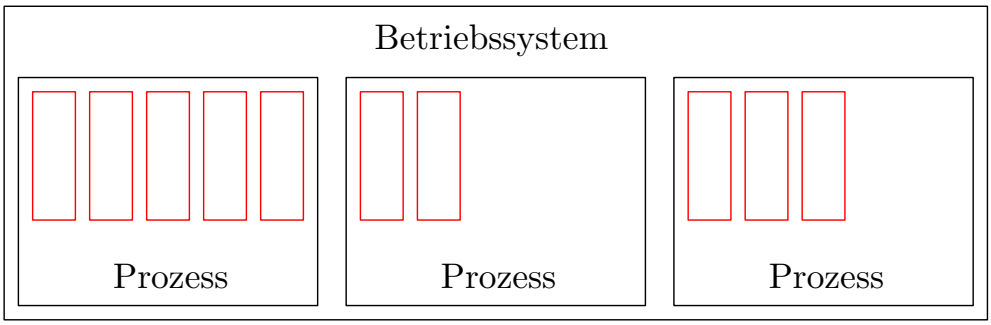


Abbildung 5.2.1: Mögliche Kombination von Prozessen und Threads. Threads sind als rote Rechtecke dargestellt.

5.2.2 GPU [3]

GPU steht für *Graphics processing unit* (Grafik-Recheneinheit, Grafikkarte). GPUs sind meistens Steckkarten oder Chips, die an der Hauptplatine des Computers angeschlossen werden. Sie wurden ursprünglich für Grafikberechnungen konzipiert, können jedoch auch andere Aufgaben ausführen; letzteres wird als *GPGPU* (**G**eneral **p**urpose computation on **g**raphics **p**rocessing **u**nit) bezeichnet.

GPUs können in vielerlei Hinsicht wie kleine Computer betrachtet werden; sie besitzen einen oder mehrere Grafikprozessoren (in CUDA auch als *Stream-Prozessoren* bekannt) und haben eine spezielle Art von Arbeitsspeicher, sogenannte *VRAM* (Video-RAM).

CUDA-Kerne Grafikprozessoren sind das Pendant zur CPU, von dieser jedoch sehr verschieden. Grafikprozessoren haben, mit einer CPU verglichen, eine sehr grosse Anzahl an Kernen, welche in CUDA *CUDA-Kerne* genannt werden. CUDA-Kerne haben im Vergleich zu CPU-Kernen eher Mühe mit komplexen Algorithmen, welche viele logische Verzweigungen besitzen. Sie sind darauf spezialisiert, gleiche Aufgaben parallel auszuführen; diese Art von Parallelität wird als *Datenparallelität* bezeichnet, da die Kerne an verschiedenen Daten im Gegensatz zu verschiedenen Aufgaben arbeiten.

Kernel Programme auf dem Grafikprozessor heissen *Kernel*. Sie können von einem CPU-Programm gestartet werden. Der Code eines Kernels wird in der Regel mehrere Male parallel ausgeführt. CUDA stellt zwei Unterteilungen zur Verfügung, um die parallelen Codeinstanzen eines Kernels zu organisieren.

Blöcke Die erste Unterteilung sind *Blöcke*. Blöcke bilden virtuell ein dreidimensionales Gitter, der Programmierer muss anstelle einer einfachen Anzahl Blöcke die Dimensionen des Gitters angeben. Die drei Dimensionen sind in manchen Problemstellungen, in denen die Daten bereits dreidimensional vorliegen, von Vorteil; natürlich kann auch nur eine Dimension verwendet werden, wenn die anderen auf eins gesetzt werden. Jede GPU hat eine maximale Gitter-

[3] Dieses Kapitel basiert auf [CJ14.1, Kapitel 3].

grösse, welche nicht überschritten werden darf; sie kann zur Laufzeit abgefragt werden.

Ein Block wird nie über mehrere Grafikprozessoren verteilt; es kann aber vorkommen, dass auf einem Grafikprozessor mehrere Blöcke ausgeführt werden.

Threads Blöcke werden weiter in Threads unterteilt. Jeder Block eines Kernels hat die gleiche Anzahl an Threads. Threads sind wie Blöcke virtuell dreidimensional angeordnet. Es wird immer ein Thread pro CUDA-Kern ausgeführt; aus diesem Grund sollte die Anzahl Threads pro Block die Anzahl CUDA-Kerne pro Grafikprozessor nicht überschreiten. Threads können direkt mit anderen Threads desselben Blocks kommunizieren und synchronisieren. Threads werden immer in Gruppen von 32 ausgeführt, sogenannte *Warps*. Innerhalb einer Warp führen alle Threads die gleiche Berechnung durch. Abbildung 5.2.2 zeigt eine mögliche Anordnung von Blöcken und Threads innerhalb eines Kernels.

Diese und weitere Einschränkungen bestimmen, wie die optimale Gitterkonfigurationen für eine gegebene Problemstellung aussehen. Der *CUDA Occupancy Calculator* [4] ermöglicht, die theoretische Effizienz einer Konfiguration zu berechnen. Kapitel 6.1 befasst sich mit den Dimensionen der Gitter, welche in `cuRRay` verwendet werden.

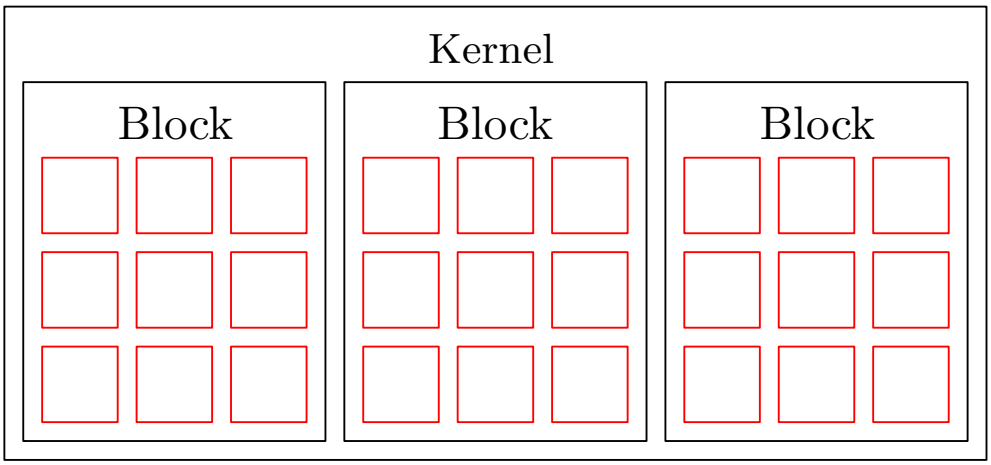


Abbildung 5.2.2: Mögliche Dimensionen des Block- und Thread-Gitters innerhalb eines Kernels. Die Threads sind als rote Quadrate dargestellt.

Anwendungen `cuRRay` verwendet CUDA beim Raytracing. Diese Aufgabe hat eine hohe Datenparallelität; eine Implementation mit CUDA ist deshalb geeignet. Da Photonen, bzw. Pixel nicht miteinander interagieren, können wir alle Photonen einem Thread zuweisen und das Problem in Blöcke aufteilen, so dass die Effizienz maximal ist.

[4] `developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls`

Kapitel 6

Raytracing

Wie in Kapitel 4.1 bereits beschrieben, verwenden wir in `cuRRay` einen Raytracing-Algorithmus, um die Farbe und Rotverschiebung jedes Pixels zu berechnen.

Grundlegende Idee `cuRRay` berechnet für jeden Pixel den dazugehörigen Lichtstrahl rückwärts. Der Lichtstrahl ist eindeutig durch die Position des Beobachters und die Richtung des auftreffenden Lichtstrahls definiert. Weil die Kerr-Newman-Metrik eine Zeitumkehrung zulässt [1], können wir Lichtstrahlen vorwärts gleich berechnen wie rückwärts.

Wir denken uns deshalb neue Lichtstrahlen, die an der Position des Beobachters entstehen und als Anfangsgeschwindigkeiten die umgekehrten Auftreffgeschwindigkeiten der ursprünglichen, einfallenden Lichtstrahlen besitzen. Wir können diese neuen Lichtstrahlen vorwärts berechnen, um die Quellen der einfallenden Strahlen zu finden. Wir können uns vorstellen, dass der Beobachter für jeden Pixel einen Lichtstrahl aussendet. Die Lichtstrahlen sind die Geodäten von Photonen. Wir berechnen deshalb die Lichtstrahlen durch Lösen der Geodätengleichungen von Photonen.

In diesem Kapitel werden wir sehen, wie dies genau funktioniert und wie aus den gefundenen Quellen Bilder erstellt werden.

6.1 Gitter-Layout

Damit die Berechnungen möglichst effizient sind, muss der Rechenaufwand auf die verschiedenen Grafikprozessoren der GPU verteilt werden. Wir beschreiben hier, wie dies in `cuRRay` funktioniert.

GPUs Weil der Datenaustausch zwischen verschiedenen GPUs relativ langsam und mühsam ist, wurde beschlossen, nur eine GPU pro Bild zu verwenden. Falls ein System mehrere GPUs besitzt, können aber mehrere Bilder gleichzeitig gezeichnet werden.

[1] Wie in Kapitel 2.2 diskutiert, ändert sich die Drehrichtung des schwarzen Lochs, wenn die Zeit umgekehrt wird; ansonsten ist die Raumzeit komplett zeitsymmetrisch.

Kernel, Threads und Blöcke Der Raytracing-Algorithmus besteht aus einem einzigen Kernel, der für jeden Lichtstrahl die Geodätengleichung integriert. Wir werden den Kernel weiter unten genauer betrachten.

Für jeden Pixel, beziehungsweise Lichtstrahl, ist ein Thread zuständig. Das Problem wird in mehrere Blöcke unterteilt, so dass die Berechnungen möglichst effizient sind. Um die Gitterstrukturen zu berechnen, geht `cuRRay` folgendermassen vor [2]:

1. Jeder Thread benötigt eine Anzahl *Register*, Speicherbereiche für Variablen, die für die Berechnungen benötigt werden. `cuRRay` berechnet, wie viele Register pro Warp benötigt werden. Dabei wird darauf geachtet, dass Register pro Warp stets in 256er-Gruppen organisiert sind. Dies wird *Register-Granularität* genannt.

2. Als nächstes wird berechnet, wie viele Warps pro Grafikprozessor maximal ausgeführt werden können, ohne dass die maximale Anzahl Register pro Grafikprozessor von den Warps überschritten wird. Es wird beachtet, dass Warps in Vierergruppen organisiert sind; dies wird *Warp-Granularität* genannt. Ziel ist es, möglichst an diesen Maximalwert heranzukommen.

3. Die maximale Anzahl Warps pro Block wird berechnet. Hier werden Hardware-Beschränkungen und Warp-Granularität beachtet.

4. Finden mehr Warps in einem Block Platz, als maximal auf einem Grafikprozessor ausgeführt werden können, wird ein Block pro Grafikprozessor verwendet. Die Blockgrösse wird dementsprechend berechnet und der nächste Schritt übersprungen.

5. `cuRRay` berechnet für verschiedene Konfigurationen (2, 3, 4 und 5 Blöcke pro Grafikprozessor), wie viele Warps verwendet werden. Dabei wird auf die Warp-Granularität geachtet. Die Konfiguration, welche dem vorher berechneten Maximalwert am nächsten kommt, wird für die Berechnung der Blockgrösse verwendet.

6. Die Threads werden gemäss der berechneten Blockgrösse auf Blöcke verteilt.

Überschuss Generell werden mehr Threads gestartet als notwendig. Alle überschüssigen Threads werden sofort beendet. Maximal ein Block enthält überschüssige Threads.

[2] Der genaue Algorithmus kann im C++-Quellcode in der Datei `gpuManager.cpp` eingesehen werden.

6.2 Abarbeitung der Frames

Die Szenendatei wird zu Beginn des Raytracing in den Arbeitsspeicher geladen. `cuRRay` erstellt einen Stapel mit allen Frames.

CUDA-Version Jede GPU, welche aktiviert wurde, kann eine Frame vom Stapel verlangen. Anschliessend werden die Szenendaten in VRAM geladen. Hier werden auch die Werte von animierten Variablen für die aktuelle Frame berechnet. Die Frame wird berechnet. Sobald das Raytracing beendet ist und die Ausgabedateien erstellt wurden, kann die GPU eine neue Frame vom Stapel verlangen. So werden alle Frames auf dem Stapel von beliebig vielen GPUs abgearbeitet.

CPU-Version In der CPU-Version wird der Stapel nicht parallel, sondern seriell abgearbeitet. Die Szenendaten werden hier in gewöhnlichen Arbeitsspeicher geladen. Anschliessend wird die Frame parallel mittels verschiedenen Threads berechnet.

6.3 Anfangswertproblem

Die Geodäte eines Photons wird eindeutig durch die Krümmung der Raumzeit und durch *Anfangsbedingungen* bestimmt. In diesem Kapitel fassen wir zusammen, von welchen Bedingungen die Geodäten der Photonen in unserem Fall abhängen.

Differentialgleichungssystem Die Gleichungen (2.3.8), (2.3.9), (2.3.10) und (2.3.11) bilden das Differentialgleichungssystem, welches `cuRRay` integriert. Sie bestimmen, welchen Einfluss die Raumzeitkrümmung auf die Geodäten der Photonen hat.

Anfangsbedingungen Das im letzten Abschnitt beschriebene Gleichungssystem hat unendlich viele Lösungen, da unendlich viele unterschiedliche Geodäten von Photonen existieren. Um die Lösungen zu finden, welche unsere Lichtstrahlen beschreiben, brauchen wir *Anfangsbedingungen*. Jeder Lichtstrahl ist durch die ursprüngliche Position und ursprüngliche Geschwindigkeit des Photons genau definiert.

Die Anfangsposition $x^\alpha(\lambda = 0)$ jedes Photons entspricht der Position des Beobachters. Die Anfangsgeschwindigkeit $u^\alpha(\lambda = 0)$ kann aus der Position und Ausrichtung des Beobachters berechnet werden. Aus diesen Anfangsbedingungen kann zudem E und L gemäss Gleichungen (2.3.5) und (2.3.6) berechnet werden.

Anfangswertproblem Die Anfangsbedingungen $x^\alpha(\lambda = 0)$, $u^\alpha(\lambda = 0)$, E und L bilden zusammen mit dem Differentialgleichungssystem ein *Anfangswertproblem*.

6.4 Berechnung der Anfangsgeschwindigkeit

Für jedes Frame berechnet `cuRRay` aus der Position, der Ausrichtung und dem Sichtfeld des Beobachters die Anfangsgeschwindigkeitsvektoren $u_0^\alpha = dx^\alpha/d\lambda$ ($\lambda = 0$) der Photonen. Wie wir in Kapitel 6.3 sahen, sind diese für die Berechnung der Lichtstrahlen notwendig.

Inertialsystem Wir berechnen die Anfangsgeschwindigkeiten zuerst im Inertialsystem des Beobachters. Dies ist möglich, da wegen dem Äquivalenzprinzip überall in der Raumzeit für jeden Beobachter stets ein Inertialsystem gefunden werden kann. Wir wählen als flache Metrik dieses Systems die Minkowski-Metrik $\eta_{\alpha\beta} = \mathrm{diag}(-1, 1, 1, 1)$. Dies bedeutet, dass alle Vektoren kartesisch sind.

`cuRRay` geht von stationären Beobachtern aus; das heisst, es können keine Beobachter innerhalb der Ergosphäre gewählt werden.

Konstruktion der Geschwindigkeitsvektoren Wir denken uns für das Inertialsystem kartesische Koordinaten, in denen der Beobachter um den Nullpunkt zentriert ist und in Richtung der negativen x-Achse schaut. Vor dem Beobachter wird ein rechteckiger Bildschirm aufgespannt, welcher gleichmässig in Pixel unterteilt ist. Die horizontale und vertikale Auflösung entsprechen den Massen des Frames.

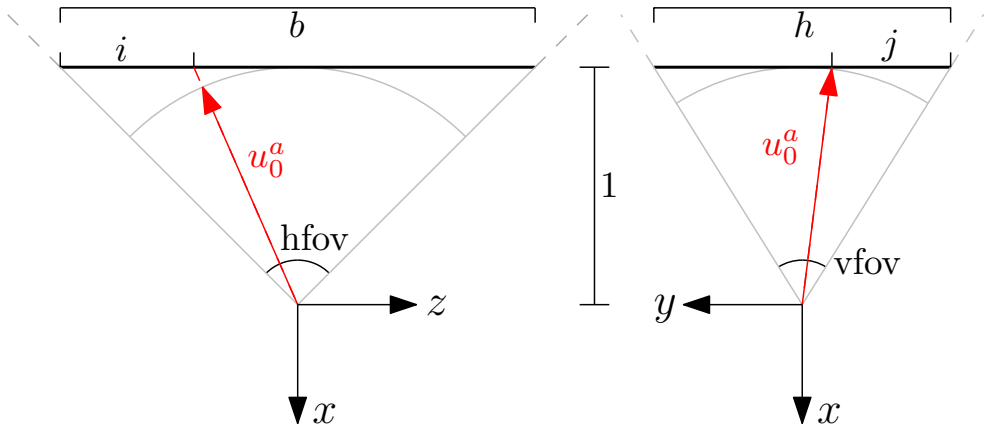


Abbildung 6.4.1: Konstruktion des Anfangsgeschwindigkeitsvektors in kartesischen Koordinaten für den Pixel (i, j). b und h sind die Breite bzw. Höhe des Frames. Der Beobachter befindet sich am Koordinatenursprung.

Der Schirm ist parallel zur yz-Ebene und befindet sich im Abstand 1 zum Beobachter. Strahlen, welche durch den Nullpunkt und die Ecken des Schirms gehen, grenzen das Sichtfeld ein. Die Breite und Höhe des Schirms werden nun skaliert, so dass die Winkel des Sichtfeldes (hfov und vfov) den Einstellungen aus den Szeneninformationen entsprechen.

Die Koordinaten des aktuellen Pixels werden ermittelt und der dazugehörige Punkt auf dem Bildschirm wird bestimmt. Der räumliche Teil des Geschwindigkeitsvektors u_0^α ist parallel zum Ortsvektor dieses Punkts und zeigt

in die gleiche Richtung. Wir normalisieren den räumlichen Teil auf die Länge 1 und setzen die Zeit-Komponente auf 1, so dass der neue Geschwindigkeitsvektor gemäss $\eta_{\alpha\beta}$ ein Nullvektor ist.

Abbildung 6.4.1 zeigt, wie der Geschwindigkeitsvektor für den Pixel (i, j) konstruiert wird.

Ausrichtung Die Koordinatenvektorfelder ∂_r, ∂_θ und ∂_ϕ sind alle rechtwinklig zueinander, wie wir anhand von Gleichung (2.2.1) erkennen können. Deshalb ist es möglich, die lokalen kartesischen Koordinaten des Inertialsystems folgendermassen auszurichten: Der Basisvektor der x-Achse ist tangential zum Koordinatenvektorfeld ∂_r und zeigt in die gleiche Richtung. Dasselbe gilt für den Basisvektor der y-Achse mit ∂_θ und dem Basisvektor der z-Achse mit ∂_ϕ.

Abbildung 6.4.2 zeigt, wie wir die Koordinaten ausrichten möchten. Diese

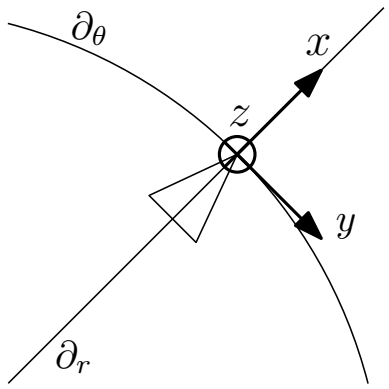


Abbildung 6.4.2: Ausrichtung der lokalen kartesischen Koordinaten relativ zu den BL-Koordinaten. Dargestellt ist ein Schnitt ϕ = konst durch die Raumzeit. Neben der kartesischen Koordinatenbasis sind die durch die Koordinatenvektorfelder ∂_r und ∂_θ definierten Koordinatenfeldlinien und der aufgespannte Bildschirm eingezeichnet. Der Beobachter befindet sich im Ursprung des kartesischen Koordinatensystems.

Ausrichtung sorgt dafür, dass der Beobachter standardmässig das schwarze Loch direkt betrachtet.

Der Beobachter kann zusätzlich mittels *Roll-*, *Nick-* und *Gierwinkel* um die drei kartesischen Achsen in folgender Reihenfolge gedreht werden: Der Rollwinkel dreht ihn um die negative x-Achse, der Nickwinkel um die negative z-Achse und der Gierwinkel um die negative y-Achse. Alle Winkel werden in der Szenendatei angegeben. Nachdem die Geschwindigkeitsvektoren im lokalen kartesischen Koordinatensystem konstruiert werden, wendet `cuRRay` diese drei Rotationen auf sie an.

Koordinatentransformation Bevor wir die Vektoren aber fürs Raytracing verwenden können, müssen wir sie in BL-Koordinaten transformieren. Wir suchen dazu eine Koordinatentransformation $\Psi^\alpha{}_\beta$, welche Vektoren von unseren lokalen kartesischen Koordinaten zu BL-Koordinaten umwandelt:

$$v^\alpha_{\mathrm{BL}} = \Psi^\alpha{}_\beta v^\beta_{\mathrm{k}}. \tag{6.4.1}$$

Gleichzeitig muss die Koordinatentransformation die kartesischen Basisvektoren so ausrichten, wie wir im vorherigen Abschnitt verlangten (siehe Abbildung 6.4.2), und dafür sorgen, dass der Beobachter stationär ist. Siehe Anhang D für die Herleitung von ${\Psi^{\alpha}}_{\beta}$.

Die transformierten Vektoren werden anschliessend als Anfangsbedingungen im Raytracing-Algorithmus verwendet.

6.5 Runge-Kutta-Algorithmus vierter Ordnung

Das in Kapitel 6.3 beschriebene Anfangswertproblem kann analytisch nicht gelöst werden; wir müssen es numerisch integrieren, um die Position $\boldsymbol{x}(\lambda)$ jedes Photons als Funktion des affinen Parameters zu erhalten.

Einschrittverfahren Wenn

$$y'(x) = f(x, y(x)), \qquad y(x_0) = y_0, \quad y'(x_0) = y'_0 \tag{6.5.1}$$

unser Anfangswertproblem und $y(x)$ die gesuchte Funktion ist, können wir die kontinuierliche Funktion $y(x)$ durch diskrete Werte y_i an den Positionen x_i approximieren (vgl. [BI78.1, S. 709]).

Dazu führen wir eine potentiell variable *Schrittweite* h ein, so dass

$$x_{i+1} = x_i + h. \tag{6.5.2}$$

y ist dann durch die *Differenzengleichung*

$$y_{i+1} = y_i + h\hat{f}(x_i, y_i), \quad y(x_0) = y_0 \tag{6.5.3}$$

an den diskreten Stellen definiert. $\hat{f}$ ist die *Schrittfunktion* und hängt vom gewählten Algorithmus ab. Alle numerischen Verfahren, welche nach dem Schema von Gleichung (6.5.3) funktionieren, heissen *Einschrittverfahren*, weil jeder diskrete Wert y_i nur aus den Werten des letzten Schritts bestimmt wird. Vgl. [BI78.1, S. 709].

Fehler Für einen numerischen Algorithmus kann die tiefste Ordnung des Fehlers in h angegeben werden. Je höher die Ordnung des Fehlers, desto besser die numerische Approximation. Der absolute Fehler kann trotzdem bestenfalls abgeschätzt werden. Vgl. [BI78.1, S. 709].

RK4-Verfahren Wir verwenden für die Lösung des Anfangswertproblems den *Runge-Kutta-Algorithmus vierter Ordnung* (kurz: RK4). RK4 ist ein Einschrittverfahren mit einem Fehler der Ordnung h^5.

Die Schrittfunktion des RK4-Algorithmus ist

$$\begin{aligned}
\hat{f}(x_i, y_i) &= \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\
K_1 &= f\left(x_i, y_i\right) \\
K_2 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}K_1\right)
\end{aligned}$$

$$
\begin{aligned}
K_3 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}K_2\right) \\
K_4 &= f\left(x_i + h, y_i + hK_3\right).
\end{aligned} \tag{6.5.4}
$$

$\hat{f}$ ist ein gewichteter Mittelwert von f an verschiedenen Teilschritten. Vgl. [BI78.1, S. 710].

Lösen der Geodätengleichungen Die Geodätengleichungen sind gekoppelt, das heisst, dass wir sie gleichzeitig integrieren müssen, weil sie voneinander abhängen. Wir erreichen dies, indem wir einen Teilschritt für alle Differentialgleichungen berechnen, bevor wir mit dem nächsten weiterfahren.

Gleichungen erster Ordnung Die Gleichungen (2.3.8) und (2.3.9) sind von der Form $y' = f(y)$. Sie sind für die Integration mit RK4 geeignet. Ein einzelner Schritt in x^t wird folgendermassen durchgeführt:

$$
\begin{aligned}
x^t_{i+1} &= x^t_i + h\frac{1}{6}(X^t_1 + 2X^t_2 + 2X^t_3 + X^t_4) \\
X^t_1 &= t'\left(x^\alpha_i\right) \\
X^t_2 &= t'\left(x^\alpha_i + \frac{h}{2}X^\alpha_1\right) \\
X^t_3 &= t'\left(x^\alpha_i + \frac{h}{2}X^\alpha_2\right) \\
X^t_4 &= t'\left(x^\alpha_i + hX^\alpha_3\right).
\end{aligned} \tag{6.5.5}
$$

Analog dazu ist ein Schritt in x^ϕ gegeben durch

$$
\begin{aligned}
x^\phi_{i+1} &= x^\phi_i + h\frac{1}{6}(X^\phi_1 + 2X^\phi_2 + 2X^\phi_3 + X^\phi_4) \\
X^\phi_1 &= \phi'\left(x^\alpha_i\right) \\
X^\phi_2 &= \phi'\left(x^\alpha_i + \frac{h}{2}X^\alpha_1\right) \\
X^\phi_3 &= \phi'\left(x^\alpha_i + \frac{h}{2}X^\alpha_2\right) \\
X^\phi_4 &= \phi'\left(x^\alpha_i + hX^\alpha_3\right).
\end{aligned} \tag{6.5.6}
$$

Die Primen bedeuten Ableitungen nach λ. t' und ϕ' sind durch die Gleichungen (2.3.8) und (2.3.9) gegeben.

Gleichungen zweiter Ordnung Die Gleichungen (2.3.10) und (2.3.11) sind von der Form $y'' = f(y, y')$. Damit wir y bestimmen können, müssen wir zweimal integrieren. Wir teilen dazu die Gleichungen auf:

$$
y'' = f(y, y') \rightarrow \left\{ \begin{array}{l} x_1 = y'' \\ x_2 = y' \end{array} \right. . \tag{6.5.7}
$$

Die Ausdrücke auf der rechten Seite in (6.5.7) bilden wieder ein Differentialgleichungssystem. Jede Gleichungen zweiter Ordnung wurde mit zwei gekoppelten

Differentialgleichungen erster Ordnung ersetzt. Diese neuen Gleichungen sind ebenfalls mit den bereits weiter oben besprochenen Differentialgleichungen für t und ϕ gekoppelt und müssen deshalb ebenfalls simultan integriert werden.

Ein Schritt in x^r ist

$$\begin{aligned}
x^r_{i+1} &= x^r_i + h\frac{1}{6}(X^r_1 + 2X^r_2 + 2X^r_3 + X^r_4) \\
X^r_1 &= u^r \\
X^r_2 &= u^r + \frac{h}{2}U^r_1 \\
X^r_3 &= u^r + \frac{h}{2}U^r_2 \\
X^r_4 &= u^r + hU^r_3 \\
\\
u^r_{i+1} &= u^r_i + h\frac{1}{6}(U^r_1 + 2U^r_2 + 2U^r_3 + U^r_4) \\
U^r_1 &= r''\left(x^\alpha_i, u^\alpha_i\right) \\
U^r_2 &= r''\left(x^\alpha_i + \frac{h}{2}X^\alpha_1, u^\alpha_i + \frac{h}{2}U^\alpha_1\right) \\
U^r_3 &= r''\left(x^\alpha_i + \frac{h}{2}X^\alpha_2, u^\alpha_i + \frac{h}{2}U^\alpha_2\right) \\
U^r_4 &= r''\left(x^\alpha_i + hX^\alpha_3, u^\alpha_i + hU^\alpha_3\right)
\end{aligned} \tag{6.5.8}$$

Analog gilt für einen Schritt in x^θ:

$$\begin{aligned}
x^\theta_{i+1} &= x^\theta_i + h\frac{1}{6}(X^\theta_1 + 2X^\theta_2 + 2X^\theta_3 + X^\theta_4) \\
X^\theta_1 &= u^\theta \\
X^\theta_2 &= u^\theta + \frac{h}{2}U^\theta_1 \\
X^\theta_3 &= u^\theta + \frac{h}{2}U^\theta_2 \\
X^\theta_4 &= u^\theta + hU^\theta_3 \\
\\
u^\theta_{i+1} &= u^\theta_i + h\frac{1}{6}(U^\theta_1 + 2U^\theta_2 + 2U^\theta_3 + U^\theta_4) \\
U^\theta_1 &= \theta''\left(x^\alpha_i, u^\alpha_i\right) \\
U^\theta_2 &= \theta''\left(x^\alpha_i + \frac{h}{2}X^\alpha_1, u^\alpha_i + \frac{h}{2}U^\alpha_1\right) \\
U^\theta_3 &= \theta''\left(x^\alpha_i + \frac{h}{2}X^\alpha_2, u^\alpha_i + \frac{h}{2}U^\alpha_2\right) \\
U^\theta_4 &= \theta''\left(x^\alpha_i + hX^\alpha_3, u^\alpha_i + hU^\alpha_3\right)
\end{aligned} \tag{6.5.9}$$

r'' und θ'' sind durch die Gleichungen (2.3.10) und (2.3.11) gegeben.

Die Gleichungen (6.5.5), (6.5.6), (6.5.8) und (6.5.9) definieren, wie `cuRRay` einen einzelnen RK4-Schritt durchführt.

6.6 Schrittweitensteuerung

Wir steuern die Schrittweite h mit dem Algorithmus, welcher in [PD11.1] verwendet wird. h passt sich der am schnellsten ändernden Koordinate an:

$$h = c \cdot \max\left(|u^t|, |u^r|, |u^\theta|, |u^\phi|\right)^{-1}. \tag{6.6.1}$$

c ist eine frei wählbare Konstante. `cuRRay` verwendet standardmässig $c = 1/32$. c kann in der Systemkonfigurationsdatei angepasst werden.

Um sicherzugehen, dass h genügend klein gewählt wird, kann die Konstante c verkleinert werden. Sobald sich das Bild beim Verringern von c nicht verändert, ist c klein genug.

6.7 Überwachung des Geschwindigkeitsvektors

Weil Photonen sich auf Null-Geodäten bewegen, ist die Magnitude ihres Geschwindigkeitsvektors stets null. Wir verwenden diese Eigenschaft, um die Genauigkeit des Raytracing-Algorithmus zu kontrollieren. Wir bemerken, dass angewandt auf den Geschwindigkeitsvektor eines Photons, ein beliebiger Teil der Summe in (2.2.1) geteilt durch den Rest der Summe stets -1 ergibt. Ein Vergleich mit -1 ermöglicht uns, einen relativen Fehler zu bestimmen. Hier teilen wir durch die Zeit-Zeit-Komponente, weil diese als einzige ausserhalb des Ereignishorizonts nur einmal null wird, nämlich wenn sich das Photon genau auf der Ergosphäre befindet. Die anderen Summanden können je nach Geschwindigkeit des Photons mehrmals null werden. Der Fehler e wird folgendermassen berechnet:

$$e \equiv \left|1 + \left[g_{rr} \cdot (u^r)^2 + g_{\theta\theta} \cdot (u^\theta)^2 + g_{\phi\phi} \cdot (u^\phi)^2 + 2g_{t\phi} u^t u^\phi\right] \Big/ \left[g_{tt} \cdot (u^t)^2\right]\right|. \tag{6.7.1}$$

e zeigt an, ob der Raytracing-Algorithmus korrekt und ob die Schrittweitensteuerung effektiv ist. e wird für jeden Schritt berechnet. Pro Lichtstrahl wird der durchschnittliche Fehler $\bar{e}$ und die Standardabweichung σ_e des Fehlers berechnet. In Kapitel 8.1 werden wir die Genauigkeit von `cuRRay` diskutieren.

6.8 Abbruchbedingungen

Die Integration einer Geodäte wird beendet, wenn eine der Abbruchbedingungen erfüllt ist. Je nach Art der Abbruchbedingung werden unterschiedliche Pixeldaten geschrieben.

Ereignishorizont Die r-Koordinate des Ereignishorizonts ist r_+ und durch Gleichung (2.2.5) gegeben. Wir stoppen die Integration, wenn die radiale Koordinate r des Photons kleiner als $r_+ + \epsilon$ wird. ϵ kann in der Systemkonfigurationsdatei gesetzt werden und ist standardmässig 10^{-2}. ϵ verhindert, dass die Schrittweiten von Geodäten nahe des Ereignishorizonts beliebig klein wer-

den. Pixel, deren Photonen diese Abbruchbedingung erfüllen, werden mit der Ereignishorizontfarbe eingefärbt.

Szenengrenze Die Szenengrenze $r_{\max}$ wird so gewählt, dass sich alle Objekte der Szene, einschliesslich Beobachter, innerhalb von $r_{\max}$ befinden. Ausserdem kann die Grenze explizit grösser gewählt werden. Überschreitet die radiale Koordinate eines Photons $r_{\max}$, wird der Pixel mit der Himmelsfarbe eingefärbt. Falls ein Sternenhimmel konfiguriert wurde, berechnet `cuRRay` die Farbe des Pixels gemäss Kapitel 6.10.

Fehler Tritt während einem RK4-Schritt ein Fehler auf, wird der Pixel mit der Fehlerfarbe eingefärbt. Als Fehler gilt, wenn entweder $r < 0$, $\theta < 0$ oder $\theta > \pi$. Wie solche Fehler entstehen können, untersuchen wir in Kapitel 8.1.

Kugeln Nach jedem Schritt wird für jeden Lichtstrahl berechnet, ob eine der Kugeln, welche in der Szenendatei definiert werden können, getroffen wurde. Diese Kollisionserkennung wird in kartesischen Koordinaten durchgeführt. Auch hier wird dabei aus den Gründen, welche bereits weiter oben dargelegt worden sind, die Raumzeitkrümmung ignoriert. Der Pixel wird entsprechend dem Schachbrettmuster und der gewählten Farbe eingefärbt.

Akkretionsscheibe Ist die Akkretionsscheibe konfiguriert, wird bei jedem Schritt und für jeden Strahl getestet, ob sie getroffen wurde. Diese Kollisionserkennung wird in BL-Koordinaten durchgeführt. Der Pixel wird entsprechend dem Schachbrettmuster und den gewählten Farben eingefärbt.

6.9 Rotverschiebung

`cuRRay` berechnet für alle Lichtstrahlen das Verhältnis von Empfängerfrequenz ω_E (durch den Beobachter gemessene Frequenz) und Senderfrequenz ω_S (Frequenz der Quelle) gemäss Gleichung (3.5.1). Die Berechnung wird nur durchgeführt, wenn sich die Quelle ausserhalb der Ergosphäre befindet. Die möglichen Werte für das Verhältnis reichen von ∞ bis 0.

Falls die Ausgabe der Rotverschiebung aktiviert ist, wird eine zusätzliche PNG-Datei angelegt, um die Daten zur Rotverschiebung zu speichern. Alle Pixel, deren Quelle entweder der Ereignishorizont, eine der Kugeln oder die Akkretionsscheibe ist, werden folgendermassen eingefärbt:

$$\begin{aligned} R &= 255 - 255 \cdot \arctan\left(\frac{\omega_E}{\omega_S}\right)\frac{2}{\pi} \\ G &= 0 \\ B &= 255 \cdot \arctan\left(\frac{\omega_E}{\omega_S}\right)\frac{2}{\pi}. \end{aligned} \tag{6.9.1}$$

Jede Farbkomponente kann Werte von 0 (keine Intensität) bis 255 (maximale Intensität) annehmen. Ein komplett roter Pixel bedeutet unendliche Rotverschiebung, ein komplett blauer Pixel unendliche Blauverschiebung und ein

violetter Pixel bedeutet, dass keine Verschiebung stattfand. Falls die Quelle des Pixels innerhalb der Ergosphäre liegt, wird der Pixel grün eingefärbt. Alle Pixel, die zm Himmel gehören oder fehlerhaft sind, werden schwarz eingefärbt.

6.10 Sternenhimmel

Es besteht die Möglichkeit, den Himmel der Szene mit einem Bild zu überziehen. Dies ermöglicht zum Beispiel die Darstellung eines realistischen Sternenhimmels im Hintergrund des schwarzen Lochs.

Bilddatei `cuRRay` benötigt für die Berechnung des Himmels eine *Plattkartenprojektion* des Himmels in Form einer PNG-Datei. Die Plattkartenprojektion hat die Eigenschaft, dass die sphärischen Koordinaten θ und ϕ eines Punkts auf der Himmelskugel direkt den kartesischen Koordinaten x und y des Pixels in der Bilddatei entsprechen:

$$\begin{aligned} \theta &= y \cdot \frac{\pi}{h} \\ \phi &= 2\pi - x \cdot \frac{2\pi}{b}. \end{aligned} \tag{6.10.1}$$

h ist die Höhe der Bilddatei und b die Breite. θ ist der Zenitwinkel, ϕ der Azimut. Abbildung 6.10.1 zeigt beispielsweise eine Plattkartenprojektion des Nachthimmels, welche wir in Kapitel 7 für die Darstellung des Sternenhimmels verwenden werden.

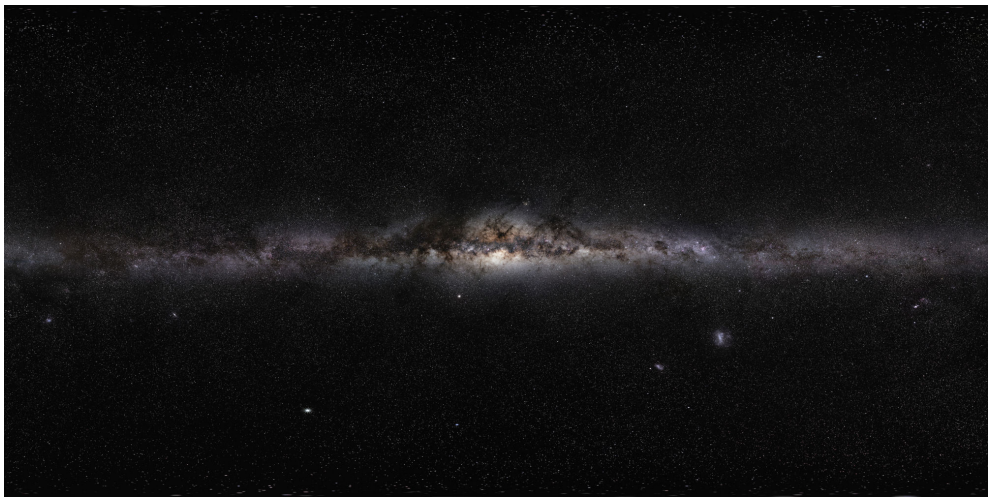


Abbildung 6.10.1: Plattkartenprojektion des Nachthimmels. Sterne nahe der galaktischen Pole erscheinen wegen der Projektion in die Breite gezogen. Bildquelle: ESO/ S. Brunier, [BS09.1].

Asymptotische Fläche Weil die Kerr-Newman-Raumzeit asymptotisch flach ist, können wir eine r-Koordinate R finden, so dass Lichtstrahlen ausserhalb von R nicht mehr merkbar abgelenkt werden.

R kann in der Szenendatei eingestellt werden. Ist R grösser als die Szenengrenze $r_{\max}$ (siehe Kapitel 6.8), wird $r_{\max} = R$ gesetzt. Ist $R < r_{\max}$, wird bis $r_{\max}$ integriert.

Richtungsvektor Wir nehmen an, dass Lichtstrahlen, welche die Szenengrenze erreichen, nicht mehr stark abgelenkt werden, und können sie von nun an vereinfacht als Geraden in flacher Raumzeit betrachten. Weiter gehen wir davon aus, dass die Himmelskugel einen unendlichen Radius hat, sprich, dass die Sterne unendlich weit entfernt sind. Dies ist eine gerechtfertigte Annahme, da die Distanzen zu den Sternen in der Regel viel grösser sind (Lichtjahre) als die typischen Distanzen zwischen den Objekten einer Szene.

Von der Himmelskugel aus gesehen, scheint wegen der Grösse der Kugel jedes Photon, welches die Szenengrenze überschreitet, aus der Mitte der Szene zu kommen. Deshalb spielt nur die Richtung des Photons zum Zeitpunkt des Überschreitens der Szenengrenze eine Rolle. Wir transformieren den Geschwindigkeitsvektor des Photons an der Szenengrenze mittels Gleichung (1.5.1) von BL-Koordinaten in kartesische Koordinaten:

$$u^{\alpha}_{\mathrm{K}} = \sum_{\beta} \frac{\partial x^{\alpha}_{\mathrm{K}}}{\partial x^{\beta}_{\mathrm{BL}}}\, u^{\beta}_{\mathrm{BL}}, \tag{6.10.2}$$

wobei Subskript-BL Boyer-Lindquist-Komponenten und Subskript-K kartesische Komponenten bezeichnet. Die partiellen Ableitungen können wir aus den Gleichungen (2.2.8) berechnen.

Wir normalisieren den räumlichen Teil des Geschwindigkeitsvektors und erhalten den *kartesischen Richtungsvektor*:

$$n^a = \frac{u^a_{\mathrm{K}}}{|u^a_{\mathrm{K}}|}, \tag{6.10.3}$$

wobei $|u^a_{\mathrm{K}}|$ die Länge des räumlichen Teils des kartesischen Geschwindigkeitsvektors ist.

Himmelspunkt Wir berechnen die Winkelkoordinaten θ und ϕ des Himmelspunkts, der vom Photon getroffen wird, aus dem kartesischen Richtungsvektor:

$$\begin{aligned} \tan\phi &= \frac{n^y}{n^x}, \\ \cos\theta &= n^z. \end{aligned} \tag{6.10.4}$$

Wir erhalten die Koordinate des Pixels in der Bilddatei, indem wir die Gleichungen (6.10.1) anwenden. Schliesslich färben wir den Pixel des Frames mit der Farbe des Pixels der Bilddatei ein.

6.11 Ausführliche Pixeldaten

Wie bereits in Kapitel 4.1 erwähnt, kann für jede Frame eine CSV-Datei mit ausführlichen Pixeldaten angelegt werden. Für jeden Pixel wird eine Zeile angelegt. Die folgenden Spalten werden geschrieben: x- und y-Koordinate des

Pixels, R-, G- und B- Farbkomponente des Pixels, die Anzahl benötigter RK4-Schritte, das Verhältnis ω_E/ω_S (Rotverschiebung), der durchschnittliche Fehler $\bar{e}$ und die Standardabweichung σ_e des Fehlers.

Falls die Rotverschiebung nicht berechnet wurde, weil sich die Quelle innerhalb der Ergosphäre befand, wird $\omega_E/\omega_S = -1$ gesetzt. Es sollte angemerkt werden, dass ω_E/ω_S für alle Pixel gespeichert wird, auch wenn der Pixel fehlerhaft ist oder zum Himmel gehört.

Kapitel 7

Ergebnisse

In diesem Kapitel präsentieren und interpretieren wir verschiedene Bilder, welche mit `cuRRay` erzeugt wurden.

Alle Bilder und dazugehörige Szenendateien sind jeweils am Ende des Unterkapitels aufgelistet. Die genauen Parameter der Bilder können in den korrespondierenden Szenendateien nachgelesen werden. Anhang E liefert eine Anleitung, wie `cuRRay` bedient wird und wie die Informationen in den Szenendateien zu verstehen sind.

Alle erzeugten Bilder sowie die verwendeten Szenendateien sind im Git-Repository vorhanden. Siehe Anhang F.

7.1 Schwarzschild-Löcher

Schwarzschild-Löcher sind für die Visualisierung von vielen Raumzeitkrümmungseffekten geeignet und wegen der sphärischen Symmetrie einfach zu interpretieren. Im Folgenden zeigen wir verschiedene Bilder von Schwarzschild-Löchern und zeigen damit die Ablenkung sowie die Rotverschiebung von Licht.

Akkretionsscheibe Die Abbildungen 7.1.1 und 7.1.2 zeigen ein Schwarzschild-Loch mit Akkretionsscheibe (Oberseite Grün, Unterseite Magenta) von zwei verschiedenen Beobachterpositionen. Listing 7.1.1 ist die Szenendatei, welche für beide Bilder verwendet wurde.

In Abbildung 7.1.1 scheint die Scheibe nur sehr wenig verzerrt, weil sich der Beobachter nahe ihrer Achse befindet ($\theta = 5°$) und somit die verschiedenen Teile der Scheibe wegen der sphärischen Symmetrie der Raumzeit achsensymmetrisch verzerrt werden. Nahe des Ereignishorizonts ist die Unterseite der Scheibe sichtbar: Licht von der Unterseite wird durch die gekrümmte Raumzeit um rund 180° gebogen und erreicht den Beobachter durch den Spalt zwischen Scheibe und Horizont.

In Abbildung 7.1.2 ist die Scheibe stark verzerrt. Der Beobachter befindet sich weit von der Symmetrieachse entfernt ($\theta = 85°$). Die Hinterseite der Scheibe, welche in Absenz von Raumzeitkrümmung durch den Ereignis-

horizont verdeckt würde, ist stark vergrössert und über dem Ereignishorizont gebogen sichtbar, weil ihr Licht durch die Raumzeitkrümmung nach unten (zum schwarzen Loch hin) abgelenkt wird. Die regelmässig verteilten Sektoren der Scheibe erscheinen auf der Hinterseite stark verzogen. Die Unterseite der Scheibe ist unter dem Ereignishorizont ebenfalls stark verzerrt sichtbar, weil ihr Licht durch die Raumzeitkrümmung nach oben (zum schwarzen Loch hin) abgelenkt wird.

Sternenhimmel hinter Schwarzschild-Loch Als nächstes untersuchen wir, wie der Sternenhimmel von einem Schwarzschild-Loch verzerrt wird. Wir verwenden als Projektion des Himmels das Milchstrassen-Panorama aus Abbildung 6.10.1. Abbildung 7.1.3 zeigt das berechnete Bild und Listing 7.1.2 ist die dazugehörige Szenendatei.

Die Sterne direkt hinter dem schwarzen Loch werden zu Ringen um das schwarze Loch gebogen. Solche ringförmige Bilder von astronomischen Objekten werden auch *Einstein-Ringe* genannt. Sie wurden bereits mehrmals mit Teleskopen entdeckt und fotografiert. Siehe [SP99.1, Kapitel 2.5.6].

Kugel um Ereignishorizont Abbildung 7.1.4 zeigt ein Schwarzschild-Loch, welches von einer Kugel umschlossen wird. Wir wählen den Radius der Kugel so klein, dass die Kugel möglichst den Ereignishorizont imitiert. Wir können uns so vorstellen, dass das Schachbrettmuster der Kugel auf dem Ereignishorizont selber eingezeichnet ist. Listing 7.1.3 zeigt die verwendete Szenendatei.

Weil Lichtstrahlen gegen das schwarze Loch hin gebogen werden, können wir beide Pole des schwarzen Lochs sehen (Punkte, an denen sich die Dreiecke des Musters treffen), obwohl wir das schwarze Loch vom Äquator aus betrachten.

Rotverschiebung Abbildung 7.1.5 zeigt die Rotverschiebung einer Akkretionsscheibe, welche bis an den Ereignishorizont reicht. Listing 7.1.4 ist die dazugehörige Szenendatei.

Der Ereignishorizont ist komplett rot eingefärbt. Gemäss den Gleichungen (6.9.1) bedeutet dies, dass Licht von dort unendlich rotverschoben ist, beziehungsweise dass gar kein Licht den Beobachter erreicht.

Punkte auf der Scheibe sind umso weniger rotverschoben, je weiter sie vom Ereignishorizont entfernt sind. Dies zeigt sich in der roten Farbe, die kontinuierlich nach aussen hin zu Magenta wird; am äusseren Rand der Scheibe ist fast keine Rotverschiebung feststellbar.

Blauverschiebung Abbildung 7.1.6 zeigt die Blau- und Rotverschiebung, die ein nach aussen gerichteter Beobachter knapp über dem Ereignishorizont in der gleichen Szene wie in Abbildung 7.1.5 sieht. Abbildung 7.1.7 zeigt das dazugehörige Farbbild. Listing 7.1.5 ist die zu den Bildern gehörende Szenendatei.

Wir sehen anhand des Farbbilds, dass der Himmel (weiss) in der Mitte des Bilds erscheint und rund herum vom Ereignishorizont eingefasst wird.

Im Vergleich zu den vorherigen Bildern sind Himmel und Ereignishorizont vertauscht.

Am Rotverschiebungsbild sehen wir, dass Licht vom Ereignishorizont immer noch rotverschoben ist: Der Beobachter befindet sich zwar knapp über dem Horizont, doch die Rotverschiebung zum Horizont ist trotzdem unendlich. Entlang der Scheibe wird das Licht immer mehr blauverschoben, je weiter aussen sich die Quelle befindet.

Kugel in der Nähe eines Schwarzschild-Lochs Die Abbildungen 7.1.8, 7.1.9 und 7.1.10 zeigen ein Schwarzschild-Loch mit einer Kugel in verschiedenen Positionen. Der Beobachter befindet sich in allen drei Bildern an der gleichen Position und betrachtet das schwarze Loch direkt. Die Kugel befindet sich zuerst zwischen Beobachter und schwarzem Loch, dann entlang ϕ um 90° verschoben und schliesslich vom Beobachter aus gesehen hinter dem schwarzen Loch. Listing 7.1.6 zeigt die Szenendatei, welche für die Bilder verwendet wurde.

In Abbildung 7.1.8 befindet sich die Kugel vom Beobachter aus gesehen direkt vor dem schwarzen Loch. Sie erscheint komplett rund. Knapp über dem Horizont ist ein kaum sichtbares, ringförmiges Bild der Kugel zu erkennen, welches aus Lichtstrahlen besteht, die hinter dem schwarzen Loch rund 180° abgelenkt wurden.

In Abbildung 7.1.9 bilden die Kugel, das schwarze Loch und der Beobachter einen rechten Winkel. Die Kugel befindet sich auf der rechten Seite des Ereignishorizonts. Sie erscheint nahezu rund. Auf der linken Seite des Horizonts ist eine kleine Sichel sichtbar, ein Bild der Kugel, welches aus Photonen gebildet wird, welche durch die Raumzeitkrümmung hinter dem schwarzen Loch rund 90° abgelenkt wurden.

In Abbildung 7.1.10 befindet sich die Kugel vom Beobachter aus gesehen direkt hinter dem Schwarzschild-Loch. Sie erscheint stark vergrössert als symmetrischer Ring um den Ereignishorizont.

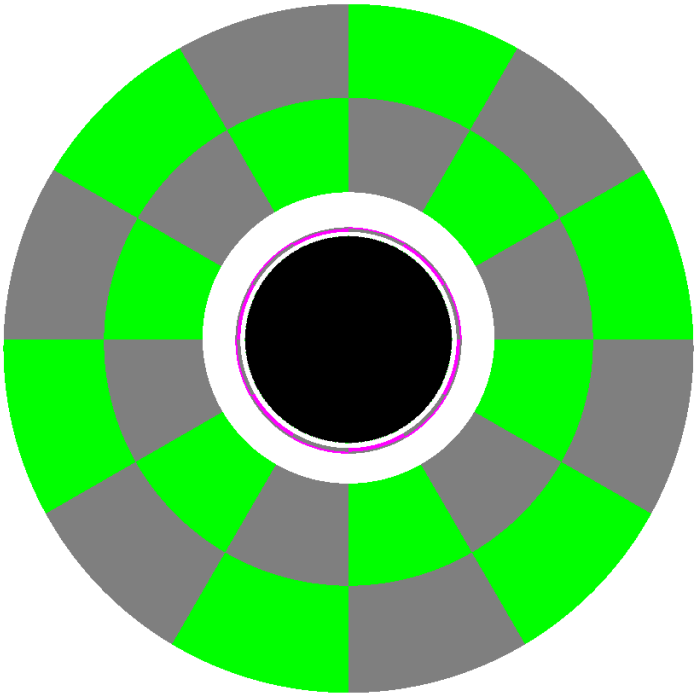


Abbildung 7.1.1: Schwarzschild-Loch mit Akkretionsscheibe. Zenitwinkel des Beobachters: $\theta = 5°$.

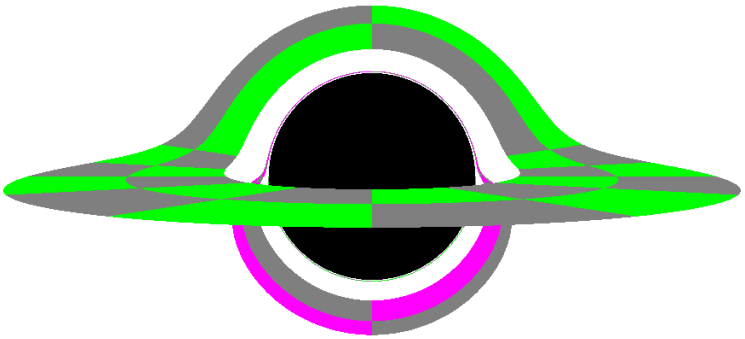


Abbildung 7.1.2: Schwarzschild-Loch mit Akkretionsscheibe. Zenitwinkel des Beobachters: $\theta = 85°$.

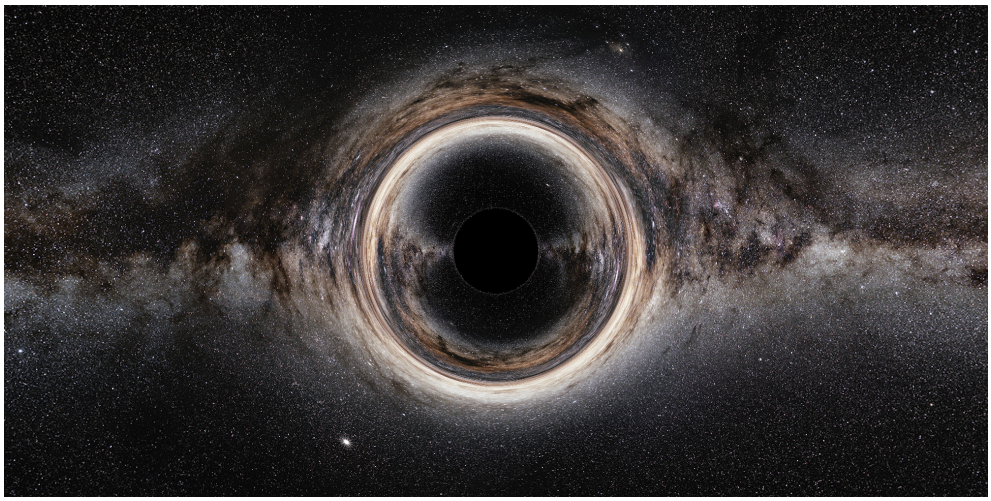


Abbildung 7.1.3: Bild eines Schwarzschild-Lochs mit Sternenhimmel im Hintergrund.

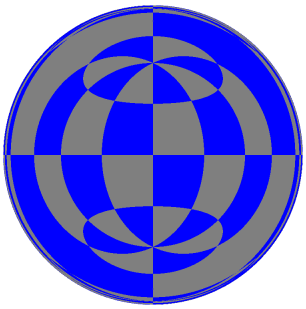


Abbildung 7.1.4: Schwarzschild-Loch mit einer Kugel um den Ereignishorizont.

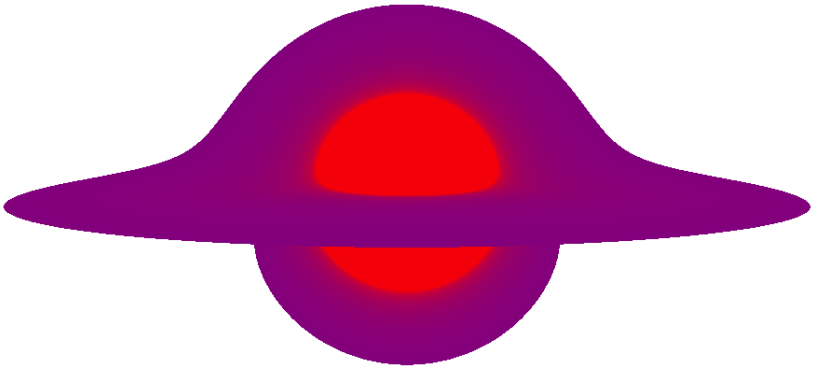


Abbildung 7.1.5: Rotverschiebung von Licht durch ein Schwarzschild-Loch. Die Farbe des Himmels wurde von schwarz auf weiss gesetzt.

Abbildung 7.1.6: Blauverschiebung von Licht durch ein Schwarzschild-Loch. Der Beobachter blickt von oberhalb des Ereignishorizont nach aussen. Die Farbe des Himmels wurde von schwarz auf weiss gesetzt.

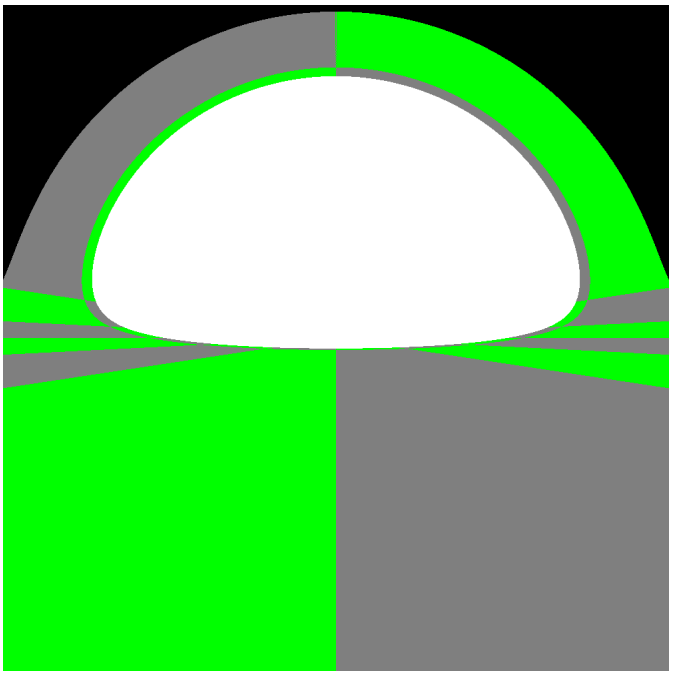


Abbildung 7.1.7: Der Beobachter blickt von oberhalb des Ereignishorizont eines Schwarzschild-Lochs nach aussen.

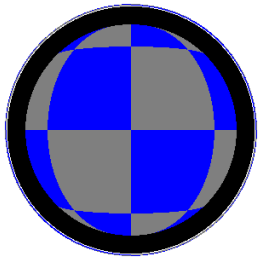


Abbildung 7.1.8: Kugel zwischen einem Schwarzschild-Loch und dem Beobachter.

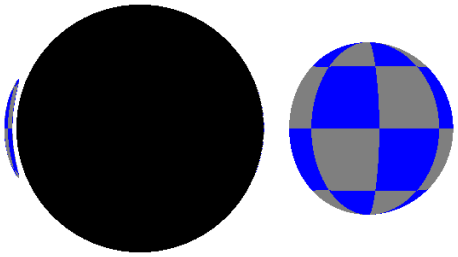


Abbildung 7.1.9: Kugel neben einem Schwarzschild-Loch. Die Kugel ist entlang ϕ um 90° weitergedreht als der Beobachter.

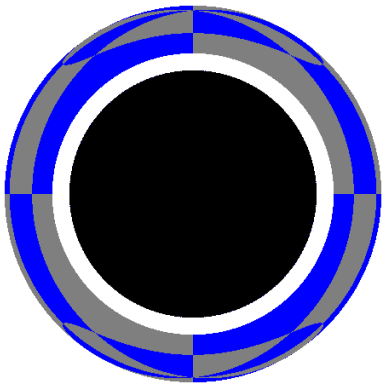


Abbildung 7.1.10: Kugel hinter einem Schwarzschild-Loch.

Listing 7.1.1: Akkretionsscheibe eines Schwarzschild-Lochs

```
---
metric:
    m: 1
    a: 0
    q: 0
sky_color: [255, 255,
   255]
horizon_color: [0, 0, 0]
observer:
    r: 30
    theta: [linear, 5,
   175]
    phi: 0
    hfov: 70
accretion:
    color1: [0, 255, 0]
    resolution: [2, 12]
    radius: [6, 15]
```

Listing 7.1.2: Sternenhimmel hinter Schwarzschild-Loch

```
---
metric:
    m: 1
    a: 0
    q: 0
horizon_color: [0, 0, 0]
observer:
    r: 50
    theta: 90
    phi: 0
    hfov: 100
skymap:
    image: sky.png
    boundary: 50
```

Listing 7.1.3: Schwarzschild-Loch mit Kugel um Ereignishorizont

```
---
metric:
    m: 1
    a: 0
    q: 0
sky_color: [255, 255,
   255]
horizon_color: [0, 0, 0]
observer:
    r: 10
    theta: 90
    phi: 0
    hfov: 70
sphere:
    color: [0, 0, 255]
    resolution: [4, 8]
    r: 0
    theta: 90
    phi: 0
    radius: 2.1
```

Listing 7.1.4: Rotverschiebung um Schwarzschild-Loch

```
---
metric:
    m: 1
    a: 0
    q: 0
sky_color: [255, 255,
   255]
horizon_color: [0, 0, 0]
observer:
    r: 30
    theta: 85
    phi: 0
    hfov: 70
accretion:
    color1: [0, 255, 0]
    resolution: [2, 12]
    radius: [2, 15]
```

Listing 7.1.5: Blauverschiebung in der Nähe eines Schwarzschild-Lochs

```
---
metric:
    m: 1
    a: 0
    q: 0
sky_color: [255, 255,
   255]
horizon_color: [0, 0, 0]
observer:
    r: 2.1
    theta: 85
    phi: 0
    yaw: 180
    hfov: 70
accretion:
    color1: [0, 255, 0]
    resolution: [2, 12]
    radius: [2, 15]
```

Listing 7.1.6: Kugel in der Nähe von Schwarzschild-Loch

```
---
metric:
    m: 1
    a: 0
    q: 0
sky_color: [255, 255,
   255]
horizon_color: [0, 0, 0]
observer:
    r: 25
    theta: 90
    phi: 0
    hfov: 70
sphere:
    color: [0, 0, 255]
    resolution: [4, 8]
    r: 8
    theta: 90
    phi: [linear, 0, 350]
    radius: 3
```

7.2 Kerr-Löcher

In diesem Kapitel vergleichen wir Bilder ungeladener, rotierender schwarzer Löcher mit Bildern von Schwarzschild-Löchern. Ungeladene, rotierende schwarze Löcher heissen *Kerr-Löcher*, weil sie durch die Kerr-Metrik beschrieben werden. Die Kerr-Newman-Metrik wird zur Kerr-Metrik, wenn $Q = 0$, $M \neq 0$ und $a \neq 0$.

Akkretionsscheibe eines Kerr-Lochs Analog zu den Abbildungen 7.1.1 und 7.1.2 eines Schwarzschild-Lochs zeigen die Abbildungen 7.2.1 und 7.2.2 ein Kerr-Loch mit Akkretionsscheibe. Die Szeneninformationen sind die gleichen, ausser dass nun $a \neq 0$. Der Beobachter betrachtet die Szene wieder aus zwei verschiedenen Zenitwinkeln: $\theta = 5°$ und $\theta = 85°$. Listing 7.2.1 ist die dazugehörige Szenendatei.

Abbildung 7.2.1 unterscheidet sich von Abbildung 7.1.1 darin, dass die Scheibe nahe des Horizonts wie durch einen Wirbel verzerrt wird. Dies ist die Auswirkung des Lense-Thirring-Effekts. Der Beobachter blickt ungefähr entlang der negativen z-Achse (Rotationsachse); es ist deshalb zu erwarten, dass ein positiver Rotationsparameter gemäss der rechten-Hand-Regel die Raumzeit im Gegenuhrzeigersinn verdreht. Die Grafik zeigt das Gegenteil: die Pho-

tonen werden im Uhrzeigersinn mitgerissen. Weil die Zeit beim Raytracing umgedreht wird, ändert auch die Drehrichtung des schwarzen Lochs.

In allen von `cuRRay` erstellten Bildern dreht sich das schwarze Loch deshalb gemäss der *linken-Hand-Regel* um die Polachse.

In Abbildung 7.2.2 hat der Beobachter einen Zenitwinkel $\theta = 85°$. Der Lense-Thirring-Effekt ist an den Sektoren der Scheibe zu erkennen: die Sektorgrenzen befinden sich zum Beispiel im Vergleich zu Abbildung 7.1.2 nicht in der Mitte des Bildes, weil ihr Licht durch den Effekt abgelenkt wurde.

Ausserdem scheint der Ereignishorizont auf einer Seite ausgedehnter als auf der anderen. Dies kommt daher, dass Photonen in der Ergosphäre nur in Richtung der Rotation am Kerr-Loch vorbeikommen. Zudem kommen Photonen, welche sich in die Richtung des rotierenden Flusses der Raumzeit bewegen, in weniger Koordinatenzeit am schwarzen Loch vorbei als solche, welche sich gegen den Fluss bewegen. Da die radiale Ablenkung für alle gleich ist, fallen Photonen, welche sich gegen den rotierenden Fluss bewegen, schneller ins Kerr-Loch, als solche, welche sich mit dem Fluss fortbewegen. Dies führt zu einem asymmetrischen Bild.

Eine weitere Konsequenz der Rotation ist, dass die Photonensphäre nicht symmetrisch ist und vom Blickwinkel abhängt. Siehe [CC13.1] für eine Ausführliche Diskussion von Photonensphären rotierender schwarzer Löcher.

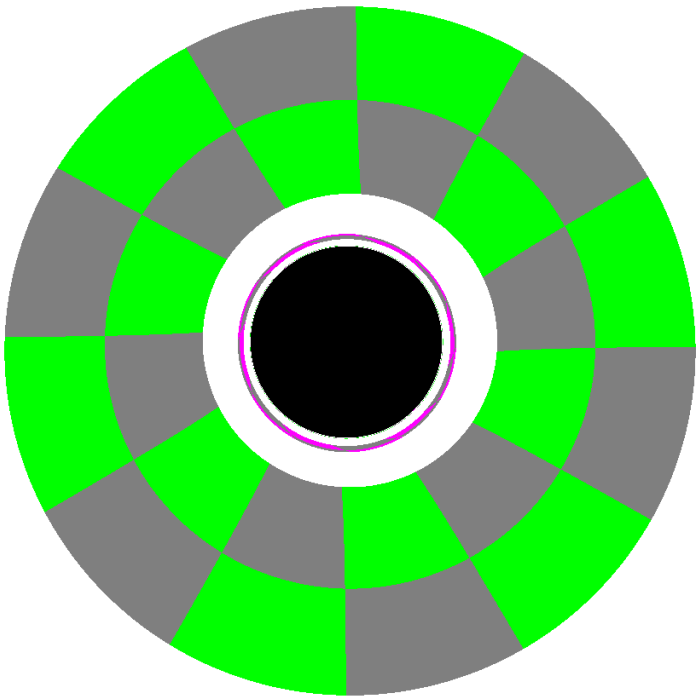


Abbildung 7.2.1: Kerr-Loch mit Akkretionsscheibe. Zenitwinkel des Beobachters: $\theta = 5°$.

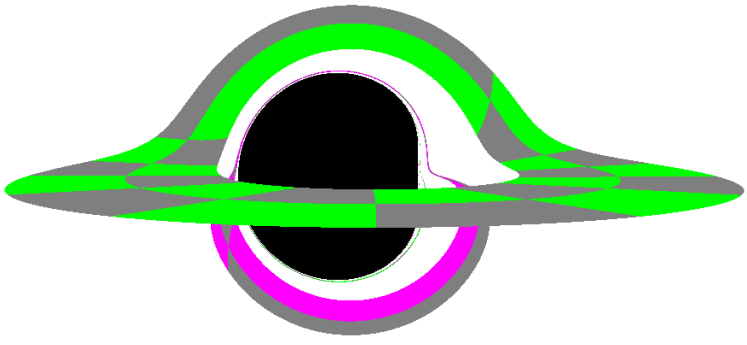


Abbildung 7.2.2: Kerr-Loch mit Akkretionsscheibe. Zenitwinkel des Beobachters: $\theta = 85°$.

Listing 7.2.1: Akkretionsscheibe eines Kerr-Lochs

```
---
metric:
    m: 1
    a: 1
    q: 0
sky_color: [255, 255, 255]
horizon_color: [0, 0, 0]
observer:
    r: 30
    theta: [linear, 5, 175]
    phi: 0
    hfov: 70
accretion:
    color1: [0, 255, 0]
    resolution: [2, 12]
    radius: [6, 15]
```

7.3 Reissner-Nordström-Löcher

Zum Schluss vergleichen wir ein *Reissner-Nordström-Loch* ($a = 0$, $Q \neq 0$) mit einem Schwarzschild-Loch.

Reissner-Nordström-Löcher unterscheiden sich äusserlich nicht besonders stark von Schwarzschild-Löchern. Der Unterschied liegt vor allem innerhalb des Ereignishorizonts: Reissner-Nordström-Löcher können genau wie die allgemeinen Kerr-Newman-Löcher zwei Horizonte haben. Wir behandeln die innere Struktur schwarzer Löcher jedoch nicht, da `cuRRay` die Integration der Geodäten am äusseren Ereignishorizont stoppt.

Kugel hinter Reissner-Nordström-Loch Eine von aussen sichtbare Eigenschaft Reissner-Nordström-Löcher ist, dass der Ereignishorizont kleiner als derjenige eines Schwarzschild-Lochs gleicher Masse ist. Dies folgt direkt aus Gleichung (2.2.5).

Abbildung 7.3.1 zeigt die gleiche Szene wie Abbildung 7.1.10 mit dem Unterschied, dass $Q \neq 0$. Listing 7.3.1 ist die dazugehörige Szenendatei. Der Ereignishorizont erscheint kleiner als in Abbildung 7.1.10.

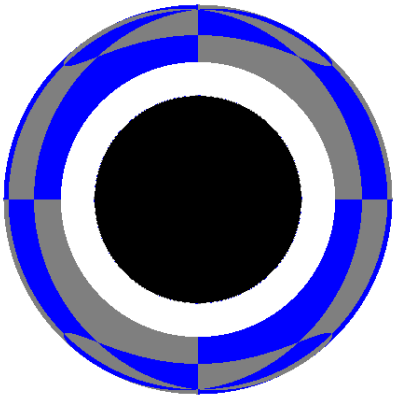


Abbildung 7.3.1: Kugel hinter Reissner-Nordström-Loch.

Listing 7.3.1: Kugel hinter Reissner-Nordström-Loch

```
---
metric:
    m: 1
    a: 0
    q: 1
sky_color: [255, 255, 255]
horizon_color: [0, 0, 0]
observer:
    r: 25
    theta: 90
    phi: 0
    hfov: 70
sphere:
    color: [0, 0, 255]
    resolution: [4, 8]
    r: 8
    theta: 90
    phi: [linear, 0, 350]
    radius: 3
```

Kapitel 8

Diskussion

Hier besprechen wir verschiedene Aspekte der Software `cuRRay` und erwähnen mögliche Verbesserungen. Wir beginnen mit einer Diskussion der Zuverlässigkeit und untersuchen mögliche Programmfehler. Anschliessend betrachten wir kurz die Effizienz der Software. Im Ausblick werden mögliche Erweiterungen diskutiert. Zum Schluss erwähnen wir `GRay`, die Software, von welcher `cuRRay` die Schrittweitensteuerung verwendet.

8.1 Zuverlässigkeit von `cuRRay`

`cuRRay` wurde während der Entwicklung mehrere Stunden getestet. Alle so erkannten Programmierfehler (u.a. falsche Berechnung von Christoffelsymbolen) wurden behoben.

Ausserdem wurden gezielt Fehler ausgeschlossen, indem die Algorithmen auf verschiedene Arten getestet wurden. Zum Beispiel sollte die Parametrisierung der Geodäten verändert werden können, ohne dass sich dadurch die berechneten Bilder verändern; gerade mit diesem Test konnten falsche Bewegungsgleichungen korrigiert werden.

Einige Fehler sind bekannt, wurden aber bewusst nicht entfernt, da sie seltenen Spezialfällen entsprechen und ihre Korrektur keinen grossen Mehrwert erzeugt. Folgende Arten von Fehler können immer noch auftreten:

Betriebssystembedingte Fehler Fehler, wie z.B. mangelnder Arbeits- oder Grafikspeicher und Hardwarefehler, können von `cuRRay` nicht behoben werden und führen meistens zu einer frühzeitigen Beendung der Software. Diese Fehler können durch Einhalten der Systemanforderungen aus Kapitel 4.2 vermieden werden.

Unter Windows kann es zudem aufgrund des TDR-Timers zu Abstürzen kommen. Siehe dazu Anhang E.5.

Fehler aufgrund von Benutzereingaben `cuRRay` filtert nicht alle fehlerhaften Benutzereingaben heraus.

Zum Beispiel können in der Szenendatei unmögliche Einstellungen gewählt werden (z.B. $M^2 < a^2 + Q^2$ oder Beobachter innerhalb der Ergosphäre). Es ist

dem Benutzer überlassen, die Korrektheit der Einstellungen zu überwachen. Unmögliche Einstellungen führen unter Umständen zu undefiniertem Verhalten.

Numerische Ungenauigkeit von RK4 Die numerische Ungenauigkeit des RK4-Algorithmus kann nur schwer bestimmt werden, da die exakten Lösungen der Geodätengleichungen nicht bekannt sind. Der RK4-Algorithmus wird aber umso genauer, je kleiner die Schrittweite gewählt wird.

Gleichung (6.6.1) steuert die Schrittweite, die `cuRRay` verwendet, indem sie kleiner gewählt wird, wenn die Geschwindigkeitskomponenten grösser werden. Der Faktor c ermöglicht eine manuelle Anpassung der Schrittweite. Folgendermassen kann sichergestellt werden, dass die Schrittweiten genügend klein sind: Mehrere Bilder der gleichen Szene werden für immer kleinere Konstanten c erstellt. Sobald sich die Bilder nicht mehr unterscheiden, kann davon ausgegangen werden, dass c klein genug ist und die Schrittweitensteuerung genügend genau ist. Fehler aufgrund von ungenauen Berechnungen können so minimiert werden.

Auf diese Art wurde der Standardwert $c = 1/32$ empirisch bestimmt. Auch Extremsituationen, wie Beobachter, die sich knapp über dem Ereignishorizont befinden, wurden damit erfolgreich getestet.

Koordinatensingularitäten Die Polachse ($\theta = 0$, bzw. $\theta = \pi$) ist eine Koordinatensingularität, da die θ- und ϕ-Koordinaten dort diskontinuierlich sind. RK4-Schritte werden nur in kontinuierlichen Koordinatensystemen korrekt berechnet; deshalb entstehen Fehler, wenn in einem Schritt die Koordinatensingularität überschritten wird.

Solche Fehler zeigen sich, indem die Koordinaten des Photons ungültig werden ($r < 0$, $\theta < 0$ oder $\theta > \pi$). `cuRRay` erkennt diese Fehler und markiert die betroffenen Pixel in einer konfigurierbaren Fehlerfarbe. Fehler entstehen nur, wenn Geodäten die Polachse genau treffen. Deshalb können sie leicht verhindert werden, indem entweder gerade Frame-Abmessungen gewählt werden, oder der Beobachter leicht abgedreht wird. Ausserdem sollte der Beobachter selber nicht genau auf der Polachse positioniert werden. Abbildung 8.1.1 zeigt ein Schwarzschild-Loch mit bewusst erzeugten Fehlern entlang der Polachse.

8.2 Effizienz von `cuRRay`

`cuRRay`'s Leistung wurde mit keiner bereits existierenden Software verglichen.

Vergleiche zwischen CUDA-Version und CPU-Version konnten zeigen, dass die CUDA-Version generell schneller als die CPU-Version ist. Auf dem Entwicklungscomputer (Intel Core i7-4770 @ 3.4 GHz, ASUS GTX-760 DirectUI II OC) ist die CUDA-Version mindestens doppelt so schnell wie die CPU-Version. Auf einem zweiten Testcomputer (Intel Core i7-6700K @ 4.0 GHz, EVGA GTX-1070 Founder's Edition) war die CPU-Version nur rund 10% schneller als auf dem ersten Computer, die CUDA-Version aber doppelt so

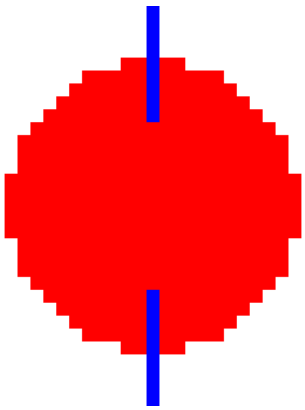


Abbildung 8.1.1: Schwarzschild-Loch mit Fehlern entlang der Achse. Die Abmessungen des Bildes sind ungerade (31x31 Pixel), damit die horizontal zentrierten Geodäten genau auf die Koordinatensingularität der Polachse treffen. Damit die fehlerhaften Pixel gut sichtbar sind, wurden kleine Frame-Abmessungen verwendet und das schwarze Loch rot eingefärbt.

schnell. Anhand der Tests können wir die Überlegenheit der CUDA-Version über die CPU-Version klar feststellen.

Im Verlauf der Entwicklung konnte die Leistung der Software stark verbessert werden. `cuRRay` wurde seit dem ersten Prototypen schätzungsweise 200 Mal schneller. Trotz all dieser Optimierungen könnte vor allem die CUDA-Version verbessert werden: Untersuchungen mit dem NVIDIA Visual Profiler [1] zeigten, dass die Grafikkarte rund die Hälfte der Zeit auf Daten wartet, die entweder in oder aus der VRAM geladen werden. Indem in `cuRRay` verschiedene Arten von VRAM effizienter benutzt werden, könnte die Leistung noch weiter gesteigert werden.

8.3 Ausblick

`cuRRay` eignet sich zur Visualisierung der Kerr-Newman-Raumzeit und könnte als Unterrichtsmittel für Physikstudenten eingesetzt werden. Weitere Funktionen könnten die Visualisierung nach facettenreicher gestalten:

Eine nützliche Funktion wäre, die einzelnen Schritte einer Geodäte zu speichern, so dass sie später in einem dreidimensionalen Graph eingetragen werden können. Die Flugbahn eines einzelnen Photons wäre somit sofort klar ersichtlich. Ausserdem müsste Raytracing nicht nur auf Photonen beschränkt sein; Flugbahnen von beliebigen Teilchen könnten durch Lösen der Geodätengleichungen berechnet werden. So könnten zum Beispiel Orbits kleiner Asteroiden berechnet werden.

Neben der Kerr-Newman-Metrik könnten andere Raumzeiten visualisiert werden. Der RK4-Algorithmus ermöglicht Raytracing prinzipiell in allen denkbaren Raumzeiten. Zum Beispiel könnten *Kruskal-Szekeres-Koordinaten* ver-

[1] Siehe `https://developer.nvidia.com/nvidia-visual-profiler`.

wendet werden, um Raytracing *innerhalb* eines schwarzen Lochs durchzuführen.

8.4 Bestehende Software

Obwohl `cuRRay` nicht mit bestehender Software verglichen wurde, soll hier trotzdem eine Software erwähnt werden, welche ähnliche Funktionen wie `cuRRay` besitzt: `GRay` [CC13.1] ist ein CUDA-beschleunigter Raytracer für relativistische Raumzeiten um schwarze Löcher. `GRay` verwendet dazu ebenfalls den RK4-Algorithmus. `cuRRay` verwendet die Schrittweitensteuerung von `GRay`.

Anhang A

Geodätengleichungen erster Ordnung

In diesem Anhang vereinfachen wir die Geodätengleichungen für t und ϕ mit Hilfe der Killing-Vektorfelder der Kerr-Newman-Metrik zu Differentialgleichungen erster Ordnung.

Die Killing-Vektorfelder der Kerr-Newman-Metrik liefern als Erhaltungsgrössen

$$E = g_{tt}u^t + g_{t\phi}u^\phi \tag{A.1}$$

und

$$L = g_{\phi\phi}u^\phi + g_{t\phi}u^t. \tag{A.2}$$

Siehe Kapitel 2.3.1. Die beiden Gleichungen (A.1) und (A.2) enthalten die Geschwindigkeitskomponenten $u^t = dt/d\lambda$ und $u^\phi = d\phi/d\lambda$. Wir können nach diesen Komponenten auflösen und somit Differentialgleichungen erster Ordnung für t und ϕ erhalten.

Wir lösen zuerst Gleichung (A.2) nach $d\phi/d\lambda$ auf:

$$\frac{d\phi}{d\lambda} = \frac{L - g_{t\phi}dt/d\lambda}{g_{\phi\phi}} \tag{A.3}$$

und setzen in (A.1) ein:

$$E = -g_{tt}\frac{dt}{d\lambda} - g_{t\phi}(L - g_{t\phi}\frac{dt}{\lambda})/g_{\phi\phi}. \tag{A.4}$$

Wir lösen nun nach $dt/d\lambda$ auf:

$$\frac{dt}{d\lambda} = (E + L\frac{g_{t\phi}}{g_{\phi\phi}})/(\frac{{g_{t\phi}}^2}{g_{\phi\phi}} - g_{tt}). \tag{A.5}$$

Schliesslich multiplizieren wir mit $g_{\phi\phi}$ und erhalten die Differentialgleichung erster Ordnung, welche die Entwicklung der t-Koordinate bestimmt:

$$\frac{dt}{d\lambda} = \frac{E \cdot g_{\phi\phi} - L \cdot g_{t\phi}}{g_{\phi\phi}g_{tt} - {g_{t\phi}}^2}. \tag{A.6}$$

Auf ähnliche Art und Weise erhalten wir die Gleichung für die Entwicklung von ϕ:

$$\frac{d\phi}{d\lambda} = \frac{L \cdot g_{tt} - E \cdot g_{t\phi}}{g_{tt} g_{\phi\phi} - {g_{t\phi}}^2}. \tag{A.7}$$

Anhang B

Christoffelsymbole in Kerr-Newman-Raumzeit

In diesem Anhang listen wir die für die Gleichungen (2.3.10) und (2.3.11) notwendigen Christoffelsymbole der Kerr-Newman-Raumzeit auf. Zuerst zeigen wir, wie für die Berechnung vorgegangen werden muss, anschliessend leiten wir ein Christoffelsymbol exemplarisch her, bevor wir alle relevanten Christoffelsymbole auflisten.

Relevante Christoffelsymbole Die Christoffelsymbole müssen anhand von Gleichung (1.7.9) berechnet werden:

$$\Gamma^{\alpha}{}_{\beta\gamma} = g^{\alpha\delta}\frac{1}{2}(\partial_\gamma g_{\delta\beta} + \partial_\beta g_{\delta\gamma} - \partial_\delta g_{\beta\gamma}). \tag{B.1}$$

Bevor wir die Komponenten der Metrik invertieren oder ableiten, halten wir fest, dass nur die Christoffelsymbole für $\alpha = \{r, \theta\}$ benötigt werden. Dies wird unsere Arbeit um einiges erleichtern.

Inverse Metrik In Gleichung B.1 benötigen wir die Komponenten des inversen metrischen Tensors. Sie sind durch Gleichung (1.6.5) gegeben. Betrachten wir die Komponenten des metrischen Tensors als Einträge einer 4x4-Matrix, sind die Komponenten des inversen metrischen Tensors die Einträge der inversen 4x4-Matrix.

Die Metrik der Kerr-Newman-Raumzeit hat gemäss Gleichung (2.2.1) folgende Form:

$$g_{\alpha\beta} = \begin{pmatrix} g_{tt} & 0 & 0 & g_{t\phi} \\ 0 & g_{rr} & 0 & 0 \\ 0 & 0 & g_{\theta\theta} & 0 \\ g_{t\phi} & 0 & 0 & g_{\phi\phi} \end{pmatrix}. \tag{B.2}$$

Die inverse Metrik ist demnach:

$$g^{\alpha\beta} = \begin{pmatrix} -\frac{g_{\phi\phi}}{{g_{t\phi}}^2 - g_{tt}g_{\phi\phi}} & 0 & 0 & \frac{g_{t\phi}}{{g_{t\phi}}^2 - g_{tt}g_{\phi\phi}} \\ 0 & \frac{1}{g_{rr}} & 0 & 0 \\ 0 & 0 & \frac{1}{g_{\theta\theta}} & 0 \\ \frac{g_{t\phi}}{{g_{t\phi}}^2 - g_{tt}g_{\phi\phi}} & 0 & 0 & -\frac{g_{tt}}{{g_{t\phi}}^2 - g_{tt}g_{\phi\phi}} \end{pmatrix}. \tag{B.3}$$

Wir werden nur die inversen metrischen Komponenten benötigen, für die $\alpha = \{r, \theta\}$ gilt. Das sind

$$g^{rr} = \frac{1}{g_{rr}} = \frac{\Delta}{\Sigma}, \tag{B.4}$$

$$g^{\theta\theta} = \frac{1}{g_{\theta\theta}} = \frac{1}{\Sigma}. \tag{B.5}$$

Da wir nur Komponenten $g^{\alpha\beta}$ verwenden, für die $\alpha = \beta$ gilt, muss in Gleichung (B.1) nur jeweils ein Term pro Christoffelsymbol aufsummiert werden.

Ableitungen der Komponenten der Metrik Der grösste Aufwand ist die Berechnung der Ableitungen der metrischen Komponenten. Doch auch hier kann ein Teil der Arbeit gespart werden, indem wir beachten, dass die Metrik von t und ϕ unabhängig ist, das heisst $\partial_t g_{\alpha\beta} = \partial_\phi g_{\alpha\beta} = 0$. Wir können die Ausdrücke für die einzelnen Christoffelsymbole vereinfachen, indem wir Terme, die dadurch null werden, herausstreichen:

$$\begin{aligned}
{\Gamma^r}_{tt} &= -\frac{1}{2}g^{rr}\partial_r g_{tt} & {\Gamma^\theta}_{tt} &= -\frac{1}{2}g^{\theta\theta}\partial_\theta g_{tt} \\
{\Gamma^r}_{rr} &= \frac{1}{2}g^{rr}\partial_r g_{rr} & {\Gamma^\theta}_{rr} &= -\frac{1}{2}g^{\theta\theta}\partial_\theta g_{rr} \\
{\Gamma^r}_{\theta\theta} &= -\frac{1}{2}g^{rr}\partial_r g_{\theta\theta} & {\Gamma^\theta}_{\theta\theta} &= \frac{1}{2}g^{\theta\theta}\partial_\theta g_{\theta\theta} \\
{\Gamma^r}_{\phi\phi} &= -\frac{1}{2}g^{rr}\partial_r g_{\phi\phi} & {\Gamma^\theta}_{\phi\phi} &= -\frac{1}{2}g^{\theta\theta}\partial_\theta g_{\phi\phi} \\
{\Gamma^r}_{tr} &= 0 & {\Gamma^\theta}_{tr} &= 0 \\
{\Gamma^r}_{t\theta} &= 0 & {\Gamma^\theta}_{t\theta} &= 0 \\
{\Gamma^r}_{t\phi} &= -\frac{1}{2}g^{rr}\partial_r g_{t\phi} & {\Gamma^\theta}_{t\phi} &= -\frac{1}{2}g^{\theta\theta}\partial_\theta g_{t\phi} \\
{\Gamma^r}_{r\theta} &= \frac{1}{2}g^{rr}\partial_\theta g_{rr} & {\Gamma^\theta}_{r\theta} &= \frac{1}{2}g^{\theta\theta}\partial_r g_{\theta\theta} \\
{\Gamma^r}_{r\phi} &= 0 & {\Gamma^\theta}_{r\phi} &= 0 \\
{\Gamma^r}_{\theta\phi} &= 0 & {\Gamma^\theta}_{\theta\phi} &= 0.
\end{aligned} \tag{B.6}$$

Im nächsten Schritt müssen die Ausdrücke für die Christoffelsymbole evaluiert werden. Exemplarisch berechnen wir ${\Gamma^r}_{tt}$; die anderen Christoffelsymbole werden auf ähnliche Weise berechnet.

Beispiel Für ${\Gamma^r}_{tt}$ erhalten wir aus (B.4), (B.6) und (2.2.1):

$$ {\Gamma^r}_{tt} = -\frac{1}{2}\frac{\Delta}{\Sigma}\frac{d}{dr}\left[-\frac{\Delta - a^2\ \sin^2\theta}{\Sigma}\right]. \tag{B.7} $$

Wir wenden die Quotientenregel für Ableitungen an und vereinfachen:

$$ \begin{aligned} {\Gamma^r}_{tt} &= \frac{1}{2}\frac{\Delta}{\Sigma}\left[\frac{d/dr(\Delta)\cdot\Sigma - (\Delta - a^2\ \sin^2\theta)\cdot d/dr(\Sigma)}{\Sigma^2}\right] \\ &= \frac{1}{2}\frac{\Delta}{\Sigma^3}\left[(2r-2M)\cdot\Sigma - (\Delta - a^2\ \sin^2\theta)\cdot 2r\right] \\ &= \frac{\Delta}{\Sigma^3}\left[Mr^2 - Q^2r - Ma^2\ \cos^2\theta\right]. \end{aligned} \tag{B.8} $$

Liste der Christoffelsymbole Die Christoffelsymbole, welche für die Integration der Geodäten benötigt werden, sind

$$ {\Gamma^r}_{tt} = \frac{\Delta}{\Sigma^3}\left[Mr^2 - Q^2r - Ma^2\cos^2\theta\right] \tag{B.9} $$

$$ {\Gamma^r}_{rr} = \frac{1}{\Delta\Sigma}\left[-Mr^2 + Q^2r + a^2r + (M-r)a^2\cos^2\theta\right] \tag{B.10} $$

$$ {\Gamma^r}_{\theta\theta} = -\frac{\Delta}{\Sigma}r \tag{B.11} $$

$$ \begin{aligned} {\Gamma^r}_{\phi\phi} =& \frac{\Delta}{\Sigma^3}\sin^2\theta\left[\Sigma\left((r-M)a^2\sin^2\theta - 2r(r^2+a^2)\right)\right. \\ &\left. + r\left((r^2+a^2)^2 - \Delta a^2\sin^2\theta\right)\right] \end{aligned} \tag{B.12} $$

$$ {\Gamma^r}_{t\phi} = \frac{\Delta}{\Sigma^3}a\cdot\sin^2\theta\left[-Mr^2 + Q^2r + Ma^2\cos^2\theta\right] \tag{B.13} $$

$$ {\Gamma^r}_{r\theta} = -\frac{1}{\Sigma}a^2\ \cos\theta\sin\theta \tag{B.14} $$

$$ {\Gamma^\theta}_{tt} = \frac{1}{\Sigma^3}a^2\sin\theta\cos\theta\left[Q^2 - 2Mr\right] \tag{B.15} $$

$$ {\Gamma^\theta}_{rr} = \frac{1}{\Sigma\Delta}a^2\sin\theta\cos\theta \tag{B.16} $$

$$ {\Gamma^\theta}_{\theta\theta} = -\frac{1}{\Sigma}a^2\sin\theta\cos\theta \tag{B.17} $$

$$ {\Gamma^\theta}_{\phi\phi} = \frac{1}{\Sigma^3}\sin\theta\cos\theta\left[(r^2+a^2)(\Delta a^2\sin^2\theta - (r^2+a^2)^2) + \Sigma\Delta a^2\sin^2\theta\right] \tag{B.18} $$

$$ {\Gamma^\theta}_{t\phi} = \frac{1}{\Sigma^3}a\cdot\sin\theta\cos\theta(2Mr - Q^2)(r^2+a^2) \tag{B.19} $$

$$ {\Gamma^\theta}_{r\theta} = \frac{1}{\Sigma}r. \tag{B.20} $$

Die Christoffelsymbole wurden von Hand berechnet und mit dem Computeralgebrasystem `wxMaxima` [1] überprüft.

[1] `andrejv.github.io/wxmaxima/`

Anhang C

Gravitative Rotverschiebung in Kerr-Newman-Raumzeit

In diesem Anhang leiten wir Gleichung (3.5.1) für die gravitative Rotverschiebung in Kerr-Newman-Raumzeit her. Wir berechnen dazu das Verhältnis der Frequenzen jedes Lichtstrahls zwischen Sender und Empfänger. Unter Sender verstehen wir die Quelle das Lichtstrahls und unter Empfänger den Beobachter.

Kreisfrequenz Die Kreisfrequenz eines Photons für einen Beobachter mit Geschwindigkeit v^α ist laut Gleichung (3.2.1)

$$\omega = -k_\alpha v^\alpha. \tag{C.1}$$

Wie wir in Kapitel 3.2 sahen, sind der Wellenvektor k^α und die Geschwindigkeit u^α des Photons zueinander proportional: $k^\alpha = a \cdot u^\alpha$, wobei a der Proportionalitätsfaktor ist. Wir erhalten

$$\omega = -a \cdot g_{\alpha\beta} u^\alpha v^\beta. \tag{C.2}$$

Es sollte angemerkt werden, dass u^α und v^β gleich parametrisiert sein müssen:

$$u^\alpha = \frac{dx^\alpha}{d\lambda}, \qquad v^\beta = \frac{dy^\beta}{d\lambda}, \tag{C.3}$$

wobei y^β die Position des Beobachters ist.

Beobachter Um Rotverschiebung aufgrund des speziell relativistischen Dopplereffekts auszuschliessen, dürfen sich Sender und Empfänger nicht relativ zueinander bewegen. Diese Bedingung ist insbesondere erfüllt, wenn beide stationär sind. Wir setzen deshalb für die Geschwindigkeiten v^α der beiden Beobachter $v^0 \neq 0$ und $v^a = 0$.

Diese Entscheidung vereinfacht die Berechnungen. Jedoch kann so die Rotverschiebung von Quellen unterhalb der statischen Grenze nicht berechnet werden, weil diese nie stationär sein können.

Die Frequenz wird zu

$$\omega = -a \cdot \left(g_{tt} u^t v^t + g_{t\phi} u^\phi v^t \right). \tag{C.4}$$

Wir haben

$$\boldsymbol{v} \cdot \boldsymbol{v} = |\boldsymbol{v}|^2 = g_{\alpha\beta} v^\alpha v^\beta = g_{tt} v^t v^t, \tag{C.5}$$

beziehungsweise

$$v^t = \sqrt{\frac{|\boldsymbol{v}|^2}{g_{tt}}} = \sqrt{-|\boldsymbol{v}|^2} \frac{1}{\sqrt{-g_{tt}}}. \tag{C.6}$$

Im letzten Schritt beachteten wir, dass $|\boldsymbol{v}|^2$ und g_{tt} stets negativ sind [1].

Rotverschiebung Das Verhältnis beider Frequenzen ist

$$\frac{\omega_E}{\omega_S} = \frac{-a \cdot \sqrt{-|\boldsymbol{v}|^2} \left(u^t \frac{g_{tt}}{\sqrt{-g_{tt}}} + u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}} \right)\Big|_E}{-a \cdot \sqrt{-|\boldsymbol{v}|^2} \left(u^t \frac{g_{tt}}{\sqrt{-g_{tt}}} + u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}} \right)\Big|_S}. \tag{C.7}$$

Wir bemerken, dass a überall und somit für beide Beobachter gleich ist. Weil u^α und v^α gleich parametrisiert sind und weil die Parametrisierung von u^α entlang des Lichtstrahls nicht ändert, ist v^α bei beiden Beobachtern gleich parametrisiert. Wir wissen (siehe [WR84.1, S. 61]), dass $|\boldsymbol{v}|^2$ für zeitartige Geodäten nur von der Parametrisierung abhängt (insbesondere $|\boldsymbol{v}|^2 = -1$, wenn mit der Eigenzeit des Teilchens parametrisiert wird). $|\boldsymbol{v}|^2$ ist deshalb ebenfalls für beide Beobachter gleich. Wir vereinfachen und erhalten die Gleichung für die gravitative Rotverschiebung von Quellen ausserhalb der statischen Grenze:

$$\frac{\omega_E}{\omega_S} = \frac{u^t \sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\Big|_E}{u^t \sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\Big|_S}. \tag{C.8}$$

Der Zähler der rechten Seite wird an der Position des Empfängers ausgewertet, der Nenner an der Position des Senders. Die Werte für u^α erhalten wir durch Raytracing, g_{tt} und $g_{t\phi}$ können mittels Gleichung 2.2.1 berechnet werden.

[1] g_{tt} wird innerhalb der Ergosphäre positiv, aber wir betrachten diese Fälle nicht.

Anhang D

Herleitung der Kartesisch-BL-Koordinatentransformation

In diesem Anhang leiten wir die Koordinatentransformation (6.4.1) her, welche Vektoren von lokalen kartesischen Koordinaten im Inertialsystem des Beobachters in BL-Koordinaten umwandelt:

$$v^{\alpha}_{\mathrm{BL}} = \Psi^{\alpha}{}_{\beta} v^{\beta}_{\mathrm{k}}. \tag{D.1}$$

Transformation der Metrik Wir wissen, dass die gesuchte Transformation beliebige Tensoren gemäss Gleichung (1.5.1) umrechnet. Sie muss deshalb auch die Metrik der kartesischen Koordinaten in die Metrik der BL-Koordinaten umrechnen. Die Metrik der BL-Koordinaten ist die Kerr-Newman-Metrik ausgewertet an der Position des Beobachters. Wir haben folglich

$$\eta_{\alpha\beta} = \Psi^{\gamma}{}_{\alpha} \Psi^{\delta}{}_{\beta} \hat{g}_{\gamma\delta}, \tag{D.2}$$

wobei $\hat{g}_{\gamma\delta}$ die Komponenten der Kerr-Newman-Metrik an der Position des Beobachters und $\eta_{\alpha\beta}$ die Komponenten der Minkowski-Metrik (flache Raumzeit in kartesischen Koordinaten) sind. Es soll angemerkt werden, dass $\Psi^{\alpha}{}_{\beta}$ hier umgekehrt angewandt wird, weil die Komponenten der Metrik im Gegensatz zu den Komponenten von Vektoren kovariant sind.

Transformationsmatrix Um die Koordinatentransformation besser zu verstehen, stellen wir uns die Komponenten der beiden metrischen Tensoren als Matrizen vor. Die Koordinatentransformation wird dann zu einer Transformationsmatrix. Matrizen sind Rang-(1,1)-Tensoren, metrische Tensoren hingegen Rang-(2,0)-Tensoren. Aus diesem Grund müssen wir vorsichtig sein, wenn wir die Gleichungen (D.1) und (D.2) in Matrix-Schreibweise umwandeln.

Zuerst definieren wir die Matrizen für die metrischen Tensoren. Die Zeile der Matrix wird durch den ersten Index, die Spalte durch den zweiten Index gegeben:

$$\eta_{\alpha\beta} \mathrel{\hat{=}} \eta = \begin{matrix} \\ \alpha \\ \downarrow \end{matrix} \begin{matrix} \beta \rightarrow \\ \begin{pmatrix} \cdots \end{pmatrix} \end{matrix}. \tag{D.3}$$

Die Komponenten der Matrix sind einfach die Komponenten der Metrik. Gleiches gilt für die Matrix $\hat{g}$, welche $\hat{g}_{\alpha\beta}$ entspricht.

Gleichung (D.1) wird nun zu

$$\vec{v}_{\mathrm{BL}} = \Psi \cdot \vec{v}_{\mathrm{k}} \tag{D.4}$$

und Gleichung (D.2) zu

$$\eta = \Psi^T \cdot \hat{g} \cdot \Psi. \tag{D.5}$$

Ψ ist die Transformationsmatrix:

$$\Psi = \Psi^{\alpha}{}_{\beta}; \tag{D.6}$$

Der obere (kontravariante) Index bezeichnet die Zeile, der untere (kovariante) Index die Spalte.

Metrik-Matrizen Betrachten wir die Form der Metrik-Matrizen:

$$\eta = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad \hat{g} = \begin{pmatrix} \hat{g}_{tt} & 0 & 0 & \hat{g}_{t\phi} \\ 0 & \hat{g}_{rr} & 0 & 0 \\ 0 & 0 & \hat{g}_{\theta\theta} & 0 \\ \hat{g}_{t\phi} & 0 & 0 & \hat{g}_{\phi\phi} \end{pmatrix}. \tag{D.7}$$

Wir bemerken, dass wir $\hat{g}$ mittels einer Diagonalisierung und anschliessender Skalierung in η umformen können.

Diagonalisierung Wir wissen aus der linearen Algebra, dass wir eine quadratische Matrix A *diagonalisieren*, das heisst in eine *Diagonalmatrix* D umformen können, deren Komponenten überall ausser auf der Hauptdiagonale null sind:

$$D = P^{-1} \cdot A \cdot P. \tag{D.8}$$

Die Spaltenvektoren von P sind die *Eigenvektoren* von A. Die Komponenten von D entlang der Hauptdiagonalen entsprechen den *Eigenwerten* von A. Weiter bemerken wir, dass wir für eine symmetrische Matrix stets eine *orthogonale* Matrix P finden können, weil alle Eigenvektoren der symmetrischen Matrix zueinander rechtwinklig sind (siehe [BI78.1, S. 150]). Ist P orthogonal, gilt:

$$P^{-1} = P^T. \tag{D.9}$$

Da die Matrix $\hat{g}$ symmetrisch ist, können wir sie wie folgt diagonalisieren:

$$D = P^T \cdot \hat{g} \cdot P. \tag{D.10}$$

Hier ist P die Matrix der normalisierten Eigenvektoren von $\hat{g}$ und D die Diagonalmatrix mit den Eigenwerten von $\hat{g}$.

Die Eigenwerte von $\hat{g}$ können durch Lösen des *charakteristischen Polynoms* von $\hat{g}$ gefunden werden:

$$\lambda_0 = \frac{1}{2}\left(\hat{g}_{tt} + \hat{g}_{\phi\phi} + \sqrt{(\hat{g}_{tt} - \hat{g}_{\phi\phi})^2 + 4\hat{g}_{t\phi}^2}\right)$$

$$\begin{aligned}\lambda_1 &= \hat{g}_{rr}\\ \lambda_2 &= \hat{g}_{\theta\theta}\\ \lambda_3 &= \frac{1}{2}\left(\hat{g}_{tt} + \hat{g}_{\phi\phi} - \sqrt{(\hat{g}_{tt} - \hat{g}_{\phi\phi})^2 + 4\hat{g}_{t\phi}^2}\right).\end{aligned} \tag{D.11}$$

Die Eigenvektoren werden folgendermassen berechnet:

$$P = \begin{pmatrix} 1 & 0 & 0 & \hat{g}_{t\phi}/(\lambda_0 - \hat{g}_{tt}) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hat{g}_{t\phi}/(\lambda_3 - \hat{g}_{\phi\phi}) & 0 & 0 & 1 \end{pmatrix}. \tag{D.12}$$

Wir normalisieren anschliessend die Spalten von P, so dass P orthogonal wird.

Skalierung Um von der diagonalisierten Metrik die flache Metrik zu erhalten, muss zusätzlich skaliert werden:

$$\eta = S \cdot (P^T \cdot \hat{g} \cdot P), \tag{D.13}$$

wobei S eine Skalierungsmatrix ist:

$$S = \mathrm{diag}(|\lambda_0|^{-1}, |\lambda_1|^{-1}, |\lambda_2|^{-1}, |\lambda_3|^{-1}), \tag{D.14}$$

dies ist möglich, weil die Vorzeichen von $P^T \cdot \hat{g} \cdot P$ gleich sind wie die von η.

Wir bemerken, dass die Matrix S einfach in zwei diagonale Matrizen aufgeteilt werden kann, da S selber diagonal ist. Ausserdem ist die Matrixmultiplikation zweier diagonaler Matrizen kommutativ. Wir können aus diesen Gründen eine Koordinatentransformation Ω aufstellen:

$$\eta = \sqrt{S}^T \cdot (P^T \cdot \hat{g} \cdot P) \cdot \sqrt{S} = \Omega^T \cdot \hat{g} \cdot \Omega, \tag{D.15}$$

wobei

$$\sqrt{S} = \sqrt{S}^T = \mathrm{diag}(|\lambda_0|^{-1/2}, |\lambda_1|^{-1/2}, \dots) \tag{D.16}$$

bedeutet.

Weitere Transformationen Ω entspricht noch nicht der gesuchten Koordinatentransformation Ψ. Wir können Ω durch Transformationen erweitern, welche Vektoren verändern, aber die Metrik unangetastet lassen. Die beiden Möglichkeiten sind eine Rotation und eine Lorentz-Transformation.

Rotation Wir betrachten zuerst die Möglichkeit, den Beobachter beliebig auszurichten. Rotationen verändern die räumliche Distanz zwischen zwei Ereignissen nicht und lassen Zeit-Komponenten unangetastet; die Metrik bleibt deshalb unter einer Rotation erhalten. Wir benötigen eine Rotation, um die kartesischen Achsen gemäss Abbildung 6.4.2 auszurichten.

Wenn wir jedoch die Auswirkung von Ω auf die kartesischen Basisvektoren untersuchen, sehen wir, dass keine zusätzliche Ausrichtung notwendig ist: Die gewählte Anordnung der Eigenvektoren von $\hat{g}$ in P hat zur Folge, dass die Aktion von P auf die räumlichen Teile von Vektoren eine reine Skalierung

ist; ausserdem ist durch die Anordnung gewährleistet, dass die kartesischen Basisvektoren den richtigen BL-Basisvektoren entsprechen, wie wir verlangten.

Lorentz-Transformation Eine Lorentz-Transformation (siehe Kapitel 1.1) ermöglicht uns, die Geschwindigkeit des Beobachters relativ zu den BL-Koordinaten beliebig zu wählen. Wir verlangen, dass der Beobachter relativ zu den BL-Koordinaten stationär ist. Der Geschwindigkeitsvektor des Beobachters ist demnach

$$\vec{V}_{\mathrm{BL}} \hat{=} V^{\alpha}_{\mathrm{BL}} = \left(\sqrt{\frac{1}{|\hat{g}_{tt}|}}, 0, 0, 0 \right). \tag{D.17}$$

Die Zeitkomponente wurde so gewählt, dass $\hat{g}_{\alpha\beta} V^{\alpha}_{\mathrm{BL}} V^{\beta}_{\mathrm{BL}} = -1$. Dies entspricht einer Parametrisierung mit der Eigenzeit des Beobachters.

Die quadrierte Magnitude des Vektors muss in allen Koordinatensystemen gleich sein, da diese nicht durch Koordinatentransformationen beeinflusst wird. Im kartesischen Inertialsystem des Beobachters gilt deshalb für die Geschwindigkeit des Beobachters

$$\vec{V}_{\mathrm{k}} \hat{=} V^{\alpha}_{\mathrm{k}} = (1, 0, 0, 0) \tag{D.18}$$

und $\eta_{\alpha\beta} V^{\alpha}_{\mathrm{k}} V^{\beta}_{\mathrm{k}} = -1$.

Damit der Beobachter wirklich stationär ist, muss die gesuchte Koordinatentransformation

$$\vec{V}_{\mathrm{BL}} = \Psi \cdot \vec{V}_{\mathrm{k}} \tag{D.19}$$

erfüllen. Wir sehen, dass Ω diese Bedingung nicht erfüllt. Durch Anwenden von Ω wird die ϕ-Komponente eines rein zeitartigen Vektors ungleich null; dies ist auf den Kreuzterm t-ϕ in der Kerr-Newman-Metrik zurückzuführen. Das heisst, dass Ω lokale kartesische Koordinaten eines entlang von ϕ bewegten Beobachters in BL-Koordinaten umwandelt.

Da wir einen relativ zu den BL-Koordinaten stationären Beobachter konstruieren möchten, korrigieren wir diese Bewegung durch eine Lorentz-Transformation aus:

$$\Psi = \Omega \cdot \Lambda, \tag{D.20}$$

wobei Λ die Lorentz-Transformation ist. Die Lorentz-Transformation wird in kartesischen Koordinaten angewandt. Wir wissen, dass die Bewegung ausschliesslich in ϕ-Richtung korrigiert werden muss; in lokalen kartesischen Koordinaten entspricht dies einer Korrektur in z-Richtung.

Inverse Lorentz-Transformation Wir werden von nun an mit der inversen Lorentz-Transformation Λ^{-1} fortfahren, da diese einfacher gefunden werden kann. Sie hat folgende Form:

$$\Lambda^{-1} = \begin{pmatrix} \gamma & 0 & 0 & -\gamma v \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\gamma v & 0 & 0 & \gamma \end{pmatrix}. \tag{D.21}$$

Sie erfüllt gemäss Gleichungen (D.19) und (D.20) folgende Bedingung:

$$\Lambda^{-1} \cdot \Omega^{-1} \cdot \vec{V}_{\mathrm{BL}} = \Lambda^{-1} \cdot \vec{X} = \vec{V}_{\mathrm{k}}, \tag{D.22}$$

wobei wir den Vektor $\vec{X} = \Omega^{-1} \cdot \vec{V}_{\mathrm{BL}}$ einführten. Unter Beachtung der Gleichungen (1.1.3) und (D.21) können wir nach v auflösen und γ berechnen:

$$v = \frac{\vec{X}_z}{\vec{X}_t} \tag{D.23}$$

$$\gamma = \frac{\vec{X}_t}{\sqrt{\vec{X}_t^2 - \vec{X}_z^2}} = \vec{X}_t. \tag{D.24}$$

Hier beachten wir, dass $\vec{X}_t^2 - \vec{X}_z^2 = 1$, weil $\eta_{\alpha\beta} X^\alpha X^\beta = -1$. Durch Einsetzen in Gleichung (D.21) können wir Λ^{-1} berechnen.

Komplette Koordinatentransformation Die komplette Koordinatentransformation ist

$$\Psi = P \cdot \sqrt{S} \cdot \Lambda. \tag{D.25}$$

Sie erfüllt die Gleichungen (D.4) und (D.2) und sorgt dafür, dass der Beobachter in BL-Koordinaten stationär und korrekt ausgerichtet ist.

Anhang E

cuRRay: Bedienungsanleitung

E.1 Kommandozeile

`cuRRay` wird über die Kommandozeile gestartet. Folgende Optionen können angegeben werden:

```
cuRRay  [-s <Szenendatei>] [-o <Ausgabeverzeichnis>]
        [-x <Framebreite>] [-y <Framehöhe>] [-f <Anzahl Frames>]
        [-t <Ausgabetyp>] [-c <Systemkonfigurationsdatei>]
        [-v <true/false>] [-q <true/false>] [-a <true/false>]
        [-i] [-h] [--version] [--]
```

Die Kommandozeile für die CPU-Version, `cuRRay_cpu`, sieht sehr ähnlich aus:

```
cuRRay_cpu  [-s <Szenendatei>] [-o <Ausgabeverzeichnis>]
        [-x <Framebreite>] [-y <Framehöhe>] [-f <Anzahl Frames>]
        [-t <Ausgabetyp>] [-c <Systemkonfigurationsdatei>]
        [-v <true/false>] [-q <true/false>] [-a <true/false>]
        [-h] [--version] [--]
```

In diesem Kapitel werden wir alle Optionen kurz beschreiben.

`-s, --scene_file <Szenendatei>` Mit dieser Option wird die Szenendatei angegeben. Zum Beispiel: `-s szene.yml`.

`-o, --output_dir <Ausgabeverzeichnis>` Diese Option setzt das Ausgabeverzeichnis. Zum Beispiel: `-o ausgabe`.

`-x, --x_res <Framebreite>` Die Breite der zu zeichnenden Frames (ganze Zahl, grösser als null). Zum Beispiel `-x 1000`.

`-y, --y_res <Framehöhe>` Die Höhe der zu zeichnenden Frames (ganze Zahl, grösser als null). Zum Beispiel `-y 1000`.

`-f, --frames <Anzahl Frames>` Die Anzahl der zu zeichnenden Frames (ganze Zahl, grösser oder gleich null). Wird standardmässig auf eins gesetzt. Zum Beispiel `-f 20`.

`-t, --output_type <Ausgabetyp>` Der Typ der Ausgabe. Diese Option kann eine Kombination der Zeichen `i`, `c`, `r` und `d` sein. Wird die Option nicht gesetzt, verwendet `cuRRay` standardmässig `icr`.

Wird `i` angegeben, erzeugt die Software eine Textdatei (`info.txt`) mit einer Zusammenfassung zum Raytracing-Auftrag im Ausgabeverzeichnis.

Wird `c` gesetzt, erzeugt `cuRRay` PNG-Dateien mit den berechneten Pixelfarben für jede Frame (nummeriert von `c_0.png` bis `c_n.png`).

Wird `r` gesetzt, erzeugt `cuRRay` PNG-Dateien mit den berechneten Rotverschiebungen für jede Frame (nummeriert von `r_0.png` bis `r_n.png`).

Wird `d` gesetzt, erzeugt `cuRRay` CSV-Dateien mit den genauen Pixeldaten für jede Frame (nummeriert von `0.csv` bis `n.csv`).

`-c, --config <Systemkonfigurationsdatei>` Die Systemkonfigurationsdatei (Sysconfig). Standardmässig wird `config.yml` verwendet.

`-v, --verbose <true/false>` Diese Option kann entweder auf `true` (wahr) oder `false` (falsch) gesetzt werden, um ein- oder ausgeschaltet zu werden. Wird die Option gesetzt, führt `cuRRay` eine ausführliche Berichterstattung, ansonsten werden nur die wichtigsten Meldungen angezeigt. Wird die Option in der Kommandozeile angegeben, überschreibt sie die in der Sysconfig angegebene Einstellung. Zum Beispiel `cuRRay -v true`.

`-q, --quiet <true/false>` Wird diese Option gesetzt, schreibt `cuRRay` keine Ausgabe in die Konsole, sondern nur in die Logdatei. Überschreibt die Einstellung in Sysconfig.

`-a, --auto_accept <true/false>` Bevor `cuRRay` das Raytracing startet, wird die aktuelle Konfiguration angezeigt und der Benutzer kann entscheiden, ob er einverstanden ist. Ist diese Option gesetzt, wird die Konfiguration automatisch angenommen. Überschreibt die Einstellung in Sysconfig. Falls `-q` gesetzt ist, also keine Konsolenausgabe erzeugt wird, nimmt `cuRRay` die Konfiguration ebenfalls automatisch an.

`-i` Diese Option ist nur für die CUDA-Version verfügbar. Wenn angegeben, zeigt `cuRRay` detaillierte Informationen über verfügbare CUDA-Grafikkarten an. Insbesondere wird für jede erkannte Grafikkarte angezeigt, ob sie von `cuRRay` verwendet werden kann. Listing E.1 zeigt eine mögliche Ausgabe:

Listing E.1: Mögliche Ausgabe von ***cuRRay -i***

```
> cuRRay -i

-------------------------------------------------------

                      ____      ____
       _______  __   / __ \    / __ \____ ___  __
      / ___/ / / /  / /_/ /   / /_/ / __ `/ / / /
     / /__/ /_/ /   / _, _/   / _, _/ /_/ / /_/ /
     \___/\__,_/   /_/ |_|   /_/ |_|\__,_/\__, /
                                          /____/
       CUDA(R)  Relativistic     Raytracer
                Version  2.0

                     ,
          (c) 2018 Sebastien Garmier

-------------------------------------------------------

CUDA GPU information:

Driver version: 9010
Required compute capability: 3.0

Device #0

Can be used:                        Yes

Name:                               GeForce GTX 760
Compute capability:                 3.0
Clock rate:                         1071500
Device copy overlap:                enabled
Kernel execution timeout:           disabled
Total global memory:                2147483648
Total const. memory:                65536
Max. memory pitch:                  2147483647
Texure alignment:                   512

Multiprocessor count:               6
Max. threads per multiprocessor:    2048
Shared memory per multiprocessor:   49152
Registers per multiprocessor:       65536
Threads per warp:                   32
Max. threads per block:             1024
Max. block dimensions:              (1024,1024,64)
```

```
45 Max. grid dimensions:
       (2147483647,65535,65535)
46
47 ---------------------------------------------------
48
49 No scene file specified, no raytracing will be
       performed.
```

-h, --help Diese Optionen zeigt eine Liste mit allen Optionen und deren Erklärung an.

--version zeigt die Software-Version an. Zum Zeitpunkt des Drucks ist dies 2.0.

-- Zwei Trennstriche gefolgt von einem Leerzeichen bezeichnet das Ende der Kommandozeile. Alle Zeichen danach werden ignoriert. Zum Beispiel sind die folgenden zwei Kommandozeilen identisch:

```
cuRRay -- Diese Zeichen werden ignoriert
cuRRay
```

E.2 Systemkonfigurationsdatei (Sysconfig)

Die *Systemkonfigurationsdatei* (kurz: *Sysconfig*) enthält wichtige Einstellungen betreffend GPU, CPU, Ausgabe und Raytracing-Algorithmus. Standardmässig wird `config.yml` als Sysconfig-Datei geladen. Siehe Anhang E.1, um eine andere Datei zu laden. Die Sysconfig-Datei verwendet YAML als Dateiformat (siehe `http://yaml.org`).

Es sollte angemerkt werden, dass YAML keine Tabulatoren zur Strukturierung verwendet; alle Einrückungen müssen deshalb mit Leerschlägen vorgenommen werden.

GPUs In der Sysconfig der CUDA-Version kann angeben werden, welche Grafikkarten von `cuRRay` verwendet werden sollen. Beispielsweise wird im folgenden Auszug einer Sysconfig die erste Grafikkarte (Nr. 0) nicht verwendet, dafür die zweite (Nr. 1):

```
gpus:
    0:
        enabled: false
    1:
        enabled: true
```

Besitzt das System nur eine Grafikkarte, kann der Abschnitt für Nr. 1 entfernt werden. Gleichermassen können weitere Abschnitte hinzugefügt werden, wenn mehr Grafikkarten zur Verfügung stehen. Die in der Sysconfig verwendeten Nummern sind die gleichen, die in `cuRRay -i` angezeigt werden.

CPU In der Sysconfig der CPU-Version kann angegeben werden, wie viele CPU-Threads fürs Raytracing verwendet werden sollen. Um die Leistung des Prozessors voll auszulasten, sollten gleich viele Threads verwendet werden, wie virtuelle Kerne vorhanden sind (bzw. Kerne, wenn der Prozessor kein Hyperthreading unterstützt). Zum Beispiel:

```
cpu:
    threads: 8
```

Raytracer In der Sysconfig können die Konstanten ϵ (siehe Kapitel 6.8) und c (siehe Kapitel 6.6) gesetzt werden. Zum Beispiel:

```
raytracer:
    horizon_epsilon: 0.5E-2
    step_multiplier: 0.03125
```

Log Die Berichterstattung kann ebenfalls konfiguriert werden. Zum Beispiel:

```
log:
    verbose: false
    quiet: false
    keep_log: false
```

`verbose` entspricht der Kommandozeilenoption `-v`.
`quiet` entspricht der Kommandozeilenoption `-q`.
`keep_log` kann nicht über die Kommandozeile gesetzt werden. Diese Option legt fest, ob die Logdatei separat abgespeichert wird, nachdem das Programm beendet wurde. Wird diese Option nicht gesetzt, bleibt die letzte Logdatei (`log/log.txt`) bestehen, bis `cuRRay` erneut gestartet wird und die Datei überschreibt. Fehlermeldungen werden stets separat abgespeichert.

Separat abgespeicherte Logdateien werden ebenfalls im Verzeichnis `log/` gespeichert, erhalten aber einen eindeutigen Namen, damit sie nicht überschrieben werden.

Automatische Ausführung Die Option `auto_accept` entspricht der Kommandozeilenoption `-a`. Zum Beispiel:

```
auto_accept: false
```

Beispiel Listing E.1 zeigt eine mögliche Standard-Sysconfig `config.yml`.

Listing E.1: Standard-Sysconfig `config.yml`

```
1 # Default config file
2
3 ---
4
5 # This configures the GPUs used by cuRRay.
6 # All GPUs that are not listed here are disabled
```

```
# by default.
# If your particular setup supports more than
# two GPUs, you may extend the gpus list by
# new entries.
# By default, only GPU #0 is enabled
gpus:
    0:
        enabled: true
    1:
        enabled: false

# This configures how the CPU is used in
# CPU-only mode (cuRRay_cpu)
# By default, 4 threads are used.
cpu:
    threads: 4

# Raytracer configuration
raytracer:
    horizon_epsilon: 0.5E-2
    step_multiplier: 0.03125

# This configures the logging behaviour of cuRRay.
# By default, the output is non-verbose,
# the log is mirrored to the console and the logfile
# is overriden every time the software runs.
log:
    verbose: false
    quiet: false
    keep_log: false

# Tells cuRRay to automatically accept the scene
# configuration.
# If no cmd output is created, the configuration
# is always directly accepted.
auto_accept: false
```

E.3 Raytracing

Um Raytracing durchzuführen, müssen mindestens die Optionen `-s`, `-o`, `-x` und `-y` gesetzt sein. Weiter können `-f` und `-t` das Raytracing beeinflussen. Für eine Erklärung dieser Optionen, siehe Anhang E.1.

Beispiel Um die Szenendatei `szene.yml` und das Ausgabeverzeichnis `ausgabe` zu verwenden, um für 20 Frames mit einer Auflösung von je 2000 x 1000 Pixel

Farbbilder und Rotverschiebungsbilder zu erstellen, wird folgende Kommandozeile verwendet:

```
cuRRay -s szene.yml -o ausgabe -x 2000 -y 1000 -f 20 -t cr
```

E.4 Szenendatei

Die Szenendatei beschreibt die Szene, von der die Frames gezeichnet werden. Die Szenendatei ist wie die Sysconfig-Datei eine YAML-Datei.

Winkel Winkel können in der Szenendatei auf drei Arten angegeben werden: Grad, Radiant und Radiant als Vielfaches von π. Folgende vier Werte beschreiben alle den gleichen Winkel (90°):

```
90
90 deg
1.570796 rad
0.5 pi
```

Die Suffixe `deg` (Grad), `rad` (Radiant) und `pi` (Radiant als Vielfaches von π) bestimmen, in welchen Einheiten der Winkel eingelesen wird. Wird kein Suffix aufgeführt, nimmt `cuRRay` automatisch Grad an.

Farben RGB-Farben werden in der Szenendatei als Liste der drei Farbkomponenten dargestellt. Jede Komponente kann Werte von 0 bis 255 annehmen. Die Farbe gelb ($R = 255$, $G = 255$, $B = 0$) würde zum Beispiel folgendermassen dargestellt:

```
[255, 255, 0]
```

Animierte Werte Gewisse Werte können über mehrere Frames hinweg animiert werden. Animierte Werte folgen diesem Muster:

```
[linear, <Startwert>, <Endwert>]
```

`<Startwert>` und `<Endwert>` sind die Werte, die für die erste, beziehungsweise letzte Frame verwendet werden. Für die dazwischenliegenden Frames wird der Wert zwischen Start- und Endwert linear interpoliert. Beispielsweise kann ein Winkel von 0 bis 2π folgendermassen animiert werden:

```
[linear, 0, 2 pi]
```

Metrik In jeder Szenendatei müssen die Parameter M, a und Q der Metrik gesetzt werden (siehe 2.2). Alle drei Parameter können animiert werden. Zum Beispiel:

```
metric:
    m: 1
    a: 0.5
    q: 0
```

Beobachter Der Beobachter muss ebenfalls in jeder Szene konfiguriert werden. Zum Beispiel:

```
observer:
    r: 30
    theta: 0.5 pi
    phi: 1 pi

    roll: 0
    pitch: 5 deg
    yaw: 0

    hfov: 70 deg
    vfov: 70 deg
```

`r`, `theta` und `phi` sind die BL-Koordinaten des Beobachters; sie sind zwingend notwendig und können animiert werden.

`roll`, `pitch` und `yaw` sind die Roll-, Nick- und Gierwinkel, welche die Ausrichtung des Beobachters bestimmen. Diese Winkel müssen nicht unbedingt gesetzt werden; sie sind standardmässig null. Wie in Kapitel 6.4 erklärt wird, ist der Beobachter standardmässig bereits so ausgerichtet, dass er das schwarze Loch direkt betrachtet. Alle drei Winkel können animiert werden.

`hfov` und `vfov` sind die Sichtfeldwinkel des Beobachters. Nur der horizontale Sichtfeldwinkel `hfov` muss angegeben werden, `vfov` ist optional. Wird `vfov` nicht spezifiziert, berechnet `cuRRay` den vertikalen Blickfeldwinkel aus dem Verhältnis der Frame-Abmessungen, so dass das Bild nicht verzerrt wird:

$$\frac{\tan(\texttt{vfov}/2)}{\tan(\texttt{hfov}/2)} = \frac{h}{b}, \tag{E.4.1}$$

wobei h die Höhe des Frames und b die Breite ist. Beide Winkel können animiert werden.

Akkretionsscheibe Optional kann in der Szene eine Akkretionsscheibe, welche um $r = 0$ zentriert ist und in der Ebene $\theta = \pi/2$ liegt, konfiguriert werden. Zum Beispiel:

```
accretion:
    color1: [0, 255, 0]
    color2: [255, 0, 255]

    resolution: [2, 12]

    yaw: 0
    radius: [5, 15]
```

`color1` und `color2` sind die Akzentfarben für die Ober- bzw. Unterseite der Scheibe. Nur `color1` muss unbedingt angegeben werden. Wird `color2`

nicht angegeben, berechnet `cuRRay` die Akzentfarbe der Unterseite folgendermassen:

$$\begin{aligned} R_2 &= 255 - R_1, \\ G_2 &= 255 - G_1, \\ B_2 &= 255 - G_1, \end{aligned} \tag{E.4.2}$$

wobei R_1, G_1 und B_1 die Farbkomponenten von `color1` und R_2, G_2 und B_2 die von `color2` sind.

`resolution` ist die Auflösung des Schachbrettmusters. Der erste Wert entspricht der Anzahl radialer Unterteilungen und der zweite Wert der Anzahl Sektoren der Schachbretttextur. Diese Werte müssen angegeben werden.

`yaw` bezeichnet den Winkel, um den die Schachbretttextur um die Polachse gedreht wird. Dieser Wert ist standardmässig null und kann animiert werden.

`radius` sind die beiden Radien der Scheibe. Der erste Wert bezeichnet die r-Koordinate des inneren Rands und der zweite die des äusseren Rands. Beide Werte müssen angegeben werden und können animiert sein.

Sternenhimmel Optional kann ein Bild an der Himmelskugel aufgespannt werden (siehe Kapitel 6.10). Zum Beispiel:

```
skymap:
    image: sky.png
    boundary: 20
```

`image` ist die Bilddatei, welche die Plattkartenprojektion des Himmels beinhaltet. Die Bilddatei muss angegeben werden.

`boundary` ist die Szenengrenze R, ausserhalb deren die Raumzeit als flach angenommen wird. Dieser Wert muss angeben werden und kann animiert sein.

Kugeln Bis zu acht Kugeln können in der Raumzeit positioniert werden. Für jede Kugel wird ein separater Abschnitt in der Szenendatei hinzugefügt. Zum Beispiel:

```
sphere:
    color: [255, 255, 0]
    resolution: [4, 8]

    r: 10
    theta: 90
    phi: [linear, 0, 360]

    roll: 0
    pitch: 0
    yaw: 0

    radius: 3
```

`color` ist die Akzentfarbe der Kugel und wird benötigt.

`resolution` ist die Auflösung der Schachbretttextur und wird ebenfalls benötigt. Der erste Wert ist die Anzahl Unterteilungen parallel zu den Breitengraden und der zweite Wert die Anzahl Unterteilungen entlang der Längengrade.

`r`, `theta` und `phi` sind die BL-Koordinaten des Mittelpunkts der Kugel. Sie müssen angegeben werden und können animiert sein.

`roll`, `pitch` und `yaw` ermöglichen die genaue Ausrichtung der Kugel. Die drei Winkel drehen die Kugel um lokale kartesische Koordinatenachsen in dieser Reihenfolge: `roll` dreht die Kugel um die negative x-Achse, `pitch` dreht sie um die negative z-Achse und `yaw` um die negative y-Achse. Diese Koordinatenachsen sind analog zu denen, die für die Berechnung der Anfangsgeschwindigkeitsvektoren verwendet werden, mit dem Unterschied, dass der Koordinatenursprung im Mittelpunkt der Kugel und nicht beim Beobachter gewählt wird. Alle drei Winkel sind standardmässig null und können animiert werden.

`radius` ist der Radius der Kugel. Dieser Wert muss angegeben werden und kann animiert sein.

Farben In der Szenendatei können ebenfalls spezielle Farben eingestellt werden. Zum Beispiel:

```
sky_color: [0, 0, 0]
horizon_color: [0, 0, 0]
error_color: [0, 0, 255]
```

`sky_color` ist die Farbe des Himmels. Wird sie nicht gesetzt, verwendet die Software standardmässig schwarz.

`horizon_color` ist die Farbe des Ereignishorizonts. Wird sie nicht gesetzt, verwendet die Software standardmässig rot.

`error_color` ist die Farbe, die für fehlerhafte Pixel verwendet wird. Wird sie nicht gesetzt, verwendet die Software standardmässig blau.

Beispiele Für Beispiele, siehe die Szenendateien in Kapitel 7.

E.5 TDR-Timer unter Windows

Je nach Konfiguration, kann das Raytracing mehrere Minuten dauern. Während dieser Zeit wird die Grafikkarte voll ausgelastet. Falls die Anzeige des Computers (wenn vorhanden) von der gleichen Grafikkarte angesteuert wird, friert der Computer ein und kann nicht bedient werden, bis `cuRRay` beendet ist.

Um dies zu verhindern, besitzt Windows eine eingebaute Sicherheit, den *TDR-Timer* (Timeout Detection and Recovery Timer), welcher Programme stoppt, wenn sie die Grafikkarte zu lange belasten. Dies ist natürlich nicht erwünscht; der TDR-Timer sollte deshalb deaktiviert werden.

Der TDR-Timer kann über den NVIDIA Nsight-Monitor (falls installiert) deaktiviert werden [1]. Ansonsten kann der TDR-Timer in der Registry deaktiviert werden [2].

E.6 Quellcode-Dokumentation

Alle Code-Dateien enthalten Dokumentations-Kommentare, aus denen mit Doxygen [3] eine HTML- oder LaTeX-Dokumentation erstellt werden kann. Das Repository (siehe Anhang F) enthält eine bereits kompilierte Dokumentation. Die Code-Dokumentation kann hilfreich sein, wenn an `cuRRay` weitergearbeitet wird.

[1] Siehe `http://docs.nvidia.com/gameworks/content/developertools/desktop/nsight/timeout_detection_recovery.htm`

[2] Siehe `https://docs.microsoft.com/en-us/windows-hardware/drivers/display/tdr-registry-keys`

[3] http://www.doxygen.org/

Anhang F

Git-Repository

Das Git-Repository unter `https://gitlab.com/sebiG/cuRRay` enthält den Quellcode der Software, eine Quellcode-Dokumentation, bereits kompilierte Versionen für Windows 10 und Linux, mit `cuRRay` erzeugte Beispielbilder und dieses Dokument.

Lizenz `cuRRay` und dazugehörige Werke werden unter der MIT-Lizenz [1] vertrieben. Eine Kopie der Lizenz kann im Repository am Ende der Endbenutzerlizenzvereinbarung (`EULA.txt`) gefunden werden.

Versionen Die höchste Versionsnummer, welche durch Tags im Repository vermerkt ist, widerspiegelt die neuste Version der Software. Der Master-Zweig enthält das neuste, stabile Release. Änderungen in zusätzlichen Dateien, wie kompilierte Versionen und PDFs werden nicht durch Tags gekennzeichnet. Die neusten Versionen dieser Dateien werden dem Master-Zweig hinzugefügt. Um die neuste Version zusätzlicher Dateien zu erhalten, können diese einfach vom Master-Zweig heruntergeladen werden.

Weiter Informationen Weiter Informationen werden möglicherweise in Zukunft auf der Homepage des Autors, `sebastiengarmier.ch`, aufgeschaltet.

[1] Siehe `https://opensource.org/licenses/MIT`

Anhang G

Kompilieren

Dieser Anhang beschreibt die Kompilierung von `cuRRay` unter Windows und Linux. Ausserdem wird ein Leitfaden zur Struktur des Quellcodes gegeben. Ähnliche Informationen können in der Readme-Datei (`README.txt`) des Repositorys gefunden werden.

G.1 Allgemeines

`cuRRay` sollte im 64-Bit-Modus kompiliert werden. Eine Kompilierung im 32-Bit-Modus wurde nie getestet und ist möglicherweise fehlerhaft.

Unter Windows wie auch unter Linux werden Software-Bibliotheken benötigt. Folgende Bibliotheken müssen installiert und mit der Compiler-Konfiguration, mit welcher auch `cuRRay` kompiliert werden soll, kompiliert werden:

Boost `https://www.boost.org/`

libPNG `https://www.libpng.org/pub/png/libpng.html`

zlib `https://zlib.net/`

yaml-cpp `https://github.com/jbeder/yaml-cpp`

Siehe Kapitel 4.3 für die Versionen, welche für `cuRRay` empfohlen werden. Je nach Betriebssystem sind bereits kompilierte Versionen dieser Bibliotheken verfügbar.

G.2 Kompilieren unter Windows (Visual Studio)

Visual Studio Eine Version von Visual Studio [1], welche das CUDA toolkit 8.0 und das Plattform-Toolset v120 unterstützt, muss installiert werden. Das CUDA toolkit [2] muss ebenfalls installiert werden.

Umgebungsvariablen Folgende Umgebungsvariablen sollten gesetzt werden:

[1] Siehe `https://www.visualstudio.com/`

[2] Siehe `https://developer.nvidia.com/cuda-toolkit`

`INCLUDE` : Diese Umgebungsvariable muss die Include-Verzeichnisse aller verwendeten Bibliotheken enthalten.

`LIB` : Diese Umgebungsvariable muss die Bibliotheksverzeichnisse aller (sowohl Debug, als auch Release) verwendeten Bibliotheken enthalten. Die bereits konfigurierten Projektoptionen der Projekte in der Projektmappe geben Aufschluss, wie diese Verzeichnisse und die Bibliotheksdateien angeordnet sein müssen, so dass die Kompilierung erfolgreich ist.

Kompilieren Die Projektmappe, welche im Repository (siehe Anhang F) enthalten ist, kann einfach für entweder Debug oder Release kompiliert werden. Dabei ist zu beachten, dass die Konfiguration auf 64-Bit gestellt werden muss.

Die Projektmappe enthält Projekte für die CUDA-Version (`cuRRay_dev`) und die CPU-Version (`cuRRay_dev_cpu`).

Ausführen Um `cuRRay` ausführen zu können, müssen die DLL-Dateien aller benötigten Bibliotheken ins Verzeichnis der ausführbaren Datei kopiert werden.

Die CUDA-DLL (`cudart64_80.dll`) sollte bereits automatisch durch Visual Studio kopiert werden (nur für CUDA-Version erforderlich).

Falls die Software auf einem System ohne Visual Studio ausgeführt wird, müssen die C- und C++-Runtimes des Plattform-Toolsets v120 ebenfalls kopiert werden (`msvcr120.dll` und `msvcp120.dll`). Diese werden mit Visual Studio mitgeliefert.

Ausserdem muss `libpng16.dll` kopiert werden. Diese sollte im Fall einer bereits kompilierten Version von `libPNG` mitgeliefert werden oder kann aus dem Quellcode von `libPNG` kompiliert werden.

G.3 Kompilieren unter Linux (g++-5)

GNU-make und g++ Für die Kompilierung unter Linux werden im Repository GNU-Makefiles für g++ zur Verfügung gestellt. Die Makefiles wurden mit g++ 5 unter Debian 8 erfolgreich getestet. Andere Compiler funktionieren möglicherweise auch, wenn die Makefiles angepasst werden. Wie unter Windows muss das CUDA-Toolkit installiert werden. Dazu sollte ebenfalls die Online-Dokumentation auf der NVIDIA-Homepage konsultiert werden.

Es stehen Release-Makefiles für die CUDA-Version und die CPU-Version zur Verfügung. Debug-Makefiles sind nicht vorhanden.

CUDA-Version Das Makefile `release.makefile` kompiliert die CUDA-Version.

CPU-Version Das Makefile `release_cpu.makefile` kompiliert die CPU-Version.

Ausführen Wenn die benötigten Bibliotheken mittels Paketmanager installiert wurden, sollte das System beim Ausführen beider `cuRRay`-Versionen die kompilierten Bibliotheksdateien automatisch finden. Wurden die Bibliotheken

von Hand kompiliert, kann es sein, dass die Umgebungsvariable `LD_LIBRARY_PATH` gesetzt werden muss, so dass die kompilierten Bibliotheken gefunden werden.

Abbildungsverzeichnis

1.2.1 Äquivalenzprinzip 3
1.3.1 Verbildlichung der gekrümmten Raumzeit 5
1.7.1 Paralleler Transport in flacher und gekrümmter Raumzeit . . 10
1.8.1 Relative Beschleunigung zwischen Geodäten 12

2.2.1 Kerr-Newman-Metrik 20
2.2.2 Extreme Kerr-Newman-Metrik 21

3.4.1 Ablenkung von Licht durch ein schwarzes Loch 26

5.2.1 Prozesse und Threads 34
5.2.2 Blöcke und Threads 35

6.4.1 Anfangsgeschwindigkeitsvektor eines Pixels 39
6.4.2 Ausrichtung der lokalen kartesischen Koordinaten 40
6.10.1 Plattkartenprojektion des Nachthimmels 46

7.1.1 Akkretionsscheibe eines Schwarzschild-Lochs, $\theta = 5°$ 52
7.1.2 Akkretionsscheibe eines Schwarzschild-Lochs, $\theta = 85°$ 52
7.1.3 Sternenhimmel hinter Schwarzschild-Loch 53
7.1.4 Schwarzschild-Loch mit Kugel um Ereignishorizont 53
7.1.5 Rotverschiebung um Schwarzschild-Loch 53
7.1.6 Blauverschiebung um Schwarzschild-Loch 54
7.1.7 Blick von Schwarzschild-Loch nach aussen 54
7.1.8 Kugel vor Schwarzschild-Loch 55
7.1.9 Kugel neben Schwarzschild-Loch 55
7.1.10 Kugel hinter Schwarzschild-Loch 55
7.2.1 Akkretionsscheibe eines Kerr-Lochs, $\theta = 5°$ 59
7.2.2 Akkretionsscheibe eines Kerr-Lochs, $\theta = 85°$ 59
7.3.1 Kugel hinter Reissner-Nordström-Loch 61

8.1.1 Fehler entlang der Achse 64

Titelseite: Schwarzschild-Loch mit Scheibe

Listingverzeichnis

7.1.1 Akkretionsscheibe eines Schwarzschild-Lochs 56
7.1.2 Sternenhimmel hinter Schwarzschild-Loch 56
7.1.3 Schwarzschild-Loch mit Kugel um Ereignishorizont 56
7.1.4 Rotverschiebung um Schwarzschild-Loch 56
7.1.5 Blauverschiebung in der Nähe eines Schwarzschild-Lochs 57
7.1.6 Kugel in der Nähe von Schwarzschild-Loch 57
7.2.1 Akkretionsscheibe eines Kerr-Lochs 60
7.3.1 Kugel hinter Reissner-Nordström-Loch 61

E.1 Mögliche Ausgabe von `cuRRay -i` 80
E.1 Standard-Sysconfig `config.yml` 82

Literaturverzeichnis

[AB16.1] Abbott, B. P. et al., 2016: *Observation of Gravitational Waves from a Binary Black Hole Merger*, DOI: 10.1103/PhysRevLett.116.061102 (Abruf: 18.12.16).

[BI78.1] Bronshtein, I. N.; Semendyayev, K. A., 1978: *Handbook of Mathematics: English Translation, edited by K. A. Hirsch*, Frankfurt am Main: Verlag Harri Deutsch.

[CC13.1] Chan, Chi-Kwan; Özel Feryal; Psaltis Dimitrios, 2013: *`GRay`: A massively parallel GPU-based code for ray tracing in relativistic spacetimes*, arXiv:1303.5057v1 (Abruf: 17.09.16).

[CJ14.1] Cheng, John; Grossman, Max; McKercher, Ty, 2014: *Professional CUDA® C Programming*, Indianapolis: Wiley.

[EA05.1] Einstein, Albert, 1905: *Zur Elektrodynamik bewegter Körper*, in: *Annalen der Physik und Chemie*, Band 322, Heft 10, S. 891-921.

[EA16.1] Einstein, Albert, 1916: *Die Grundlage der allgemeinen Relativitätstheorie*, in: *Annalen der Physik*, Band 354, Heft 7, S. 769 - 822.

[BS09.1] Brunier S.; European Southern Observatory (ESO), 2009: *The Milky Way panorama*, `https://www.eso.org/public/images/eso0932a/` (Abruf: 18.03.18).

[HS73.1] Hawking, Stephen W.; Ellis, George F. R., 1973: *The large scale structure of space-time*, Cambridge: Cambridge University Press.

[KR63.1] Kerr, Roy P., 1963: *Gravitational Field of a Spinning Mass as an Example of Algebraically Special Metrics*, in: *Physical Review Letters*, Band 11, S. 237 - 238.

[LD10.1] Louis, Dirk; Strasser, Shinja; Kansy, Thorsten, 2010: *Microsoft Visual C#: Das Entwicklerbuch*, Köln, O'Reilly.

[MC73.1] Misner, Charles W.; Thorne, Kip S.; Wheeler, John A., 1973: *Gravitation*, San Francisco: Freeman.

[NE65.1] Newman, Ezra T.; Couch, E.; Chinnapared, K.; Exton, A.; Prakash,

A.; Torrence, R., 1965: *Metric of a Rotating, Charged Mass*, in: *Journal of Mathematical Physics*, Band 6, S. 915 - 917.

[NG18.1] Nordström, Gunnar, 1918: *On the Energy of the Gravitational Field in Einstein's Theory*, in: *Verhandl. Koninkl. Ned. Akad. Wetenschap.*, Afdel. Natuurk., Amsterdam, Band 26, S. 1201 - 1208.

[PD11.1] Psaltis, Dimitrios; Johannsen, Tim, 2011: *A ray-tracing algorithm for spinning compact object spacetimes with arbitrary quadrupole moments. I. Quasi-kerr black holes*, in: *The astrophysical journal*, 745:1, 20. Januar 2012, doi:10.1088/0004-637X/745/1/1 (Abruf: 17.09.16).

[RH16.1] Reissner, Hans, 1916: *Über die Eigengravitation des elektrischen Feldes nach der Einsteinschen Theorie*, in: *Annalen der Physik*, Band 355, Heft 9, S. 106 - 120.

[SJ11.1] Sanders, Jason; Kandrot, Edward, 2011: *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Boston: Addison-Wesley.

[SK16.1] Schwarzschild, Karl, 1916: *Über das Gravitationsfeld eines Massenpunktes nach der Einsteinschen Theorie*, in: *Sitzungsberichte der Königlich-Preussischen Akademie der Wissenschaften*, Sitzung vom 3. Februar 1916, S. 189 - 196, Berlin: Deutsche Akademie der Wissenschaften.

[SK16.2] Schwarzschild, Karl, 1916: *Über das Gravitationsfeld einer Kugel aus inkompressibler Flüssigkeit nach der Einsteinschen Theorie*, in: *Sitzungsberichte der Königlich-Preussischen Akademie der Wissenschaften*, 1916, S. 424 - 434, Berlin: Deutsche Akademie der Wissenschaften.

[SP99.1] Schneider, Peter; Ehlers, Jürgen; Falco, Emilio E., 1999: *Gravitational Lenses*, Study Edition, 2nd printing, New York: Springer.

[TE91.1] Taylor, Edwin F.; Wheeler, John A., 1991: *Spacetime Physics: Introduction to Special Relativity*, 2nd Edition, New York: Freeman.

[VM08.1] Visser, Matt, 2008: *The Kerr spacetime: A brief introduction*, arXiv:0706.0622v3 (Abruf: 06.01.17).

[WR84.1] Wald, Robert Manuel, 1984: *General Relativity*, Chicago: The University of Chicago Press.