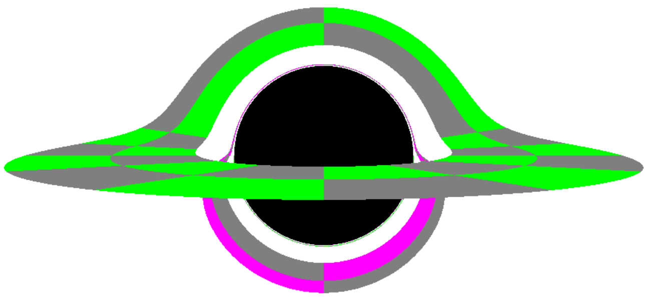
`cuRRay`: CUDA-Raytracer for Light Rays in relativistic Kerr-Newman Spacetime

Translation of the 2018 Swiss national contest (SJF) paper



Sébastien C. Garmier

January 3, 2019

Expert SJF

Marcel Balsiger

Originally submitted in January 2017 as Matura Project to
Kantonsschule Wohlen AG, Switzerland

Supervision Kantonsschule Wohlen

Adrien Cornaz, Ph.D.
Jan-Mark Iniotakis, Ph.D.

Contents

Abstract

`cuRRay` is a CUDA-accelerated raytracer for light rays in Kerr-Newman spacetime. `cuRRay` was programmed in C++, two versions exist: A CUDA version making use of the NVIDIA hardware of the system and a CPU version running on the CPU of the system. The software allows the configuration of scenes containing a Kerr-Newman black hole (described by mass, angular momentum and electric charge), an observer and optional objects (spheres, accretion disc, stars in the background). Using raytracing, images of the scene can be created from the perspective of the observer.

The images generated by `cuRRay` allow for two aspects of curved spacetime to be visualized: the deflection and redshift of light rays by gravity. We show the algorithms used in the software, as well as images created by the software. Using the generated images, we discuss the differences between types of black holes.

Introduction

The *general theory of relativity*, or simply *general relativity*, is a theory of gravity. It replaces the classical, Newtonian theory, when the latter loses accuracy due to high mass-energy densities. General relativity describes gravity as a consequence of *curved spacetime*. This idea is made mathematically precise using *differential geometry*. The theory was introduced by Albert Einstein in 1915.

It predicts the existence of *black holes*, regions in spacetime, out of which nothing can escape due to strong gravity. Black holes can only be observed indirectly, since not even light can escape out of them. They appear completely black.

Light of other bodies near a black hole is deviated by the extreme gravitational field. An observer seeing this light would see a heavily distorted image of the black hole. Such pictures of black holes and bodies around it can be used to determine the properties of the black hole and the spacetime surrounding it. D. Psaltis et. al. [PD11.1] for instance use simulated pictures of black holes, to determine the spectra of gas rotating around the holes.

Gravitation by C. Misner et. al. [MC73.1] and *General Relativity* by R. Wald [WR84.1] are modern and standard textbooks which are suitable for a rigorous introduction into general relativity. They provide the needed basics to understand the propagation of light in relativistic gravitational fields.

The software `cuRRay`, acronym for ***CUDA r**elativistic **Ray**tracer*, was developed in the context of this project. It simulates trajectories of single photons, light rays, in relativistic *Kerr-Newman spacetime*. Kerr-Newman spacetime describes the gravitational field around a rotating, electrically charged black hole. `cuRRay` calculates the trajectories of photons arriving at the observer backwards, until an object is hit. That way, the picture seen by the observer can be reconstructed. This procedure is called *raytracing*. `cuRRay` makes use of *NVIDIA® CUDA* to efficiently trace multiple photons in parallel using the graphics hardware.

CUDA by example by J. Sanders et. al. [SJ11.1] and *Professional CUDA C programming* by J. Cheng et. al. [CJ14.1] are suited as an introduction to CUDA programming.

The following goals were pursued: the learning of general relativity and the coding of the software `cuRRay`, capable of creating pictures of objects near black holes. There was no initiative to create a faster or better software than existing software. There are two reasons for this thinking. Firstly, the main goal was to learn general relativity and to apply it to some problem. It was not planned to use the software beyond this project. Although, given the final state of the software, it certainly could be employed in further research. Secondly, it is very hard to compare `cuRRay` to already existing software, as these are usually highly specialized.

In the main part we will discuss the necessary theory and explain the functionality of the software. Finally, we visualize curved spacetime around black holes using images created by `cuRRay` and discuss them.

Notation Conventions

Sign Conventions We use the metric signature $- + ++$.

Events and Objects We refer to events using script letters; for instance: $\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$, etc. Objects, like observers or bodies, are designated by uppercase, latin letters.

Tensors We use two different notations for tensors: the *coordinate-free notation* and the coordinate-dependent *index notation.*

Bold letters label tensors including vectors and covectors in coordinate-free notation. For instance, the *divergence of the energy-momentum tensor*:

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

The indices in this notation are also written in bold characters. They distinguish between different tensors of a set. For instance, the *basis vectors*:

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

If the components of a tensor are important, we will use the index notation. For instance, the *contraction of the Riemann-tensor*:

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

Indices of coordinate-dependent tensors indicate, which component of a tensor is meant. For instance, *mass-energy in the local Lorentz frame*:

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

Indices in Index Notation In index notation, greek indices take on the values 0 to 3. The index 0 is a temporal index, the indices 1 to 3 are spatial indices. From time to time we will denote specific values of indices by the letters for the coordinate bases. Depending on the index, or a combination of indices respectively, a component is either a *time-component*, a *space-component* or a mixture of the two.

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

for instance, is the metric tensor's time-time-component. Latin indices only designate space-components. They take on the values 1 to 3.

Compact index notation For derivatives we occasionally use the *compact index notation*:

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

and

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

We differentiate by all indices after the comma or the semicolon. If an index, by which we differentiate, is raised, a multiplication with the metric tensor after the differentiation is implied.

Einstein notation If not stated otherwise, we make use of the Einstein notation. We imply a sum over all indices, which occur both *covariantly* (lowered) and *contravariantly* (raised):

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

Four-vectors If not stated otherwise, we will use the following special relativistic *four-vectors* instead of the classical three-dimensional vectors (here, the components are given in local Minkowski coordinates):

four-position $\boldsymbol{x} = (ct, x, y, z)$	*four-momentum* $\boldsymbol{p} = m\boldsymbol{v}$
four-velocity $\boldsymbol{u} = d\boldsymbol{x}/d\tau$	*four-force* $\boldsymbol{F} = d\boldsymbol{p}/d\tau$
four-acceleration $\boldsymbol{a} = d\boldsymbol{v}/d\tau$	*four-current-density* $\boldsymbol{j} = (c\rho, j_x, j_y, j_z)$ $\rho =$ charge density;
(higher derivatives of $\boldsymbol{x}$)	$j =$ classical current density

Geometrized units We generally use geometrized units. The speed of light c and the gravitational constant G are set to 1:

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

Gaussian units In chapter 3 we will use *Gaussian units* for electromagnetic fields. That way, all ϵ_0's and μ_0's disappear.

Citations We will label citations using the following scheme: the two uppercase letters are the initials of the authors first and last name. The two-digit number are the final digits of the year of publication and a running number distinguishes between multiple publications of the same author in the same year. Example: [EA16.1] stands for Einstein, Albert (1916), first publication.

Chapter 1

General Relativity

In this chapter, we will set forth the general theory of relativity. We begin with a short discussion of *special relativity* followed by the *equivalence principle*, the basic principle upon which general relativity builds. Subsequently we will discuss the curved world lines of bodies in spacetime and describe them mathematically. Before finishing the chapter, we will establish the connection between spacetime curvature and mass-energy by introducing the Einstein field equations.

1.1 Special Relativity

Special Relativity describes the behaviour of physics as seen by different moving observers. It was published by Einstein in 1905 [EA05.1].

Postulates Special relativity is based on two postulates: The *postulate of relativity* says that the laws of physics must be the same for every *inertial* (non-accelerating) observer, the second postulate states that the *speed of light* is constant for every such observer. The speed of light c takes on the value

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

Lorentz Transformation The effects of special relativity on physics are most visible in the so-called *Lorentz transformations*. These transformations are used to transform coordinates and components of four vectors between *Minkowski coordinate systems* (Cartesian Coordinates plus time coordinate) of different inertial observers. A Lorentz transformation along the x-axis is given by the matrix

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

v is th relative velocity of the frames. γ is the *Lorentz factor*:

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

Lorentz transformations in different directions are also possible. One just uses the above matrix and combines it with rotations as required.

A detailed and illustrated introduction to special relativity is for instance given in [TE91.1].

1.2 Equivalence Principle

Weak Equivalence Principle The *weak equivalence principle* has already been known since Galileo Galilei. It states, that all bodies, regardless of their mass, shape or structure, fall with the same downward acceleration. In the rest frame of a falling body, other falling bodies will appear as moving inertially in straight lines, even though they are moving on parabolas as seen from an observer standing on the surface of earth. The principle is based on the equivalence of *gravitational mass* and *inertial mass.* [MC73.1, P. 13-19].

Strong Equivalence Principle In physics, we can detect force fields experimentally by observing the behaviour of test particles. Using this technique, we can for instance deduce the electromagnetic field from the movements of a charged particle. A freely falling observer however, cannot use test particles to measure the gravitational field, since, as seen in the previous paragraph, all of them will appear completely unaccelerated. For this reason, the gravitational field cannot be measured locally [1].

The *strong equivalence principle* states, that a falling observer is equivalent to an observer outside of all gravitational fields. He will not feel the gravitational field, since it cannot be measured. The strong equivalence principle is the basis of general relativity. [WR84.1, P. 66-67]. Figure 1.2.1 shows a thought experiment concerning the equivalence of two observers B_1 and B_2.

Let us turn to the upper part of the figure first: B_1 is freely falling in the gravitational field of earth and B_2 is located far away from any gravitational field. Because of the strong equivalence principle, both are equivalent to each other and they cannot measure a gravitational field. If B_2 sends out a light ray, it will propagate in a straight line, since nothing can deviate it. B_1 must observe the same effect. If B_1 sends off a light ray, it must shoot off in a straight line as well. This means, that the photons making up the ray have to fall together with B_1 at the same rate.

The lower part of the figure shows B_1 standing on the surface of earth. B_1 feels a normal force preventing them from falling towards earth's centre. The only change to the situation before is the normal force. If B_2 is accelerated by a rocket engine the two observers will again be equivalent, assuming the force due to the engine has the same magnitude as the normal force. Let us investigate the light ray sent off by B_2 perpendicular to the acceleration: since the photons are not accelerated by the engine, B_2 will see the ray bend

1. Using the relative acceleration of separated test particles the *tidal forces* of the gravitational field can be measured. However, this measurement is not local.

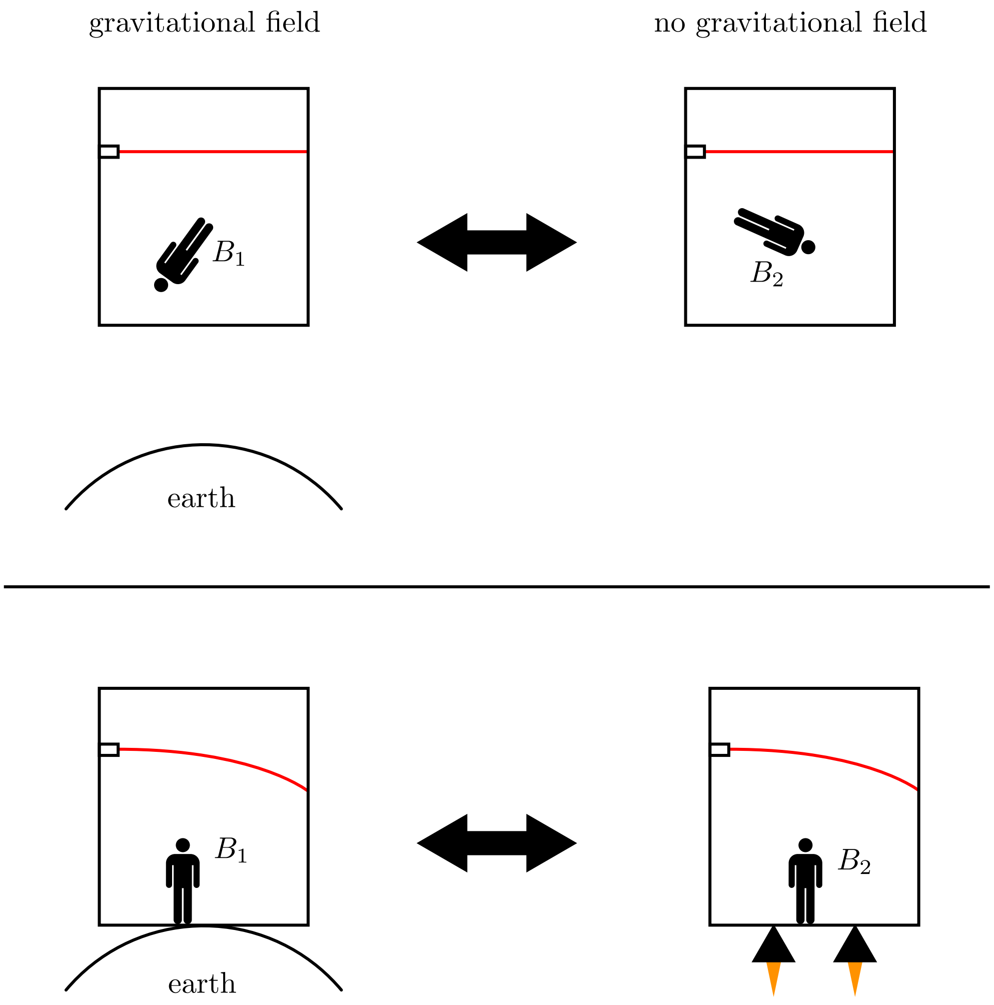


Figure 1.2.1: The observers B_1 and B_2 are equivalent („B“ stands for „Beobachter“, German for „observer“). Above, both observers are free of forces. Below, B_1 is accelerated by the normal force and B_2 is accelerated by a rocket engine. The red line represents a horizontally fired light ray as seen by the observers.

downwards. Because B_1 is equivalent to B_2, a similar light ray sent off by B_1 will also curve down as seen by B_1. The strong equivalence principle alone already leads to the deflection of light rays by gravity.

1.3 Curved Spacetime [2]

Spacetime In relativity, *spacetime* is the four-dimensional structure consisting of all points in three-dimensional space at any point in time. Points in spacetime are called *events*. In classical physics, spacetime is flat and world lines of inertial particles are always straight lines.

2. This chapter is based on [MC73.1, chapter 1].

Lorentz Frames The equivalence principle leads to an equivalence of falling observers, similar to the equivalence of inertial observers in special relativity. We call frames of reference of such observers *Lorentz frames* or *inertial frames*.

In contrast to the Lorentz frames of special relativity, the Lorentz frames of general relativity are only locally inertial; this means, tidal forces can be measured over finite distances. Local Lorentz frames have the property, that no acceleration of the frame can be measured from within; this is equivalent to the statement, that no gravitational field can be measured from within. Because of this, gravity is not considered a force in general relativity. So that a stationary observer relative to a gravitational field still perceives the trajectories of falling bodies to be curved (after all that's what we observe daily for objects being thrown in the air), Einstein suggested, that spacetime itself has to be curved. Such a curved spacetime would naturally lead to curved trajectories of falling objects, without the need of an acting force.

Special relativity as well as the pre-relativistic formulation of physics (excluding Newtonian gravity) are still valid in all Lorentz frames and thus also locally in curved spacetime, since a local Lorentz frame always exists.

Geodesics A falling observer is like an ant moving on a the curved surface of an apple: even though it crawls only forward, it will follow a curved trajectory due to the curvature of the apple's surface [3]. Analogous to this, a falling observer moves in a straight line locally, since they themselves cannot measure an acceleration. Globally, their path will be curved by spacetime.

The *straightest possible world lines* in curved spacetime are called *geodesics*. All freely falling objects move about on geodesics.

Mass-Energy Spacetime curvature is created by mass-energy, in contrast to the Newtonian gravitational field, which is generated only by mass. Figure 1.3.1 illustrates the curvature of spacetime due to a star and the resulting curved trajectory of an orbiting planet.

The *Einstein field equations* (see Chapter 1.10) describe the relation between mass-energy density and spacetime curvature. They determine how mass and energy curve spacetime, and how spacetime curvature influences the motion of mass and energy.

General relativity, the geometric interpretation of gravity, fulfils the equivalence principle. The theory is incredibly successful: numerous measurements and experiments confirm the theory today – not least the measurements of gravitational waves in September 2015 by the LIGO-detector (see [AB16.1]), a phenomenon predicted a century ago by Einstein [4].

Different approaches trying to describe gravity in the flat spacetime of special relativity failed all or lead indirectly to general relativity (see [MC73.1, Chapter 7] for an extensive overview over different failed approaches).

3. See [MC73.1, P. 3-5] for the short story „*The Parable of the Apple*“, which describes this concept wonderfully.

4. Many other measurements, like the perihelion precession of Mercury (see [WR84.1, P. 143]) or the deflection of light by the sun (see [WR84.1, P. 146]), confirm the theory.

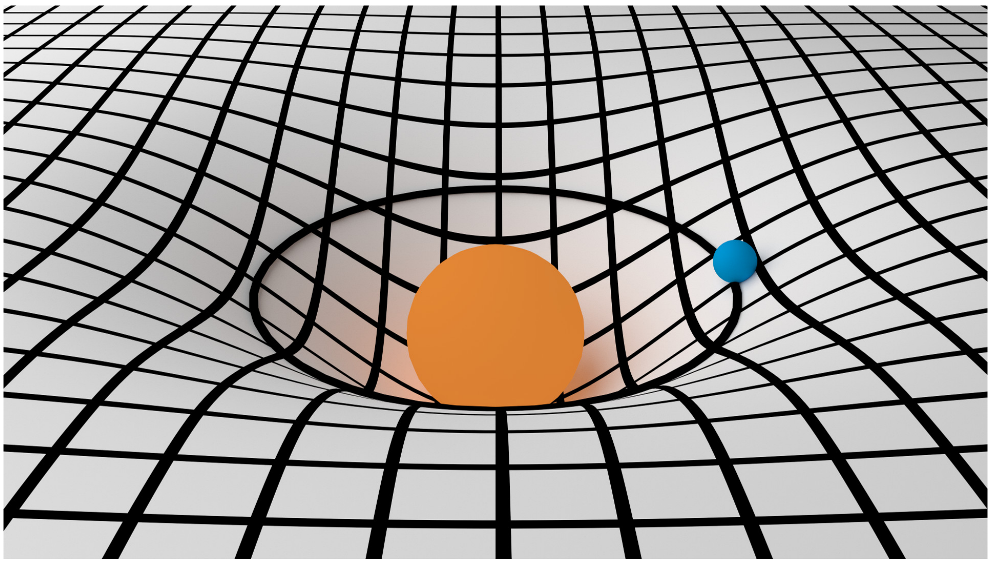


Figure 1.3.1: Spacetime, depicted as a two-dimensional surface, is being curved by the mass of the big, orange star in the centre of the figure. Therefore, the trajectory of the small, blue planet is curved. The curvature of spacetime due to the mass of the planet is neglected.

1.4 Tensors

In general relativity we use *Tensors* to describe physical quantities. Tensors are, for our purposes, generalizations of vectors and covectors. They were already proposed by Einstein in 1916 (see [EA16.1]).

Definition A rank (k, l) tensor is a multilinear function $\boldsymbol{A}$, which maps k covectors $\boldsymbol{\omega}$ and l vectors $\boldsymbol{v}$ to a real scalar s:

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

Tangent Space Tensors are bound to an event. All tensors except rank $(0, 0)$ tensors (scalars) operate on the elements of the *tangent space* and *cotangent space*, which in curved spacetime are different at every event. Only tensors operating on the vectors and covectors of the same (co-) tangent spaces can be compared. It is thus generally not trivial, to move tensors from one event in spacetime to another. See [WR84.1, P. 14-18].

Vectors are elements of the tangent space, covectors are elements of the cotangent space. However, both tangent and cotangent space are deeply connected by the *metric* (an *isomorphism*), see chapter 1.6.

The tangent space of an event can be visualised as a linear approximation of the curved spacetime at that event: Displacements around the origin of the tangent space correspond to displacements of first order in curved spacetime around the event. This idea shows in the definition of the tangent space: Its elements are tangent vectors to curves in curved spacetime. As seen before, one then arrives at the cotangent space using the metric. See [WR84.1, P. 14-18].

Indices As soon as it is known how a rank (k, l) tensor affects all (k, l) tuples of *basis vectors* and *basis one-forms* (*basis covectors*) of a coordinate system, one can then calculate the components of the tensor relative to that coordinate system (see [MC73.1, P. 53]):

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

Due to the linearity of tensors, equation (1.4.1) can be written as a product of components:

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

Upper indices on tensors are called *contravariant*, lower indices *covariant*. See [MC73.1, P. 75].

Tensor Product We can multiply two tensors together to form a new tensor of higher rank, according to the following scheme that we call the *tensor product*:

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

Here, the tensor product of a vector $\boldsymbol{v}$ and a covector $\boldsymbol{\omega}$ is shown.

Contraction A *contraction* is the vanishing of two indices of a tensor (or tensor product) due to summation. For example:

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

Gradient, Directional Derivative and Divergence These three operations (as well as other ones, not outlined here) are made possible by the *derivative operator* ∇ (see chapter 1.7):

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

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

3. divergence with respect to the index β_1
$$\nabla \cdot \boldsymbol{A} = \nabla_{\beta_1} A^{\beta_1 \dots \beta_k}{}_{\gamma_1 \dots \gamma_l}. \tag{1.4.8}$$

Note, ∇ is not a covector; the attached index only shows, that the result of the operation has one covariant index more than the operand.

1.5 Coordinates

Events and Coordinate Systems We distinguish different events in spacetime by giving them unique names. If the names follow a systematic naming convention, they form a *coordinate system*. We use tuples of numbers as coordinates. A coordinate system is *continuous*, if two infinitesimally neighbouring events differ only by an infinitesimal tuple of numbers. Sometimes, a coordinate system is mostly continuous, with the exception of certain regions called *coordinate singularities*. Continuous and mostly continuous coordinate systems are the coordinate systems we usually work with in physics. See [MC73.1, P. 5-13]

Differences in coordinates do not necessarily reflect actual distances between events. In flat spacetime it is possible to choose such coordinates, but in curved spacetime this is not generally true. It is even possible that not all of spacetime can be covered by continuous coordinates and that *coordinate singularities*, events with no or multiple coordinates, arise. In other words, in curved spacetime we usually have no choice but to choose mostly continuous systems. See [WR84.1, P. 11-14].

Local Coordinates We can always choose coordinates which are valid locally and, in that local region, correspond to the familiar coordinates of special relativity. This is true because local Lorentz frames always exist.

Special Covariance The laws of physics can be rewritten such that they take on the same form in all coordinate systems and in every Lorentz frame. This principle is known as the *principle of special covariance*. See [WR84.1, P. 58].

Coordinate Transformations Tensor components change in the following way under a coordinate transformation $\tilde{x}^\alpha(x^\beta)$:

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

For every additional covariant or contravariant index, we add a corresponding term. Equation (1.5.1) ensures that scalars of the form

$$S = A_{\alpha_1 \ldots \alpha_k} B^{\alpha_1 \ldots \alpha_k} \tag{1.5.2}$$

remain unchanged by coordinate transformations. If the laws of physics are written as equivalences of tensor products, special covariance is guaranteed. See [EA16.1, P. 779-780] and [WR84.1, P. 56-59].

1.6 Metric Tensor

The Geometry of Spacetime If the infinitesimal distances between a central event and all its neighbours are known, the local shape of spacetime is also known. Likewise, the shape of the entire spacetime can be known if all distances between mutually neighbouring events are known. The geometry of spacetime (the way it curves) is fully described by the distances between events. See [MC73.1, P. 309].

Metric Tensor The *metric tensor* $\boldsymbol{g}$ is a rank $(0,2)$ tensor defined at every event in spacetime. It defines the scalar product of two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ in the tangent space (see [MC73.1, P. 305]). It is a measure for length and angles; this is especially visible in Euclidean geometry:

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

The metric tensors of every event contain all the information about spacetime curvature.

Metric The rule telling us how to compute the scalar product is called the *metric.* It is often denoted as the squared length of an infinitesimal vector $\boldsymbol{dx}$:

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

here, $g_{\alpha\beta}$ are the components of the metric tensor. $\boldsymbol{g}$ is symmetric, that is, the indices α and β can be swapped without changing the value of the component: $g_{\alpha\beta} = g_{\beta\alpha}$. The metric tensor has 16 components, symmetry however only allows up to 10 different components. See [MC73.1, P. 310].

Metric in flat Spacetime In flat spacetime, the components of $\boldsymbol{g}$ relative to a Minkowski coordinate system are $g_{\alpha\beta} = \mathrm{diag}(-1,1,1,1)$ everywhere. The metric possesses a *Lorentz signature*; which means, that the sign of the time component is different from that of the space component. ds^2 then becomes the familiar infinitesimal spacetime interval dI^2 of special relativity.

Gravitational Potential In general relativity, the metric tensor is generally different at every event. The flat spacetime of special relativity is only an exception. The components $g_{\alpha\beta}$ of the tensor field $\boldsymbol{g}$ can be seen as the *potentials* creating the effects of gravity (see [MC73.1, P. 436]). When we speak of a *gravitational field* in general relativity we mean the tensor field $\boldsymbol{g}$.

Raising and lowering Indices Indices of vectors, covectors and tensors in general can be raised and lowered by multiplication with the metric tensor:

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

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

$g^{\alpha\beta}$ is the *inverse metric tensor* which is defined using the *Kronecker delta function* $\delta^\alpha{}_\gamma$:

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

For every vector, there is a corresponding covector and vice versa. This is the previously mentioned connection between tangent space and cotangent space. See [WR84.1, P. 25].

1.7 Geodesics

Definition An observer moves along a geodesic if they move locally inertially on a straight line. Their velocity vector does not change locally. Along the geodesic, that is at every event laying on the geodesic, the following applies for the tangent vector $\boldsymbol{u} = \boldsymbol{dx}/d\lambda$ of the geodesic:

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

The parametrization λ of the geodesic is arbitrary. See [MC73.1, P. 262-263]. Equation (1.7.1) is the abstract form of the *geodesic equation.*

Derivative and parallel Transport The geodesic equation (1.7.1) is defined via the *derivative operator* ∇. The directional derivative $\nabla_{\boldsymbol{u}}$ calculates the rate of change of an arbitrary tensor field $\boldsymbol{A}$ along $\boldsymbol{u}$ at the event $\boldsymbol{x^0}$. To do so, $\boldsymbol{A}$ has to be evaluated at the events $\boldsymbol{x} = \boldsymbol{x^0}$ and $\boldsymbol{x} = \boldsymbol{x^0} + \boldsymbol{dx}$:

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

where $\lim_{k\to 0} \boldsymbol{u}k = \boldsymbol{dx}$. Because $\boldsymbol{A}(\boldsymbol{x} = \boldsymbol{x^0})$ and $\boldsymbol{A}(\boldsymbol{x} = \boldsymbol{x^0} + \boldsymbol{dx})$ do not operate on the same (co-) tangent spaces, one of the tensors has to be parallelly displaced *following the curvature of spacetime*, that is along a geodesic. This procedure is called *parallel transport.* Generally, multiple geodesics exist between two events. In the limit where the events become neighbouring however, only one possible geodesic exists. This means that parallel transport along an infinitesimal distance is uniquely defined. See [MC73.1, P. 249].

Figure 1.7.1 shows the parallel transport of a vector $\boldsymbol{v}$. Before the vectors at x and $x + dx$ can be compared, $\boldsymbol{v}(x)$ has to be parallelly transported to $x + dx$, it then becomes $\boldsymbol{v}(x)_{||}$.

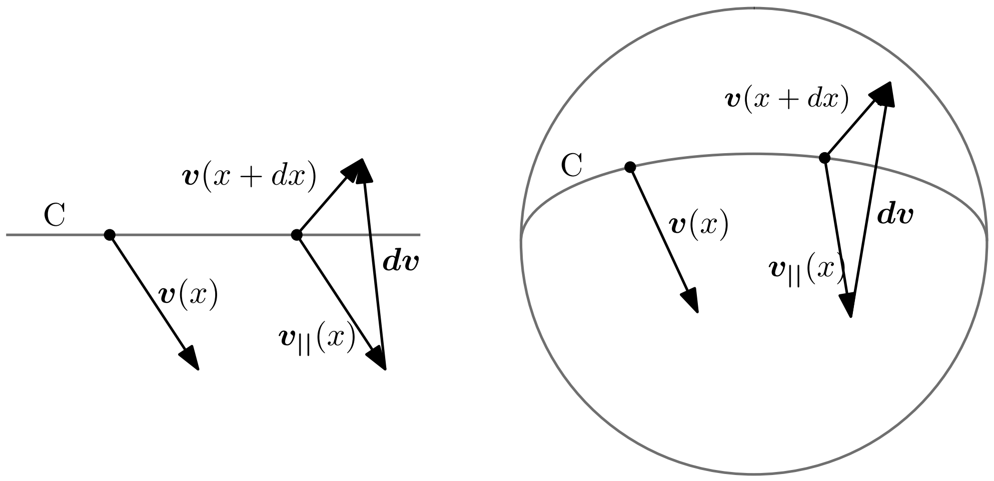


Figure 1.7.1: Geometric representation of parallel transport of a vector $\boldsymbol{v}$ *along a geodesic C in flat and curved spacetime.*

Properties of the Derivative Operator Certain rules hold for the derivative operator. Especially the *sum rule*

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

the *product rule*

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

and the *chain rule* (not shown) for all tensors $\boldsymbol{A}$ and $\boldsymbol{B}$. The sum rule only holds, if both tensors are of the same rank. See [MC73.1, P. 257-258].

Equivalence Principle From the equivalence principle we have

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

this is the mathematical formulation of the statement, that no gravitational field can be measured locally. See [WR84.1, P. 35].

Parallel Transport of Vectors The scalar product of two parallelly transported vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ remains unchanged by the transport:

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

The first term vanishes due to the equivalence principle, equation (1.7.5). The other terms vanish because both vectors are parallelly transported.

Especially the magnitude of a parallelly transported vector $\boldsymbol{v}$ remains unchanged:

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

Christoffel Symbols In index notation, the derivative operator is defined using the *Christoffel symbols* (also: *connection coefficients*) and the *ordinary derivative operator* ∂:

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

For every covariant or contravariant index, a corresponding term is added. $\Gamma^\alpha{}_{\beta\gamma}$ are *Christoffelsymbols* of second kind. They show how basis vectors of a coordinate system change, when they are parallelly displaced along themselves. Since the Christoffel symbols depend on the choice of coordinates, they are not components of a tensor and do not fulfil equation (1.5.1). Christoffel symbols can be calculated from derivatives of the metric tensor:

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

The Christoffel symbols are analogous to the force field of gravity in the classical theory. See [WR84.1, P. 35-36].

Geodesic Equations The geodesic equations for individual velocity components in index notation are:

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

See [MC73.1, P. 263]. The equations (1.7.10) are solved by `cuRRay` for photons in Kerr-Newman spacetime. The needed Christoffel symbols will be derived in appendix B.

Types Given the tangent vector u^α we distinguish types of geodesics depending on the sign of $g_{\alpha\beta}u^\alpha u^\beta$. A geodesic can never change its type. The product is negative for *timelike* geodesics, positive for *spacelike* geodesics and zero for *lightlike (null) geodesics*. Massive particles move on timelike geodesics and massless particles (for instance photons) on lightlike geodesics; no known particles move on spacelike geodesics. See [WR84.1, P. 44].

1.8 Riemann Tensor

Relative Acceleration of Geodesics The *Riemann curvature tensor* $\boldsymbol{R}$ is a measure for the relative acceleration of neighbouring, initially parallel geodesics. It thus measures the tidal gravitational field: since tidal fields are the only effects of gravity, one can measure in a freely falling frame according to the principle of relativity, the Riemann tensor is usually what is measured in such situations. Lets take a look at figure 1.8.1: a family $\gamma_n(\lambda)$ of geodesics pervade spacetime. Individual geodesics differ by a selection parameter n. For every geodesic $\boldsymbol{t} = \partial/\partial\lambda$ is the tangent vector and $\boldsymbol{\xi} = \partial/\partial n$ the *deflection vector*.

The second order change of $\boldsymbol{\xi}$ is the wanted relative acceleration of neighbouring geodesics. The acceleration can be expressed by the rank $(1,3)$ Riemann curvature tensor $\boldsymbol{R}$:

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

See [MC73.1, P. 265-270].

Path Dependency of parallel Transport Besides relative acceleration of geodesics, the Riemann curvature tensor also quantifies the path dependency of parallel transport. The *commutator* of derivatives along two coordinate vector fields $\boldsymbol{A}$ and $\boldsymbol{B}$ is give by:

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

Equation (1.8.2) produces a vector which shows how strongly $\nabla_{\boldsymbol{A}}\nabla_{\boldsymbol{B}}\boldsymbol{C}$ and $\nabla_{\boldsymbol{B}}\nabla_{\boldsymbol{A}}\boldsymbol{C}$ differ. Parallel transport along non-infinitesimal distances is therefore path dependant. See [MC73.1, P. 277-281].

Components The components of the Riemann curvature tensors are

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

See [WR84.1, P. 47-48].

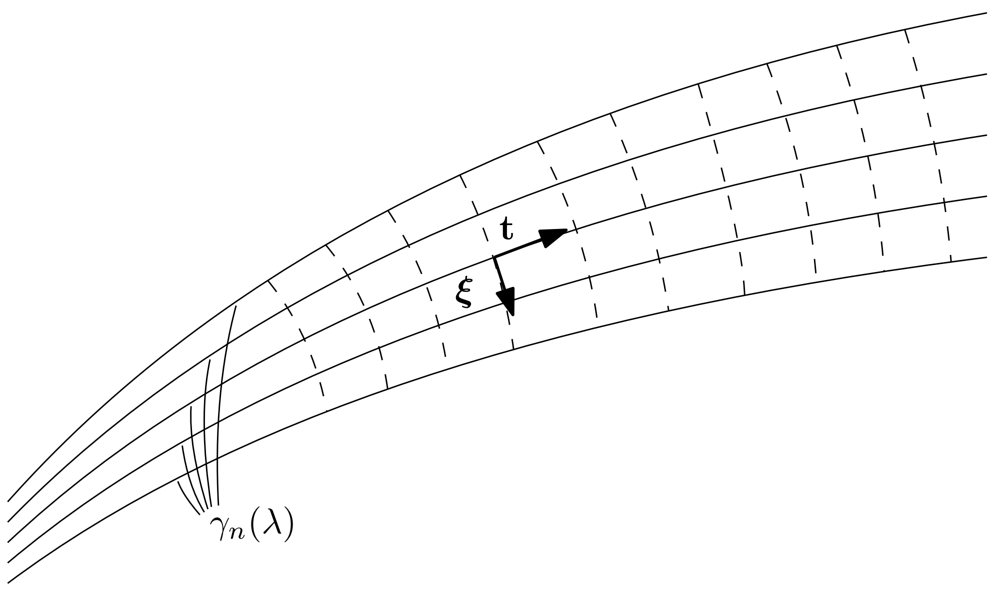


Figure 1.8.1: A family $\gamma_n(\lambda)$ of geodesics: the tangent vector $\boldsymbol{t} = \partial/\partial\lambda$ and the deflection vector $\boldsymbol{\xi} = \partial/\partial n$ are defined at every point on every geodesic. The second order change of $\boldsymbol{\xi}$ is the relative acceleration of neighbouring geodesics.

1.9 Stress-Energy-Momentum Tensor

Structure of the Stress-Energy-Momentum Tensor The symmetric rank $(0,2)$ *stress-energy-momentum tensor* $\boldsymbol{T}$ describes mass and energy at an event in spacetime. The energy of the gravitational field is not contained in $\boldsymbol{T}$.

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

describes different aspects of mass, energy, momentum, pressure and mechanical stress depending on the choice of the vectors $\boldsymbol{v}$ and $\boldsymbol{w}$. In a local Lorentz frame the components of $\boldsymbol{T}$ are:

$$\begin{aligned} T_{00} &\equiv \text{mass-energy density} \\ T_{0a} = T_{a0} &\equiv a \text{ component of momentum density} \\ T_{ab} = T_{ba} &\equiv ab \text{ component of the Maxwell stress tensor.} \end{aligned} \tag{1.9.2}$$

See [MC73.1, P. 131].

Conservation of Energy and Momentum For energy and momentum to be conserved, the divergence of $\boldsymbol{T}$ has to vanish:

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

This guarantees that no sources or sinks of mass-energy can exist. See [MC73.1, P. 146].

1.10 Einstein Field Equations

The *Einstein Field Equations* describe the interaction between spacetime curvature on one hand and mass-energy on the other. By solving the equations the indices $g_{\alpha\beta}$ of the metric tensor and thus the shape of spacetime can be calculated.

Ricci Curvature Tensor und Ricci Scalar By contracting the Riemann curvature tensor over the second and third index we arrive at the *Ricci curvature tensor*:

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

By repeated contraction we arrive at the *Ricci scalar*:

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

See [WR84.1, P. 40].

Einstein Tensor The *Einstein tensor* $\boldsymbol{G}$ can be obtained from the Ricci curvature tensor and the Ricci scalar:

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

$\boldsymbol{G}$ is symmetric and fulfils the *Bianchi identity*:

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

Field Equations The *Einstein field* equations are

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

or

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

respectively. See [WR84.1, P. 72].

Conservation of Energy and Momentum From the Bianchi identity $\nabla \cdot \boldsymbol{G} = 0$ follows that $\nabla \cdot \boldsymbol{T} = 0$. This leads to the desired finding, that energy and momentum are conserved. General relativity thus predicts the conservation of energy and momentum from geometric principles (under the assumption that the principle of relativity holds). See [MC73.1, P. 475].

Solving the Field Equations The Einstein field equations are second order differential equations. They must be solved simultaneously for the metric tensor and the stress-energy-momentum tensor, since both quantities are strongly linked. Because of their complexity, the field equations are only algebraically solvable in rare cases with high symmetry (see [WR84.1, P. 73]).

Chapter 2

Black Holes

In this chapter we will discuss two solutions to the Einstein field equations: the *Schwarzschild metric* and the *Kerr-Newman metric*. We will use both to describe black holes.

2.1 Schwarzschild Metric

The *Schwarzschild metric* is one of the few algebraic solutions to the field equations. Because of its high degree of symmetry, it is relatively easy to derive. The Schwarzschild metric describes spacetime around a *spherically symmetric* and *static* body. It can be applied to planets and stars, since those are usually sufficiently symmetric and static.

Vacuum and internal Metric The Schwarzschild metric was derived for both the vacuum outside the body [SK16.1] and the internals of the body [SK16.2] in 1916 by Karl Schwarzschild. We will only make use of the vacuum metric. From now on, we simply speak of the vacuum metric as the *Schwarzschild metric.*

Coordinates The Schwarzschild metric is defined in so-called *Schwarzschild coordinates.* The coordinates of an event $\mathscr{P}$ are

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

θ and ϕ are angular coordinates: θ is the zenith angle and ϕ the azimuth angle. The polar axis of the coordinate system is freely choosable. t is the proper time of an observer resting at infinity. r is the *radial Schwarzschild coordinate.* r does *not* measure the distance from the centre of the body, but is defined in the following way:

$$r = ([\text{Area of the sphere around } r = 0 \text{ passing through } \mathscr{P}]/4\pi)^{1/2}. \tag{2.1.2}$$

See [MC73.1, P. 596].

Metric The Schwarzschild metric depends on the mass M of the central body and the radial coordinate r. It is given by

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

See [MC73.1, P. 820].

Event Horizon Provided the surface of the central mass lies below $r = 2M$, at $r = 2M$, g_{tt} becomes zero and g_{rr} becomes infinite. It can be shown that this is not a physical singularity since none of the components of the Riemann tensor become infinite at $r = 2M$. $r = 2M$ is a mere *coordinate singularity*. See [WR84.1, P. 124].

$r = 2M$ is called the *Schwarzschild radius*. Earth for instance, has a Schwarzschild radius of $(2G \cdot 5.97 \cdot 10^{24}\text{ kg})/c^2 \approx 9\text{mm}$. A falling object requires an infinite coordinate time t to arrive at the Schwarzschild radius, but only a finite proper time.

The region inside the Schwarzschild radius is called a *black hole*. This region of spacetime is cut off from the rest of the universe at $r = 2M$ by the *event horizon*. All timelike and lightlike geodesics falling below the event horizon will never escape to $r > 2M$. They will all end at $r = 0$. This means that objects with a smaller radius than their own Schwarzschild radius will inevitably collapse to an infinitely dense point at $r = 0$. This collapse is called *gravitational collapse*. Because no light can escape from black holes, they appear completely black. See [WR84.1, P. 154-155].

Photon Sphere At $r = 3M$, unstable orbits of photons exist. The sphere $r = 3M$ is called *photon sphere* because of this. Photons in orbit either escape to infinity when nudged outwards or fall into the black hole when nudged inwards. See [WR84.1, P. 143].

Singularity After a gravitational collapse, the entire mass of the body is concentrated at $r = 0$ in an infinitely dense point [1]. $r = 0$ is at the same time a coordinate singularity and a *physical singularity*, since the components of the Riemann tensor blow up. See [WR84.1, P. 124].

Schwarzschild Black Holes The Schwarzschild metric describes the spacetime $r > 2M$ around a black hole with mass M. Such black holes are called *Schwarzschild black holes*. The software can create images of Schwarzschild black holes if the angular momentum and the electric charge are both set to zero. Schwarzschild black holes are useful for the visualization of many light deflection phenomena.

2.2 Kerr-Newman Metric

The *Kerr-Newman Metric* describes the spacetime of an axially symmetric mass M with angular momentum L and electric charge Q. The metric was

1. It is assumed that quantum-gravitational effects will play important roles at high densities. These however, are yet unknown.

found in 1965 by Ezra T. Newman [NE65.1]. It is a generalization of the *Reissner-Nordström metric* [RH16.1] (1916), [NG18.1] (1918) and the *Kerr metric* [KR63.1] (1963), which in turn are generalizations of the Schwarzschild metric.

No-hair Theorem Similar to the Schwarzschild metric, the Kerr Newman metric allows black holes. Those are called *Kerr-Newman black holes.* The *No-hair theorem* „Black holes have no hair“ [2] is metaphor stating that black holes have only three properties: the mass M, the angular momentum L and the electric charge Q. If the theorem is true, all types of stationary black holes can be described by the Kerr-Newman metric. For that reason, the Kerr-Newman metric was chosen for the application in `cuRRay`.

Coordinates We use the Kerr-Newman metric in so-called *Boyer-Lindquist coordinates* (BL coordinates for short). BL coordinates have the advantage of possessing *Killing vector fields* identical to the coordinate vector fields (See chapter 2.3). t and ϕ have the same meaning as in Schwarzschild coordinates. However, the polar axis is now the axis of rotation. The Kerr-Newman metric is not static because the central mass rotates, but it is *stationary.* This means that the components of the metric tensor are invariant with the coordinate time t (see [WR84.1, P. 119]).

The symmetry allows an arbitrary scaling of the geodesic parameter λ. During ray tracing, `cuRRay` flips time (scaling by -1) to calculate geodesics of photons backwards. The black hole now rotates in the opposite direction (see [HS73.1, P. 161]). Because of this, we have to be careful with the interpretation of images where $a \neq 0$. We will further investigate this difficulty in chapter 7.2.

r and θ are some sort of elliptic coordinates. We will discuss them later in more detail.

Metric The Kerr-Newman metric is given by

$$\begin{aligned} ds^2 = &-\left(\frac{\Delta - a^2\ \sin^2\theta}{\Sigma}\right) dt^2 - \frac{2a\ \sin^2\theta(r^2+a^2-\Delta)}{\Sigma} dtd\phi \\ &+\left(\frac{(r^2+a^2)^2 - \Delta a^2\ \sin^2\theta}{\Sigma}\right)\sin^2\theta\ d\phi^2 + \frac{\Sigma}{\Delta}dr^2 + \Sigma d\theta^2, \end{aligned} \tag{2.2.1}$$

$$\Sigma = r^2 + a^2\ \cos^2\theta, \tag{2.2.2}$$

$$\Delta = r^2 + a^2 + Q^2 - 2Mr, \tag{2.2.3}$$

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

The metric reduces to the Schwarzschild metric if $a = Q = 0$, to the Kerr metric if $Q = 0, a \neq 0$ and to the Reissner-Nordström metric if $a = 0, Q \neq 0$. See [WR84.1, P. 313-314].

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

Frame-dragging Effect In the vicinity of a rotating black hole, the spacetime is dragged in the direction of rotation because of the cross term $dtd\phi$ (see [MC73.1, P. 879]). This phenomenon is called *frame-dragging effect* or *Lense-Thirring effect*. Geodesics of particles are affected by this.

Event Horizons The Kerr-Newman metric possesses *two* event horizons (or one in the limiting case) at the positions $r_\pm$. They result from the equation $\Delta = 0$:

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

They are called *outer horizon* (r_+) and *inner horizon* (r_-) (see [WR84.1, P. 315]). If the square root in (2.2.5) becomes zero, then only one event horizon exists. Such a metric is called *extreme*. If equation (2.2.5) has no real solution, because $M^2 \leq a^2 + Q^2$, *no* event horizon exists. The singularity is *naked*. Because of the assumption of *cosmic censorship*, it is believed that no naked singularity can exist (see [WR84.1, P. 299-308]).

Ergospheres Another interesting property is the change of sign of g_{00}, that is $(a^2\ \sin^2\theta - \Delta)/\Sigma = 0$ at $r_{E\pm}$:

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

r_{E+} is called the *outer static limit*, because no static observers can exist in the range $r_{E+} > r > r_+$. All observers are dragged in the direction of rotation. The spacetime between r_+ and r_{E+} is called the *outer ergosphere*. The nearer an observer gets to r_+, the quicker their orbit must be in order to stay at $r = const$. Similarly, r_{E-} is called the *inner static limit* and the spacetime between r_- and r_{E-} is called the *inner ergosphere*. See [MC73.1, P. 894].

Photon Orbits Photon orbits also exist in Kerr-Newman spacetime (see [CC13.1]). We are however not going to study them in detail.

Singularity At

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

we have a singularity (see [WR84.1, P. 314]). If we interpret r and θ as spherical coordinates, the singularity would only appear for $r = 0$ and $\theta = \pi/2$ but not for other values of θ. This is a contradiction, since an event requires the same metric tensor, no matter which direction we are looking from. If we interpret r and θ as Boyer-Lindquist coordinates, it becomes apparent that the singularity is actually a *ring* around the axis of rotation. See [WR84.1, P. 314-315].

Cartesian Coordinates The Kerr-Newman metric is *asymptotically flat* (see [HS73.1, P. 161]), which means that spacetime becomes more and more flat, the further away from the black hole an observer is. Due to this property, we can cover the spacetime with a Cartesian coordinate system, which becomes inaccurate in the vicinity of the event horizon, but further out reflects actual distances and angles. We use Cartesian coordinates for geometrical vector calculations in `cuRRay`:

$$x = \sqrt{r^2 + a^2}\sin\theta\cos\phi,$$

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

(see [VM08.1, P. 15]). In these coordinates, the ring singularity is centred around the origin and has the radius a (see [HS73.1, P. 162-163] and [WR84.1, P. 314-315]).

Calculations in Cartesian Coordinates We will use Cartesian coordinates in `cuRRay` to compute intersections of light rays with spheres. On one hand, this simplifies computation, as we can use ordinary vector geometry. On the other hand, it introduces errors, as these coordinates reflect true spacetime structure only far away from the black hole.

We accept this inaccuracy, as construction of spheres in curved spacetime would certainly exceed the scope of this project. Even tough the deviations should remain small in most cases, images including spheres have to be interpreted carefully.

Only the collision detection between spheres and rays is concerned by this inaccuracy. The rays themselves as well as their intersection with accretion discs are computed in BL coordinates.

Visualization We can visualize the Kerr-Newman metric with Cartesian coordinates. Figure 2.2.1 shows a cut ($t = \phi = \text{const}$) through the metric $M = 0.9$, $Q = 0$ and $a = 0.85$ of a black hole rotating around the z-axis. Figure 2.2.2 shows a similar cut for the extreme case $M = a = 0.9$ and $Q = 0$. In both figures, the faint lines represent the curves $r = \text{const}$ and $\theta = \text{const}$. The Cartesian coordinates are labelled along the borders. The two black dots represent the cut through the infinitely thin ring singularity. The red curves represent the event horizons and the black ones the static limits.

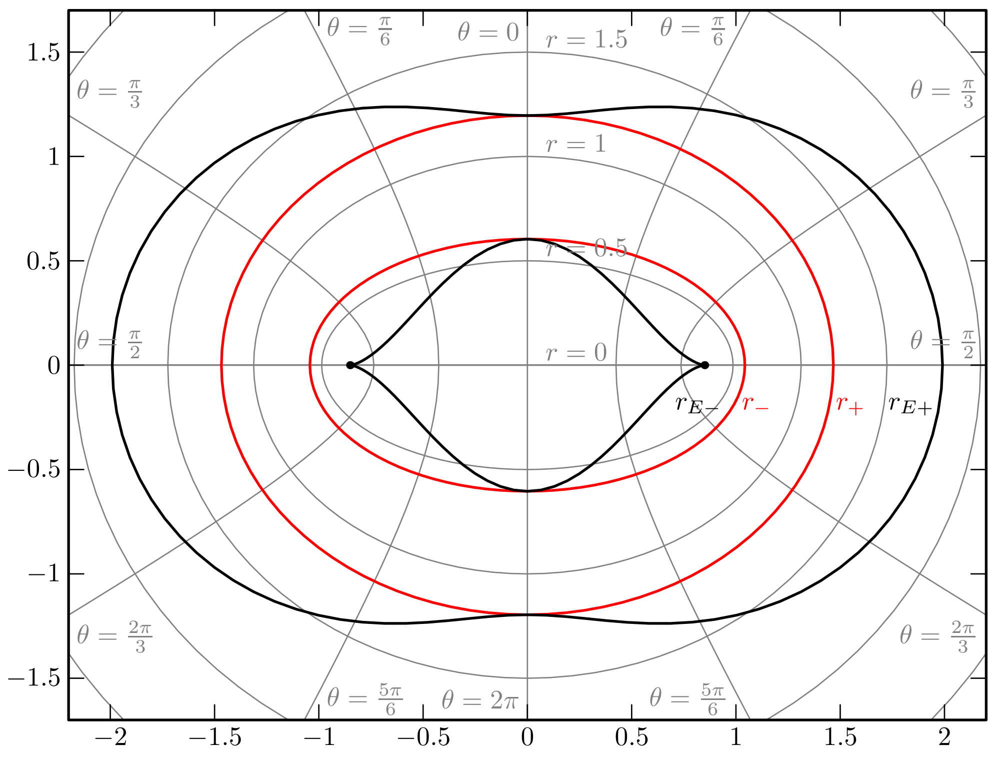


Figure 2.2.1: Kerr-Newman metric for $M = 0.9$, $Q = 0$ and $a = 0.85$. The ellipse $r = 0$ collapses to a line between two points on the ring singularity.

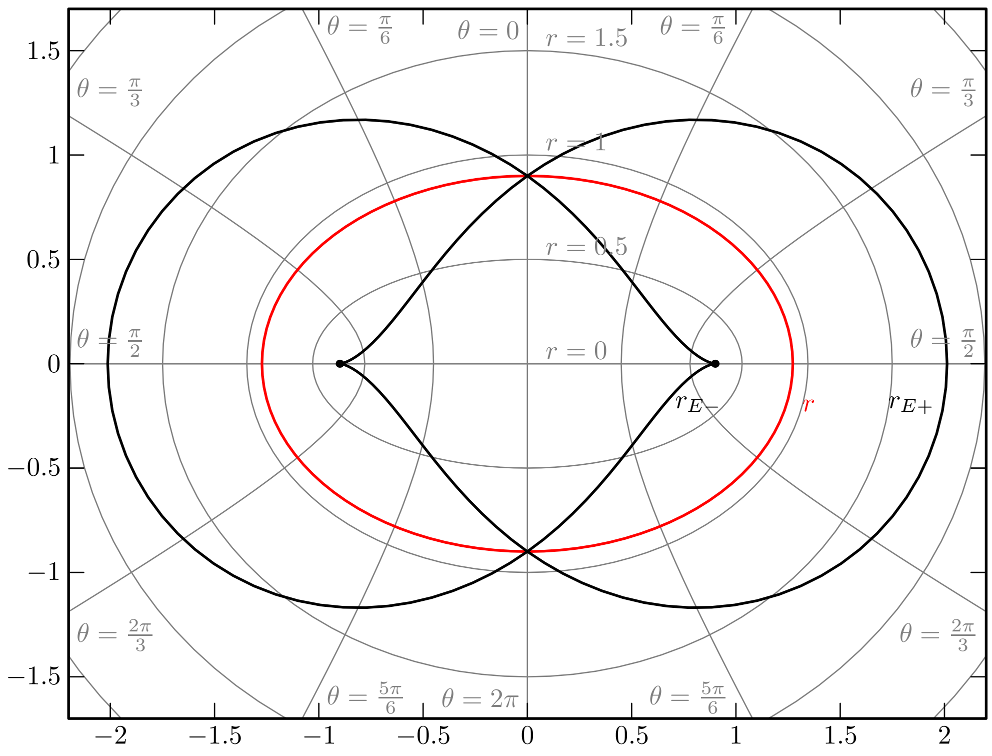


Figure 2.2.2: Extreme Kerr-Metric in Cartesian coordinates for $M = 0.9$, $Q = 0$ and $a = 0.9$. The coordinate singularities are the same as in figure 2.2.1. Only one event horizon exists.

2.3 Killing Vector Fields and Constants of Motion

Killing Vector Fields If every event $\mathscr{P}$ in the spacetime is infinitesimally displaced by $\boldsymbol{dx}(\mathscr{P})$ without changing the metric, then a *Killing vector field* exists which is the *generator* of the infinitesimal displacement. We have:

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

Every vector field fulfilling the *Killing equation*

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

is a Killing vector field. Killing vector fields embody the symmetries of spacetime. See [MC73.1, P. 652].

Constants of Motion The scalar product between the tangent vector $\boldsymbol{u}$ and the Killing vector along a geodesic remains constant. This product is called the *constant of motion*. Since the geodesic can be chosen freely, we have everywhere

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

If the Killing vector field corresponds to coordinate vector field ∂_K, we have in addition

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

See [MC73.1, P. 651].

In a suitable coordinate system, we can thus simplify the geodesic equation using constants of motion. We will use this simplification in `cuRRay`.

2.3.1 Killing Vector Fields in Kerr-Newman Spacetime [3]

The Kerr-Newman metric possesses two Killing vector fields. They correspond to the time-translational symmetry and the rotational symmetry.

Energy The Killing vector field $\xi^\alpha = (1, 0, 0, 0)$ gives in BL coordinates the following constant of motion:

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

E is proportional to the energy of a particle on the geodesic, as seen by an observer at infinity.

Angular Momentum The Killing vector field $\eta^\alpha = (0, 0, 0, 1)$ gives in BL coordinates the following constant of motion:

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

L is proportional to the angular momentum of a particle on the geodesic, as seen by an observer at infinity.

3. This chapter is based on [WR84.1, P. 313, 320]

2.3.2 Geodesic Equations of Kerr-Newman Spacetime

In this chapter, we show the geodesic equations of Kerr-Newman spacetime, which will be integrated by `cuRRay`. We will begin with the geodesic equations of the form (1.7.10):

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

The geodesic equations form a *coupled system of ordinary, second order, partial differential equations.*

First Order Equations The Killing vector fields ξ^α and η^α allow us to simplify the equations of motion for the t and ϕ components of the position. The equation for the t component becomes

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

and the one for the ϕ component

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

The initial position x_0^α, E and L alone determine the equations of motion (2.3.8) and (2.3.9). We derive these equations in appendix A.

Second Order Equations The equations for the t and θ components of position remain second order equations. However, we can remove all Γ terms which become zero. The equation for the r component becomes

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

and the one for the θ component

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

Where $u^\alpha = dx^\alpha/d\lambda$. A list of expressions for the Christoffel symbols used in (2.3.10) and (2.3.11) are also given in appendix B.

Chapter 3

Light

In this chapter, we take a look at light propagation in curved spacetime.

3.1 The Electromagnetic Field

The *electromagnetic field* affects and is generated by electrically charged particles. It is responsible for the *electromagnetic force*, one of the four fundamental forces of nature. The electromagnetic field allows the existence of waves that propagate at the speed of light: *electromagnetic radiation.*

Faraday Tensor The rank-(1,1) *Faraday Tensor* $\boldsymbol{F}$ describes the electromagnetic field. In a local Lorentz frame, it can be constructed from the components of the electric vector field $\boldsymbol{E}$ and the magnetic vector field $\boldsymbol{B}$:

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

See [MC73.1, P. 71-73].

Lorentz Force The force, which the electromagnetic field exerts on a charged particle of charge q, is the called the *Lorentz force*:

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

where $\boldsymbol{u}$ is the velocity of the particle and τ is its proper time. See [MC73.1, P. 73].

Maxwell Equations The *Maxwell equations* describe the dynamics of the electromagnetic field. They take on a simple form, when written in terms of the Faraday tensor:

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

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

See [MC73.1, P. 568].

Wave Equation The electromagnetic field is created by a *vector potential* $\boldsymbol{A}$:

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

Equation (3.1.4) can be rewritten using $\boldsymbol{A}$. This is how we get the *wave equation* [1]:

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

The wave equation (3.1.6) allows the existence of periodically changing potentials. Such potentials form a wave. The term $R^{\alpha}{}_{\beta}A^{\beta}$ incorporates spacetime curvature into the wave equation. This term makes the search for solutions of the wave equation more difficult. See [MC73.1, P. 569].

3.2 Electromagnetic Waves

Possible solutions of the wave equation (3.1.6) are *electromagnetic waves* travelling at the speed of light: *light.* Waves in curved spacetime are very similar to the electromagnetic waves observed in flat spacetime, with the only difference being the Ricci-term leading to discrepancies. If we however choose the wavelength short enough, this discrepancy becomes negligible. See [MC73.1, P. 571].

Under the assumption of a short enough wavelength, we arrive at the following conclusions [2]:

Wave Vector An observer with velocity $\boldsymbol{u}$ measures the *angular frequency* ω of an electromagnetic wave to be

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

$\boldsymbol{k}$ is called the *wave vector.* It points in the direction of the spread of the wave.

Null Geodesics For the wave vector we have in particular:

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

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

$\boldsymbol{k}$ is a *null vector* and is being transported parallelly along itself. $\boldsymbol{k}$ is also proportional to the velocity $\boldsymbol{u}$ of the wave, therefore:

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

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

Electromagnetic radiation moves along null geodesics. These geodesics are commonly called *light rays.*

1. At the same time the *Lorenz gauge condition* $A^{\alpha}{}_{;\alpha} = 0$ is applied.
2. For a rigorous derivation, see [MC73.1, chapter 22.5]

3.3 Photons [3]

Since light rays propagate along geodesics, it is possible to describe light using particles moving on these geodesics. Such particles are called *photons*. `cuRRay` solves the geodesic equations of photons.

Rest Mass Photons have no rest mass because they move on null geodesics.

Momentum The momentum $\boldsymbol{p}$ of a photon is given by

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

$\boldsymbol{p}$ points in the same direction as the velocity $\boldsymbol{u}$.

Energy The energy of a photon is

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

3.4 Deflection of Light Rays

Figure 3.4.1 shows schematically how the light rays γ_1 and γ_2 coming from a source S are deflected by a Schwarzschild black hole H by the angles α_1 and α_2. In this two-dimensional diagram, the observer O sees two images (I_1 and I_2) of the source. In three dimensions a ring appears, when the source is exactly behind the black hole.

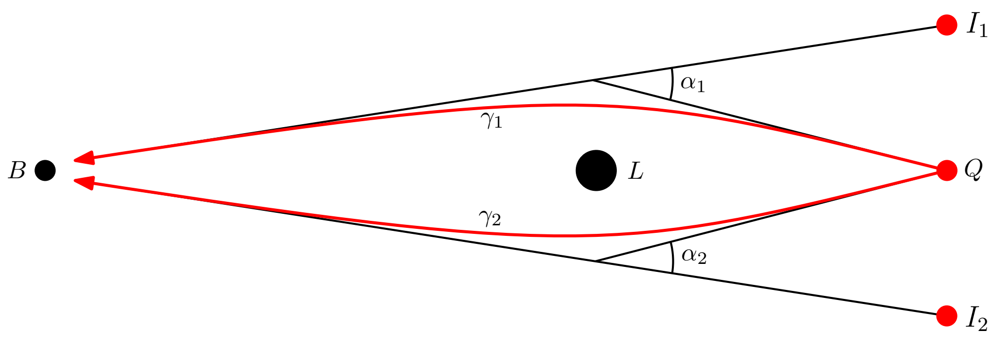


Figure 3.4.1: The Schwarzschild black hole H deflects the light rays γ_1 and γ_2 coming from the source S, such that the observer O sees the images I_1 and I_2.

Light rays from different parts of an extended source are deflected differently in general. This leads to a distortion of the source's image.

The phenomena described in this chapter are those we will see in the pictures generated by `cuRRay`.

3.5 Gravitational Redshift

Light signals sent by an observer B_s (sender) and received by an observer B_r (receiver) are subject to *gravitational redshift*. Differences in the metric at the

3. This chapter is based on [SP99.1, P. 98-100].

positions of B_s and B_r lead to changes in the period T of the electromagnetic waves and thus to a shift in angular frequency ω. If the wavelength increases, the phenomenon is called *redshift*, if the wavelength decreases, it is called *blueshift*.

Ratio of Frequencies In `cuRRay` we are interested in the ration of angular frequencies ω_r/ω_s. From the equations (2.2.1) and (3.2.1) we arrive at the expression of this ratio in Kerr-Newman spacetime:

$$\frac{\omega_r}{\omega_s} = \frac{\left. u^t\sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}} \right|_r}{\left. u^t\sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}} \right|_s}. \tag{3.5.1}$$

See appendix C for a derivation of this equation

As described more fully in chapter 6, `cuRRay` calculates $\boldsymbol{u}$ backwards along the geodesic and utilises the starting value $\boldsymbol{u}_r$ in a process called *raytracing.* $\boldsymbol{u}_s$ is known after raytracing. The software then employs equation (3.5.1) to calculate the gravitational redshift.

Here it should be noted, that equation (3.5.1) only works for stationary observers. For rotating black holes, the equation becomes unphysical if the sender is within the ergosphere, since no stationary observers can exist there. Furthermore the equation also cannot deal with sources inside the ergosphere. `cuRRay` thus only computes redshift for sources outside the ergosphere.

Chapter 4

Overview of `cuRRay`

In this chapter, we give an overview over the software `cuRRay` (**CU**DA **r**elativistic **Ray**tracer). `cuRRay` is the main object of this work and was programmed from scratch.

4.1 Functionality

`cuRRay` can create images, so-called *frames*, of black holes and objects in Kerr-Newman spacetime, like the image of an *accretion disc* around the black hole (see further down).

Two version of `cuRRay` were programmed: The CUDA version and the CPU (central processing unit) version. The CUDA version requires a supported NIVIDIA graphics card and uses CUDA to accelerate computation. It is generally faster than the CPU version. The CPU version does not require a graphics card and uses the CPU for computations.

Raytracing Raytracing is the drawing of an image by simulation of single light rays. For each pixel of the frame to create, `cuRRay` traces the corresponding light ray backwards through Kerr-Newman spacetime until an object is hit, the ray escapes to infinity or the ray crosses the event horizon. This way, both the colours and redshifts of pixels can be calculated.

Scene The software reads information about the scene from a YAML file [1], the *scene file*. This information contains the parameters of the metric, the position, orientation and field of view of the observer. In addition, an accretion disc can be configured, spheres can be placed and a sky can be chosen. Finally, the scene file allows certain special pixels to be coloured differently: The colours of the event horizon, the sky (when no sky image is chosen) and erroneous pixels can be configured. Certain parameters in the scene file can be linearly animated over multiple frames by specifying a start and an end value.

Spheres `cuRRay` allows spheres to be placed in the scene. Spheres receive

1. YAML is an easily readable markup language which can also be easily read by computers. See `http://yaml.org/`.

a chessboard pattern coloured in grey and a configurable accent colour. The radius of each sphere is also configurable. Eight sphere can be placed at most.

Accretion Disc Infalling gas often accumulates in a disc around the equator of the black hole, the so-called *accretion disc*. An accretion disc can be placed in the scene in the plane $\theta = \pi/2$. To model the disc, the inner and outer radii as well as two accent colours (one for each side) can be chosen. If only one colour is selected, the lower side receives the complimentary colour of the top. Like spheres, the disc is also covered by a chessboard pattern.

Starry Sky One can choose an image file to be projected onto the sky. It is thus possible to draw a starry sky behind the black hole.

Frame Information Frame information is sent to `cuRRay` via the command line. It contains the dimensions in pixels of the frames to be produced, the amount of frames to create, the path to the scene file, the output directory and the type of the output.

System Configuration It is possible to specify a *system configuration file* in the command line. This file is also a YAML-file. It defines which GPUs (CUDA version) or how many CPU cores (CPU version) should be used. Also it provides useful options for console and log output as well as for the ray-tracer. Some options can be overwritten in the command line. A default system configuration is chosen when none is specified.

Output `cuRRay` produces different types of output depending on the configuration. Firstly, it can produce a text file containing all information about the scene and the frames it just calculated. Secondly, pixel colours can be saved in a PNG image file for every calculated frame (this is the type of output we are most interested in). Thirdly, the redshift data for every pixel can be stored in addition PNG files, again one per frame. Lastly, `cuRRay` can create a CSV file with detailed pixel data, one per frame. All files are numbered with the frame they belong to.

4.2 System Requirements

Platform `cuRRay` was written in C++ [2] and uses only platform-independent code libraries. Because of this, `cuRRay` can be compiled on most 64-bit operating systems. See appendix G for more information about building cuRRay. The Git repository (see appendix F) contains the source code and binaries for Windows 10 and Linux Debian 8.

Processor and OS Architecture A 64-bit processor as well as a 64-bit operating system are required.

Memory `cuRRay` requires a minimum of 1 GB of memory. However, drawing large frames on the CPU might require more.

2. `isocpp.org/`

CUDA Device A NVIDIA® graphics processor with support for CUDA 8.0 and *compute capability* 3.0 is required for the CUDA version [3]. Furthermore, 500 MB of VRAM are required. The CPU version does not require a graphics card.

Disk Space The required disk space depends on the operating system. For Windows 10, around 3 MB are used. In addition, space for the generated images is required. We recommend 1 GB of hard drive space.

4.3 Used Code Libraries

The following code libraries were used. The indicated version is the one used in development. Older or never version might possibly not work with `cuRRay`.

Boost 1.61 [4] Boost is an extensive collection of additions to the C++ standard library. The library is freely available.

C++ 14 Standard Library [5] The standard library, version 14, is freely available together with C++.

libpng 1.6.26 [6] libpng is the official library to work with PNG files. It is freely available.

NVIDIA® CUDA 8.0 Runtime [7] The CUDA 8.0 runtime is required to use the functionality of CUDA devices. The library is freely available, but an NVIDIA® CUDA device is required.

pngIO 1.1 [8] pngIO is a library developed by the author. The library is freely available and directly contained in the source code of `cuRRay`.

TCLAP 1.2.1 [9] The templatized C++ command line parser (TCLAP) allows the reading of the command line. The library is freely available and included in the source code of `cuRRay`.

yaml-cpp 0.6.0 [10] yaml-cpp allows the reading and writing of YAML files. The library is freely available.

4.4 Quick Start Guide

A quick start guide can be found in appendix E.

3. See `https://developer.nvidia.com/cuda-gpus` for a detailed list of graphics cards supporting compute capability 3.0.
4. `boost.org`
5. `cplusplus.com/reference/` and `cppreference.com/`
6. `libpng.org`
7. `developer.nvidia.com/cuda-toolkit`
8. https://gitlab.com/misc-util/pngIO
9. `http://tclap.sourceforge.net/`
10. https://github.com/jbeder/yaml-cpp

Chapter 5

Program Structure

In this chapter, we discuss the program structure of `cuRRay`. This includes the design choices made concerning input, output and parallel code execution.

5.1 Input and Output

User Interaction The user interacts via a text console with the software. This has the advantage of the software being independent of graphical user interfaces and operating systems. `cuRRay` accepts information through the command line, but mostly through the YAML files discussed earlier.

Logging The software copies the entire console output including error messages into a log file. This log file can be checked after the termination of the program. Also logging can be configured in certain ways, as mentioned later on.

File Formats The software utilises YAML (`.yml`) for configuration files, ASCII text files (`.txt`) for output information files, PNG files (`.png`) for images and CSV files (`.csv`) for detailed pixel information.

5.2 Parallel Code Execution

`cuRRay` utilizes two different types of processors: *CPUs* (central processing units) and *GPUs* (graphical processing units). Both allow parallel execution of code by themselves, which is a feature we use in `cuRRay`. `cuRRay` makes use of NVIDIA® CUDA to access the GPU of a computer system alongside its CPU.

5.2.1 CPU [1]

CPU stands for *central processing unit*. Every computer has at least one CPU. CPUs are specialized in carrying out complex and heterogeneous calculations.

Cores Most modern CPUs possess multiple *cores*, which is why they are also called *multi-core CPUs*. Each core can execute code independently of the others. CPUs are mostly designed for *functional parallelism*, that is problems where different types of tasks are worked on simultaneously.

Threads In most modern operation systems the developer does not have to worry about writing code for single cores. Instead they write so-called *threads*. The operating system then decides how threads are executed in parallel on the CPU. Often a thread is only running for a very short amount time at the time: the operating system switches quickly between the execution of different threads; that way, more threads than the number of cores can be executed. The quickly alternating serial execution of threads combined with the physical parallelity of modern CPUs gives the illusion of many threads being executed in parallel.

Processes Threads belong to a *process*. Usually, a process is assigned to every running program [2]. Each process has one thread to start with, the so-called *main thread*. Others can be started at runtime. Figure 5.2.1 shows a possible arrangement of processes and threads.

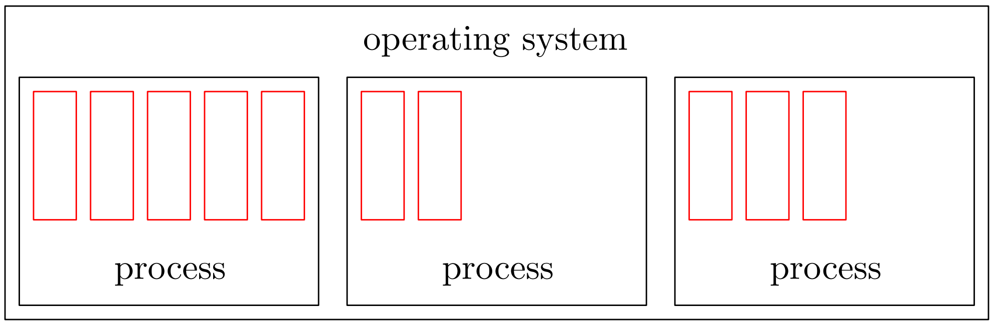


Figure 5.2.1: Possible combination of processes and threads. Threads are represented by red rectangles.

CUDA Version The CUDA version uses the *main thread* to read command line input and manage GPUs. This thread then starts new threads, one for each GPU in use. These threads make sure that all GPUs are working in parallel.

CPU Version The CPU version also uses the main thread to read input. It then creates one thread per CPU core to be used in the calculations. These secondary threads compute the images in parallel.

1. The first four sections of this chapter are based on [LD10.1, Chapter 37].
2. However, a program can often create additional processes.

5.2.2 GPU [3]

GPU stands for *graphics processing unit.* GPUs are usually cards or chips connected to the motherboard of a computer. They were originally designed to handle graphics calculations. However, they can also solve different tasks. This universality is called *GPGPU* (**G**eneral **p**urpose computation on **g**raphics **p**rocessing **u**nit).

GPUs can be considered as small computers in many ways: they contain one or more graphics processor (also known as *stream processors* in CUDA) and they have a special kind of memory, so-called *VRAM* (video ram).

CUDA Cores Graphics processors are quite different from CPUs. GPUs have a much larger number of cores than CPUs. The cores of a GPU are called *CUDA cores* in CUDA. They have difficulties with complex algorithms containing many logical branches compared to the cores of a CPU. They however excel at completing a large number of similar task in parallel. This type of parallelism is called *data parallelism*, since the cores are working on different pieces of data as opposed to different tasks.

Kernel Programs which are executed in graphics processors are called *kernels.* Kernels can be launched from a traditional CPU program. A kernel's code is usually executed many times in parallel. CUDA provides two divisions to organise the parallel instances of a kernel.

Blocks The first division is called *blocks.* Blocks form a three-dimensional, virtual grid. The developer has to choose the dimensions of this grid. Each GPU has a maximum grid size which can be determined at runtime.

A block is never spread over more than one graphics processor. However, one graphic processor can handle multiple blocks at the same time.

Threads Blocks are further divided into threads. Each block of a kernel has the same amount of threads. Threads are arranged like blocks in a virtual three-dimensional grid, whose dimensions have to be provided by the developer as well. Each thread is executed on a separate CUDA core. This is why the number of threads per block should not exceed the number of CUDA cores per graphics processor. Threads of the same block can communicate and synchronize with each other. Threads are always grouped in so-called *warps* of 32. Figure 5.2.2 shows a possible arrangement of blocks and threads inside a kernel. Chapter 6.1 deals with the dimensions of the grid used by `cuRRay`.

These and further restrictions decide the optimal grid configuration for a given problem. The *CUDA occupancy calculator* [4] can be used to find the optimal configuration. In chapter 6.1 we will discuss the dimensions of the grids used in `cuRRay`.

Applications `cuRRay` utilises CUDA for raytracing. This job's data is highly parallel; thus, an implementation using CUDA is suitable. The photons, that

3. This chapter is bases on [CJ14.1, chapter 3].
4. `developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls`

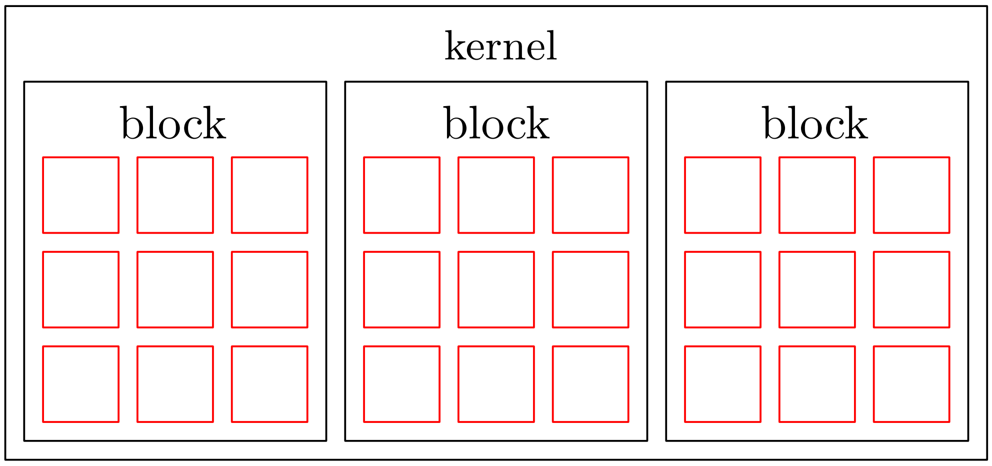


Figure 5.2.2: Possible dimensions of the block and thread grid within a kernel. The threads are represented as red squares.

is pixels, are independent of each other. This allows us to assign a thread to each photon. The blocks now need to be sized such that the load is evenly spread over the entire GPU.

Chapter 6

Raytracing

As described in chapter 4.1, `cuRRay` uses a raytracing algorithm to calculate the colour and redshift of each pixel. In this chapter, we outline this algorithm.

Basic Idea For each pixel, `cuRRay` calculates the corresponding light ray backwards. Since the Kerr-Newman metric allows time reversal [1], we can calculate light rays backwards in the same way we would calculate them forwards. A light ray is uniquely determined by its starting position and initial direction. In our case, where light rays are being calculated backwards, they are thus uniquely determined by the position of the observer (their final position) and the direction of the ray incoming at the observer (their final direction).

We think of our reversed light rays as rays originating at the observer with initial velocities set to the negative of the incident velocities of the real rays. We calculate these new light rays forwards to find the sources of the real rays. Another way to think of this, is to imagine the observer sending out rays for each pixel. The reversed light rays obey the geodesic equations. We thus have to solve these.

6.1 Grid Layout

We describe here, how the algorithm is parallelly spread over the CUDA cores of the GPU to reach maximum efficiency.

GPUs Because of slow data transfer between different GPUs, only one GPU is used per image. If a system contains more than one GPU however, multiple frames can be rendered in parallel.

Kernels, Threads and Blocks The raytracing algorithm consists of a single kernel, which integrates the geodesic equations for every pixel. We will investigate the kernel further down.

A single thread is used per pixel, that is, per light ray. The task is split into multiple blocks as to achieve maximum efficiency. To calculate the most

1. As seen in chapter 2.2, only the direction of rotation of a rotating black hole is then reversed. All other properties of spacetime remain the same.

efficient grid structure, `cuRRay` applies the following procedure [2]:

1. Every thread requires a number of so-called *registers*, storage space for variables used in the calculations. `cuRRay` calculates the number of registers required per warp while taking into account that registers are always organised in groups of 256 per warp (this restriction is called *register granularity*).
2. The number of registers per graphics processor is limited. `cuRRay` therefore calculates the maximum number of concurrently running warps per graphics processor. Warps always come in groups of four (*warp granularity*), this is also taken into account by the software.
3. The maximum amount of warps per block is then calculated. Here, further hardware limits and warp granularity are taken into account.
4. If more warps fit a block than can be executed simultaneously on a graphics processor, one block per graphics processor is chosen. Its size is chosen such that all threads run in parallel while maximizing the occupation of the processor. The next step is skipped.
5. If a maximally sized block cannot occupy the whole processor, `cuRRay` splits the maximum amount of warps per processor into 2, 3, 4 and 5 evenly sized blocks and takes the split that has the highest occupation. The block size is computed according to the chosen split.
6. The threads (one for every pixel) are then spread across multiple blocks of the size previously calculated.

Leftover Threads Generally, more threads are started than required, because every block has the same size. Those threads are immediately terminated. At most one block contains such leftover threads.

6.2 Frames

The scene file is loaded before raytracing starts. `cuRRay` creates a stack with every frame to draw.

CUDA Version Every activated GPU can fetch frames from the top of this stack. The frame is loaded into VRAM and the explicit values of animated settings are computed. Then the frame is calculated by running the raytracing kernel. Output files are subsequently created according to the settings. The GPU can then fetch another frame. This way, all frames are computed by potentially multiple GPUs (multiple frames are computed in parallel, while multiple pixels of the same frame are also computed in parallel).

2. The exact code is found in the source code file `gpuManager.cpp`.

CPU Version The CPU version does not complete the stack in parallel, but in serial. The current frame is loaded into regular memory and animated values are computed. Next, the raytracing algorithm is being run in parallel using different threads (that is, multiple pixels of the same frame are calculated in parallel). Output happens in the same manner as for the GPU version. Once a frame is terminated, the next one is fetched until the stack is empty.

6.3 Initial Value Problem

The geodesic of a photon is uniquely determined by curvature of spacetime on one hand and by a starting position and velocity, called the *initial values*, on the other hand.

System of Differential Equations The equations (2.3.8), (2.3.9), (2.3.10) and (2.3.11) form a system of differential equations that determines the influence of spacetime curvature on the geodesics. This system is integrated by `cuRRay`.

Initial Values This system of differential equations has infinitely many solutions, especially one for every possible light ray in spacetime. To find the solution corresponding to the light ray we are interested in, we need *initial values*: Initial position and velocity.

The initial position $x^\alpha(\lambda = 0)$ of every (backwards) photon is the position of the observer. The initial velocity $u^\alpha(\lambda = 0)$ can be calculated from the position and orientation of the observer. From these initial values we can calculate E and L according to the equations (2.3.5) and (2.3.6).

Initial Value Problem The initial values $x^\alpha(\lambda = 0)$, $u^\alpha(\lambda = 0)$ together with the system of differential equations forms a so-called *initial value problem.* This problem is solved by `cuRRay` using integration for each light ray.

6.4 Computation of the Initial Velocity

For every frame, `cuRRay` computes the initial velocities $u_0^\alpha = dx^\alpha/d\lambda$ $(\lambda = 0)$ of the light rays from the position, the orientation and field of view of the observer as well as the frame size.

Inertial Reference Frame We calculate these velocities first in the local inertial reference frame of the observer. We choose the flat Minkowski $\eta_{\alpha\beta} = \mathrm{diag}(-1, 1, 1, 1)$ for this frame. This means, that the spatial part of every vector is a Cartesian vector.

Since `cuRRay` only handles static observers, we cannot place observers inside the ergosphere.

Construction of the Velocity Vectors We imagine Cartesian coordinates in our frame. The observer is centred at the origin and looks down the x axis. We span a rectangular screen in front of the observer, evenly divided into pixels. The resolution of the screen reflects the dimensions of the frame.

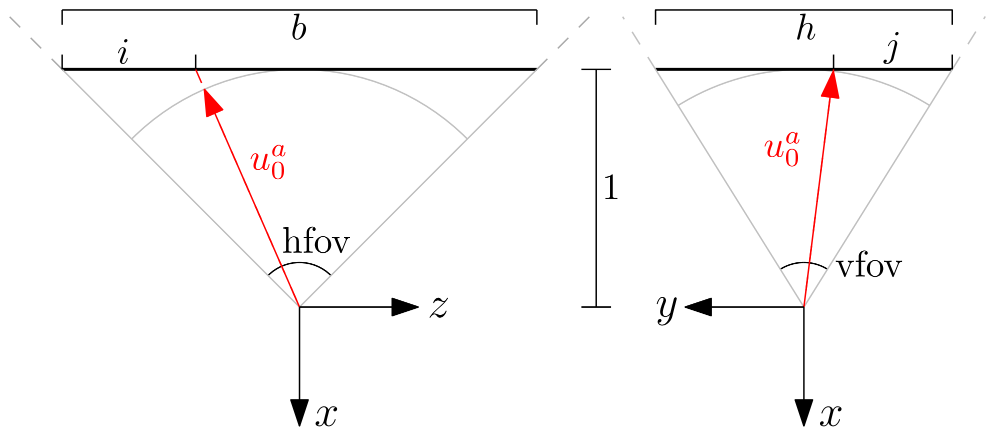


Figure 6.4.1: Construction of the initial velocity vector in Cartesian coordinates for the pixel (i, j). b and h are the width (b stands for „Breite“, „width“ in German) and height of the frame. The observer resides at the origin.

The screen is parallel to the yz plane and is located at $x = 1$. The rays passing through the origin and the corners of the screen limit the field of view. The with and height of the screen is now scaled as to match the field of view angles specified in the scene file (hfov and vfov).

The coordinates of the current pixel on the screen is computed. The spatial part of the initial velocity vector for that pixel is collinear to the position vector of the point on the screen at these coordinates. We normalise the length of the spatial part to one and set the time component to one as to create a null vector.

Figure 6.4.1 shows the construction of the initial velocity vector for the pixel (i, j).

Orientation The coordinate vector fields ∂_r, ∂_θ and ∂_ϕ are all perpendicular to each other, as we can see from equation (2.2.1). It is therefore possible to orient the local Cartesian coordinates in the following manner: The basis vector of the x axis is chosen parallel to ∂_r and points the same way. The same can be done for the basis vector of y with ∂_θ and the basis vector of z with ∂_ϕ. Figure 6.4.2 shows the orientation of the local Coordinates. This orientation leads to the observer pointing at the black hole by default.

The observer can additionally be rotated about the three Cartesian axes by *roll*, *pitch* and *yaw* angles in this order: First the frame is rolled around the negative x axis, then it is pitched around the negative z axis and finally it is yawed around the negative y axis. These angles can be configured and animated in the scene file. The three rotation are applied to all initial velocity vectors after they have been computed.

Coordinate Transformation Before the vectors can be used in raytracing, they need to be transformed to BL coordinates. We are looking for a coordinate transformation ${\Psi^\alpha}_\beta$ that transforms our vectors in local Cartesian coordinates

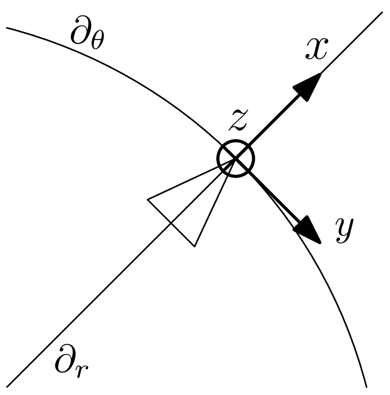


Figure 6.4.2: Orientation of the local Cartesian coordinates relative to the BL coordinates. We show a cut $\phi =$ konst through spacetime. The Cartesian coordinate basis, two BL coordinate vector field lines and the screen of the observer are shown.

to global BL coordinates:

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

The transformation needs to orient the Cartesian basis vectors as described earlier and needs to make sure the observer is stationary. For a derivation of this transformation, see appendix D.

The transformed vectors are then used as initial values in the raytracing algorithm.

6.5 Fourth Order Runge-Kutta Procedure

The initial value problem described above cannot be solved analytically. To solve it and calculate $\boldsymbol{x}(\lambda)$, we need to integrate the differential equations numerically.

Single Step Procedure If

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

is the initial value problem and $y(x)$ is the wanted solution, we can approximate the continuous function $y(x)$ by discrete values y_i at the positions x_i (see [BI78.1, P. 709]).

To this end, we introduce a potentially variable *step size* h, such that

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

y will then be given by the *difference equation*

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

at the discrete positions. $\hat{f}$ is called the *step function.* It depends on the algorithm. All numerical methods operating according to (6.5.3) are called *single* *step procedures*, since every discrete value y_i only depends on the value of the previous step. See [BI78.1, P. 709].

Errors The lowest order of the error in h due to approximation determines the accuracy of the algorithm. The higher the order, the better the numerical approximation. Usually, we can only guess the total, absolute error (If we happened to know the accurate solution, the numerical approximation would be rather useless). See [BI78.1, S. 709].

RK4 Procedure To solve the initial value problem, we use the *fourth order Runge-Kutta algorithm* (short: RK4). RK4 is a single step procedure with error of order h^5.

The step function for the RK4 algorithm is given by

$$\begin{aligned}
\hat{f}(x_i, y_i) &= \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\
K_1 &= f\left(x_i, y_i\right) \\
K_2 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}K_1\right) \\
K_3 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}K_2\right) \\
K_4 &= f\left(x_i + h, y_i + hK_3\right). \qquad (6.5.4)
\end{aligned}$$

$\hat{f}$ is a weighted average of f at different sub-steps. See [BI78.1, P. 710].

Solving the Geodesic Equations The geodesic equations are coupled, that is, they have to be integrated simultaneously since they depend on each other. We achieve this by calculating each sub-step for every equation before continuing with the next sub-step.

First Order Equations The equations (2.3.8) and (2.3.9) are of the form $y' = f(y)$. Those can easily be integrated with RK4. A single step in x^t is calculated as follows:

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

A step in x^ϕ is computed analogously:

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

$$\begin{aligned}
X_2^\phi &= \phi'\left(x_i^\alpha + \frac{h}{2}X_1^\alpha\right)\\
X_3^\phi &= \phi'\left(x_i^\alpha + \frac{h}{2}X_2^\alpha\right)\\
X_4^\phi &= \phi'\left(x_i^\alpha + hX_3^\alpha\right).
\end{aligned} \tag{6.5.6}$$

The prime symbols denote derivatives by λ. t' and ϕ' are given by the equations (2.3.8) and (2.3.9).

Second Order Equations The equations (2.3.10) and (2.3.11) are of the form $y'' = f(y, y')$. To get y, we need to integrate twice. Those are also coupled to the other equations and thus need to be integrated simultaneously as well.

A step in x^r is given by

$$\begin{aligned}
x_{i+1}^r &= x_i^r + h\frac{1}{6}(X_1^r + 2X_2^r + 2X_3^r + X_4^r)\\
X_1^r &= u^r\\
X_2^r &= u^r + \frac{h}{2}U_1^r\\
X_3^r &= u^r + \frac{h}{2}U_2^r\\
X_4^r &= u^r + hU_3^r
\end{aligned}$$

$$\begin{aligned}
u_{i+1}^r &= u_i^r + h\frac{1}{6}(U_1^r + 2U_2^r + 2U_3^r + U_4^r)\\
U_1^r &= r''\left(x_i^\alpha, u_i^\alpha\right)\\
U_2^r &= r''\left(x_i^\alpha + \frac{h}{2}X_1^\alpha, u_i^\alpha + \frac{h}{2}U_1^\alpha\right)\\
U_3^r &= r''\left(x_i^\alpha + \frac{h}{2}X_2^\alpha, u_i^\alpha + \frac{h}{2}U_2^\alpha\right)\\
U_4^r &= r''\left(x_i^\alpha + hX_3^\alpha, u_i^\alpha + hU_3^\alpha\right)
\end{aligned} \tag{6.5.7}$$

A step in x^θ is given by

$$\begin{aligned}
x_{i+1}^\theta &= x_i^\theta + h\frac{1}{6}(X_1^\theta + 2X_2^\theta + 2X_3^\theta + X_4^\theta)\\
X_1^\theta &= u^\theta\\
X_2^\theta &= u^\theta + \frac{h}{2}U_1^\theta\\
X_3^\theta &= u^\theta + \frac{h}{2}U_2^\theta\\
X_4^\theta &= u^\theta + hU_3^\theta
\end{aligned}$$

$$u_{i+1}^\theta = u_i^\theta + h\frac{1}{6}(U_1^\theta + 2U_2^\theta + 2U_3^\theta + U_4^\theta)$$

$$\begin{aligned}
U_1^\theta &= \theta''\left(x_i^\alpha, u_i^\alpha\right) \\
U_2^\theta &= \theta''\left(x_i^\alpha + \frac{h}{2}X_1^\alpha, u_i^\alpha + \frac{h}{2}U_1^\alpha\right) \\
U_3^\theta &= \theta''\left(x_i^\alpha + \frac{h}{2}X_2^\alpha, u_i^\alpha + \frac{h}{2}U_2^\alpha\right) \\
U_4^\theta &= \theta''\left(x_i^\alpha + hX_3^\alpha, u_i^\alpha + hU_3^\alpha\right)
\end{aligned} \tag{6.5.8}$$

r'' and θ'' are given by the equations (2.3.10) and (2.3.11).

The equations (6.5.5), (6.5.6), (6.5.7) and (6.5.8) describe the algorithm used by `cuRRay` for a single RK4-step.

6.6 Step Size Control

We control the step size h with an algorithm used in [PD11.1]. h adjusts to the fastest changing coordinate:

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

c is configurable positive constant. The software uses $c = 1/32$ by default, but other values can be set in the system configuration file.

To make sure that h is sufficiently small, we can make c smaller and smaller. As soon as the image does no longer change, when further shrinking c, the constant is small enough. $c = 1/32$ was tested to be accurate enough for almost all thinkable scenarios.

6.7 Monitoring the Velocity Vector

Since photons travel on null geodesics, their velocity vectors will always have magnitude zero. We note that any part of the sum in equation (2.2.1) divided by the rest always equals -1 (unless we divide by zero, in which case it is undefined). Comparing against -1 rather than 0 allows for the calculation of a relative error, which would not have been possible otherwise. We use this property to monitor the accuracy of the raytracing algorithm. Here, the time-time component is used to divided the rest, since this component only becomes zero outside the event horizon exactly on the ergosphere. The chance of a photon reaching exactly this radial distance in one of the steps is however extremely small. The error e is calculated as follows:

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

e shows whether the raytracing algorithm is correct and whether the step size control is effective. e is calculated for every step. For every light ray the average error $\bar{e}$ as well as the standard deviation σ_e of the error is computed. These values are stored in the detailed pixel data file. We will discuss the accuracy of `cuRRay` in chapter 8.1.

6.8 Break Conditions

We stop the integration of a geodesic, if one or more break conditions are met. Depending on the type of condition, the corresponding pixel is coloured differently.

Event Horizon The r coordinate of the event horizon is r_+ and given by equation (2.2.5). If the r coordinate of the photon falls below $r_+ + \epsilon$, we stop the integration. ϵ is a configurable positive constant which prevents the step size from getting arbitrarily small near the horizon. ϵ is 10^{-2} by default. Pixels fulfilling this condition are coloured using the event horizon colour.

Scene Boundary We choose the scene boundary $r_{\max}$ to include all objects and the observer. Also, the boundary can be set manually to a higher value when choosing a sky image. If the r coordinate of a photon exceeds $r_{\max}$, we colour the pixel in the sky colour. If a sky image is configured, the colour is computed according to chapter 6.10.

Errors If an error occurs during a RK4-step, the pixel is coloured in the error colour. An error occurs, when either $r < 0$, $\theta < 0$ or $\theta > \pi$. We will see reasons for such errors in chapter 8.1.

Spheres The raytracing algorithm checks after every RK4 step, whether one of the spheres was hit. This collision detection happens in Cartesian coordinates for the reasons mentioned above. The pixel is coloured according to the pattern on the sphere.

Accretion Disc If the accretion disc is hit (determined by a similar collision detection as for spheres, only that this one happens in BL coordinates), the pixel is coloured according to the pattern on the disc.

6.9 Redshift

`cuRRay` computes the ratio of receiver frequency ω_r (frequency measured by observer) and sender frequency ω_s (frequency of the source) for every light ray according to equation (3.5.1). The computation is only performed if the source lies outside the ergosphere. Possible values for the ratio range from 0 to ∞.

If redshift output is configured, an additional PNG file is created for every frame to store redshift data. The pixels of this file are coloured according to:

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

Each colour component can take on values from 0 (no intensity) to 255 (maximum intensity). A completely red pixel means infinite redshift, a completely blue one infinite blueshift and a purple pixel no shift at all. Pixels corresponding to sources inside the ergosphere are coloured green and pixels of the sky black.

6.10 Starry Sky

One can configure an image in the scene file to be projected onto the sky. This allows for instance the depiction of a starry sky behind the black hole.

Image File `cuRRay` requires a *rectangular projection* of the sky in form of a PNG file. The θ and ϕ coordinates of the sky sphere correspond directly to x and y coordinates in the rectangular projection of the sky:

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

h is the height of the image file and b the width. θ is the zenith angle, ϕ the azimuth. Figure 6.10.1 shows a rectangular projection of the night sky, which we will use in chapter 7 to create realistic looking images of black holes.

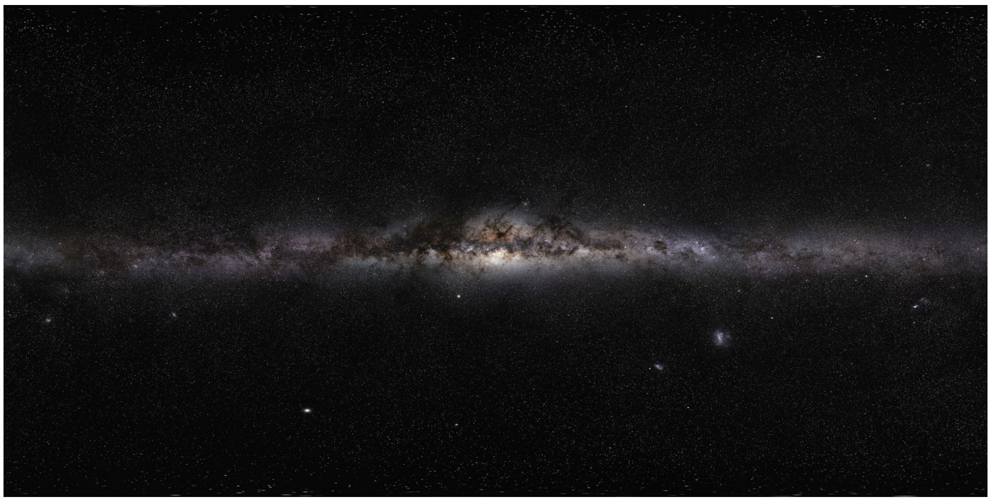


Figure 6.10.1: Rectangular projection of the night sky. Stars close to the galactic pole appear stretched due to the projection. Image source: ESO/S. Brunier, [BS09.1].

Asymptotic Flatness Since Kerr-Newman spacetime is asymptotically flat, we can find an r coordinate R, such that light rays outside of R are no longer deflected remarkably.

R can be configured in the scene file. If R is larger than the scene boundaries, we set $r_{\max}$ (see chapter 6.8) to $r_{\max} = R$. If $R < r_{\max}$, the scene boundary stays at $r_{\max}$ and R is ignored.

Direction Vector We assume that light rays outside the scene boundaries are no longer deviated strongly. We can thus approximate them by straight lines in flat spacetime. We further assume the size of the sky sphere to be infinite, i.e. that the stars are infinitely far away. This is a perfectly plausible assumption as distances between stars are many times larger than typical distances within the scene.

As seen from the sky sphere, every photon seems to be coming right from the centre of the sphere, as the scene appears infinitely small from this distance. We therefore only care about the velocity vector of light rays leaving the scene boundary. We transform this velocity vector from BL coordinates to global Cartesian coordinates using equation (1.5.1):

$$u^\alpha_\mathrm{C} = \sum_\beta \frac{\partial x^\alpha_\mathrm{C}}{\partial x^\beta_\mathrm{BL}}\, u^\beta_\mathrm{BL}, \tag{6.10.2}$$

We calculate the partial derivatives from the equations (2.2.8).

By normalising this Cartesian vector we get the *Cartesian direction vector*:

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

here, $|u^a_\mathrm{C}|$ is the length of the spatial part of u^a_C. Note the Latin indices, indicating that they only take on the values $\{1, 2, 3\}$.

Sky Point We calculate the angular coordinates θ and ϕ of the point in the sky pointed at by the direction vector in the following way:

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

Using the equations (6.10.1), we can find the coordinates of the pixel in the sky image and colour the pixel in our frame accordingly.

6.11 Complete Pixel Data

As already mentioned in chapter 4.1, we can choose whether `cuRRay` should produce a CSV file with detailed pixel information for each frame. Each pixel gets a row in the file. The following columns are written: x and y coordinates of the pixel, R, G and B colour components of the pixel, the amount of RK4 steps that were required, the ratio ω_r/ω_s, the mean error $\bar{e}$ and the standard deviation σ_e of the error.

If the redshift was not computed (because the source was inside the ergosphere), the ratio is stored as -1. The redshift is also stored, even if the pixel is erroneous; In this case, the stored value is undefined.

Chapter 7

Results

In this chapter, we will present and interpret different images created with `cuRRay`. All images and corresponding scene files are shown at the end of the corresponding section. The exact parameters of the image can be read off the scene files. Appendix E contains a user guide and explains how to interpret information in the scene files. All images and their corresponding scene file are contained in the Git repository. See Appendix F.

7.1 Schwarzschild Black Holes

Images of Schwarzschild black holes are suited for the visualisation of many curvature phenomena, as they are relatively simple to interpret due to the spherical symmetry of the black hole. Here we show several images generate by `cuRRay` of such black holes.

Accretion Disc The figures 7.1.1 and 7.1.2 show a Schwarzschild black hole with accretion disc (top side green, bottom side magenta) from two different observer positions. Listing 7.1.1 is th scene file used for both images.

In figure 7.1.1, the disc appears only slightly distorted because the observer is close to its axis (observer at $\theta = 5°$). The different parts of the disc are thus distorted symmetrically. Close to the horizon, we can see the bottom side of the disc: Light originating from this side is bent around the black hole by roughly 180° and arrives at the observer through the gap between disc and event horizon.

In figure 7.1.2, the disc is strongly distorted because the observer is located far away from the axis of symmetry. The part of the disc that would be hidden behind the hole in absence of gravity is visible as a magnified and bent image: This happens because the light coming from these parts of the disc is bent downwards (towards the black hole) and curves around the hole thus arriving at the observer from above. The evenly spaced sectors of the disc also appear strongly stretched. The bottom side of the disc is visible because its light also curves around the hole and arrives at the observer from below.

Starry Sky behind Black Hole Next, we investigate how the appearance of the night sky is warped by a black hole. We use the Milky Way panorama from figure 6.10.1 as our projection. Figure 7.1.3 shows the computed image and listing 7.1.2 is the corresponding scene file.

The stars right behind the black hole are bent into rings around the hole. Such ring-like images of astronomical objects are called *Einstein rings*. Numerous Einstein rings have been found and photographed by telescopes. See [SP99.1, chapter 2.5.6].

Sphere around Event Horizon Figure 7.1.4 shows a Schwarzschild black hole encased by a sphere. We chose the radius of the sphere so small, so it almost touches the horizon. The pattern on the sphere can thus be thought of as being printed on the horizon. Listing 7.1.3 is the corresponding scene file.

We can see both poles (points, at which the triangles meet) of the black hole at the same time, because light rays are being bent towards the black hole, even though we look at the hole from the equator.

Redshift Figure 7.1.5 shows the redshift of an accretion disc that touches the horizon. Listing 7.1.4 is the corresponding scene file.

The event horizon is completely red, as light from there is infinitely redshifted (it looses all of its energy): No light escapes the horizon.

Points on the disc start to become more and more purple, as we move out. On the outer edge of the disc, almost no redshift occurs.

Blueshift Figure 7.1.6 shows blue- and redshift an observer looking outwards from just above the horizon sees. The scene is otherwise the same as in figure 7.1.5. Figure 7.1.7 is the corresponding colour image and listing 7.1.5 the scene file used for both.

We see in the colour image that the sky (white) appears in the middle of the image and that it is surrounded by the event horizon. In contrast to earlier images, the sky and the horizon are swapped.

According to the redshift image, light coming from the event horizon is still infinitely redshifted even tough the observer almost touches the horizon. Along the disc, the redshift falls off and ultimately fades into blueshift. This happens because most of the disc is further out than the observer.

Sphere near Schwarzschild Black Hole The figures 7.1.8, 7.1.9 and 7.1.10 show a Schwarzschild black hole with a sphere in different relative positions. First, the sphere is located between observer and black hole, then it is moved by 90° along ϕ and finally it is located directly behind the hole as seen from the observer. Listing 7.1.6 is the corresponding scene file.

In figure 7.1.8, the sphere is located between the hole and the observer. It appears completely round. Close to the horizon, we make out a thin ring-like image af the sphere: This images consists of light from the sphere's backside that was bent by gravity and travelled all the way around the black hole to arrive at the observer.

In figure 7.1.9, the sphere, the hole and the observer form a right angle. The sphere is located to the right of the horizon and appears almost round. On the opposite side of the hole, we see a small sickle-like image of the sphere. This secondary image again consists of photons being bent around the hole.

In figure 7.1.10, the sphere is directly behind the hole, as seen from the observer. It appears strongly magnified as an Einstein ring around the black hole.

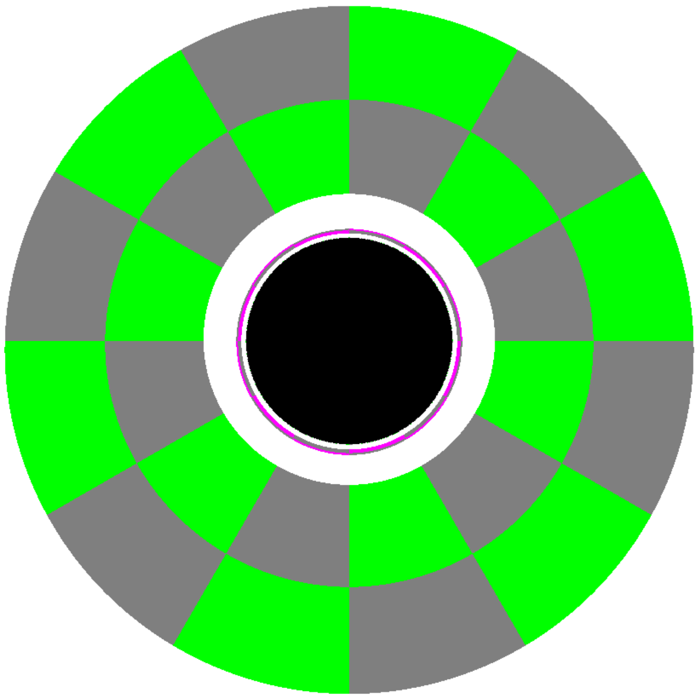


Figure 7.1.1: Schwarzschild black hole with accretion disc. Zenith angle of the observer: $\theta = 5°$.

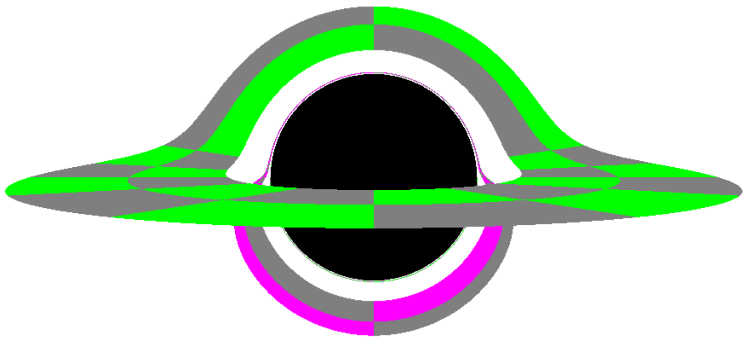


Figure 7.1.2: Schwarzschild black hole with accretion disc. Zenith angle of the observer: $\theta = 85°$.

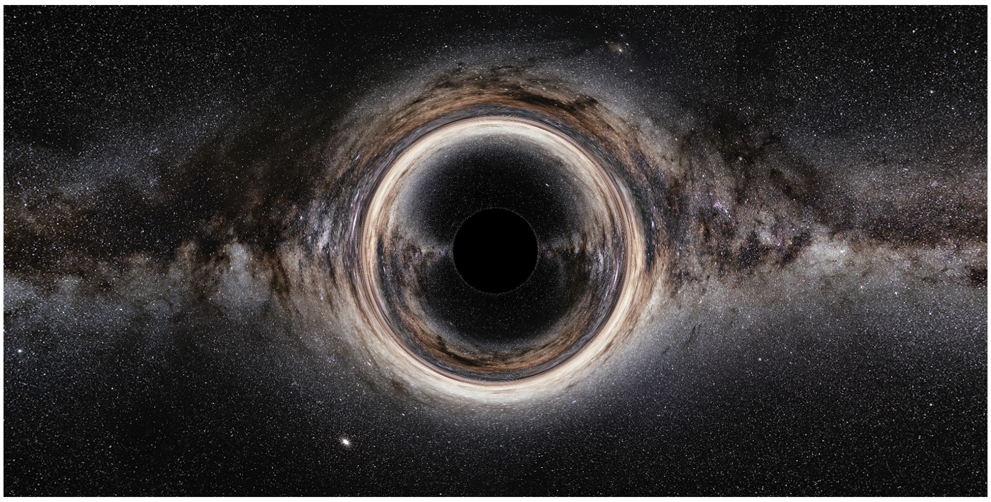


Figure 7.1.3: Image of Schwarzschild black hole with night sky in the background.

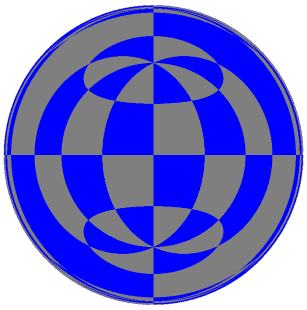


Figure 7.1.4: Schwarzschild black hole with sphere around the horizon.

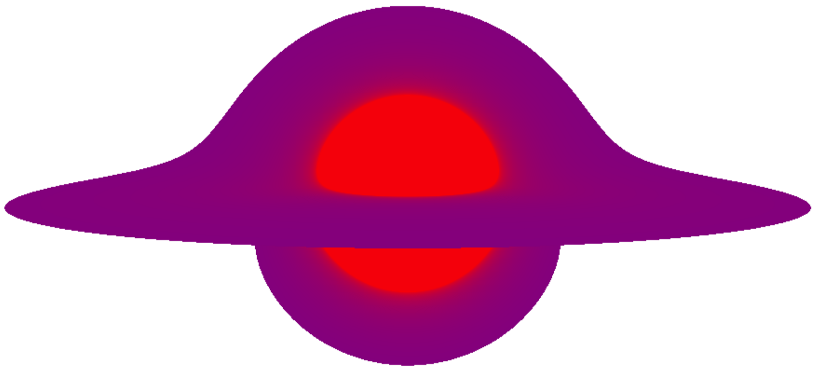


Figure 7.1.5: Redshift due to a Schwarzschild black hole. The colour of the sky was set to white.

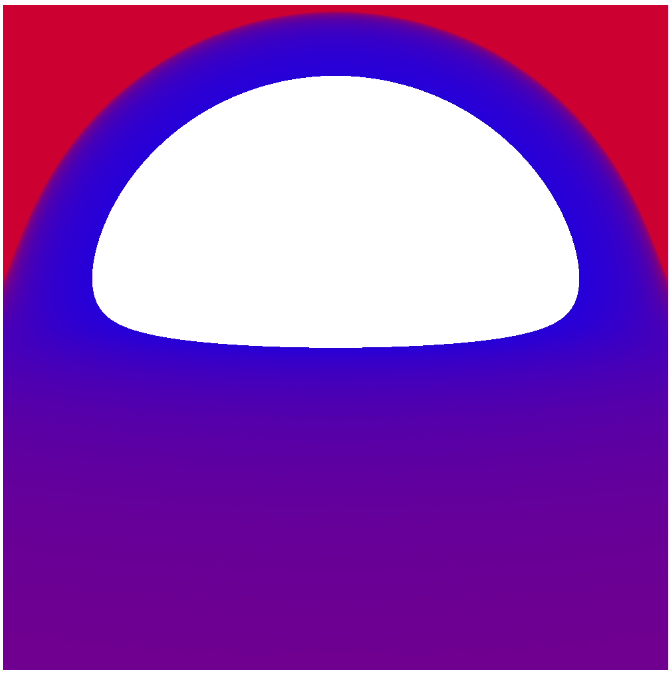


Figure 7.1.6: Blueshift due to a Schwarzschild black hole. The sky colour was set to white.

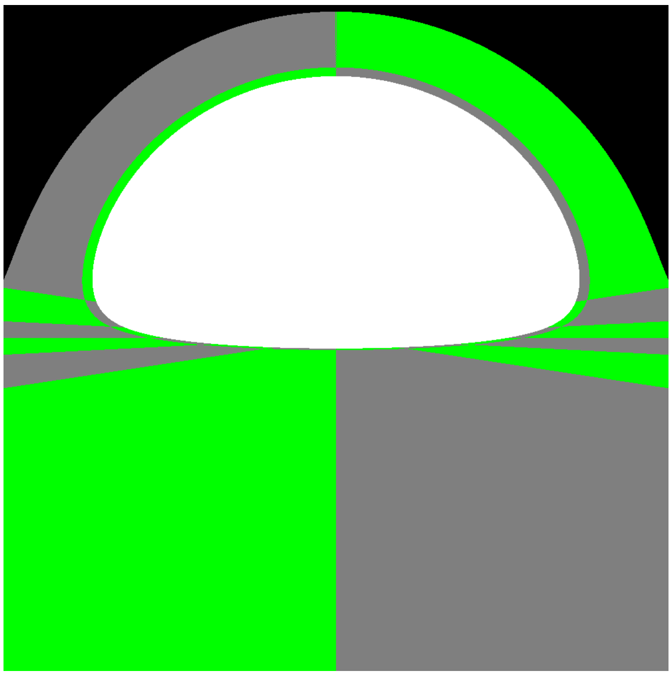


Figure 7.1.7: The observer is looking out from a point just above the horizon of a Schwarzschild black hole.

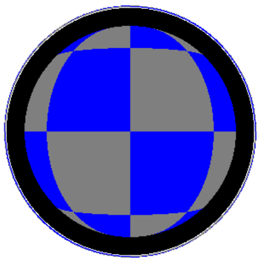


Figure 7.1.8: Sphere between Schwarzschild black hole and observer.

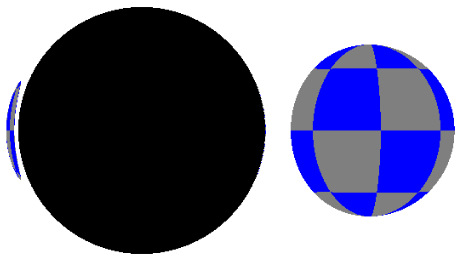


Figure 7.1.9: Sphere next to a Schwarzschild black hole. The sphere is rotated by 90° ahead of the observer along ϕ.

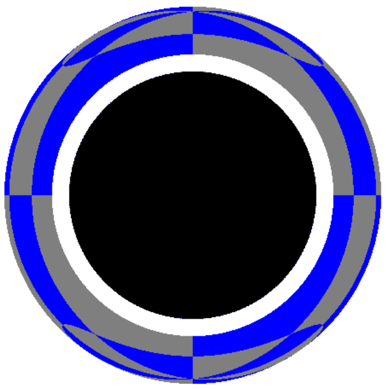


Figure 7.1.10: Sphere behind a Schwarzschild black hole.

Listing 7.1.1: Accretion disc of a Schwarzschild black hole

```
---
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: Starry sky behind Schwarzschild black hole

```
---
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 black hole with sphere around horizon

```
---
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: Redshift around Schwarzschild black hole

```
---
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: Blueshift around Schwarzschild black hole

```
---
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: Sphere close to a Schwarzschild black hole

```
---
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 Black Holes

Here, we compare images of uncharged, rotating black holes with images of Schwarzschild black holes. Electrically uncharged and rotating black holes are called *Kerr black holes* because their metric is the Kerr metric. The Kerr-Newman metric becomes the Kerr metric when $Q = 0$, $M \neq 0$ and $a \neq 0$.

Accretion Disc around Kerr Black Hole Analogously to the figures 7.1.1 and 7.1.2 showing Schwarzschild black holes, the figures 7.2.1 and 7.2.2 show Kerr black holes in the same situation. The only difference is that now $a \neq 0$. The observer is again located at the two positions $\theta = 5°$ and $\theta = 85°$ in the two figures. Listing 7.2.1 is the corresponding scene file.

Figure 7.2.1 differs from figure 7.1.1, in that the disc is distorted in a whirling manner near the event horizon. This is the result of the frame-dragging effect. The observer is roughly looking down the negative z axis ($\theta = 0°$ is the black hole's rotation axis). We thus would think that the space-time is whirled according to the right-hand rule in a counter-clockwise direction. We see quite the opposite: photons are dragged into a clockwise motion (Photons dragged clockwise result in the image being dragged clockwise too, since the photons are emitted by the imaged objects). This discrepancy can easily be explained: it stems from the fact that we *invert time* when we trace the rays backwards. Of course, when we invert time, the black hole spins in the opposite direction.

Correct results are obtained, when we interpret a positive rotation parameter a to correspond to a *clockwise* rotation of the black hole around the positive z axis (that is, according to the *left-hand rule*).

In figure 7.2.2 the observer is located at $\theta = 85°$. The frame-dragging effect is visible through the distortion of the sectors on the disk.

Furthermore, the event horizon seems to bulge out on one side. This is because photons only manage to cross the ergosphere on one side of the axis: when they move against the flow of spacetime, they fall into the hole, otherwise they come through. Generally speaking, a photon moving against the flow of spacetime will require more coordinate time to cross a certain distance than one moving with the flow. Since both are subject to the same radial gravitation field, the one travelling longer falls in deeper towards the horizon. This is were the asymmetry of the image comes from.

A further consequence of the frame-dragging effect is the asymmetry of the photon sphere. In fact, the photon sphere is not a sphere but a rather complicated surface. See [CC13.1] for a rigorous discussion of a rotating black hole's photon sphere.

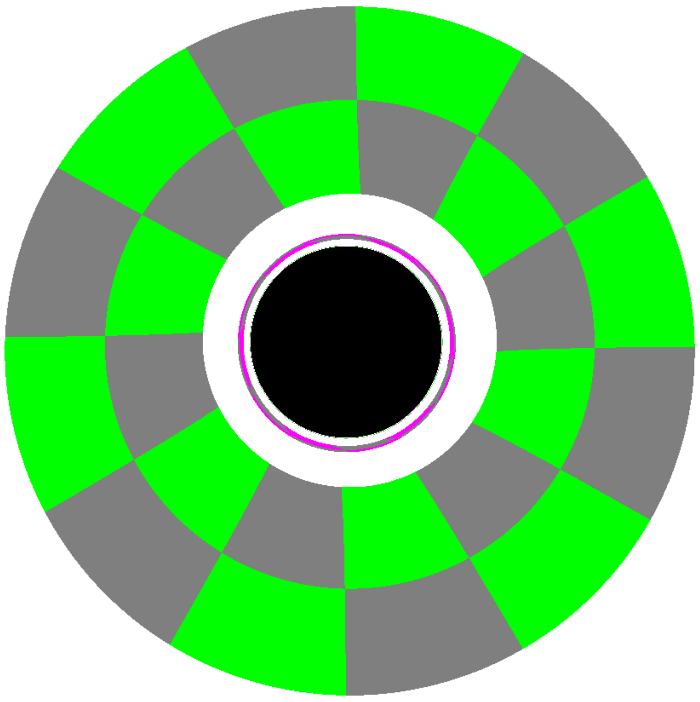


Figure 7.2.1: Kerr black hole with accretion disc. Zenith angle of observer: $\theta = 5°$.

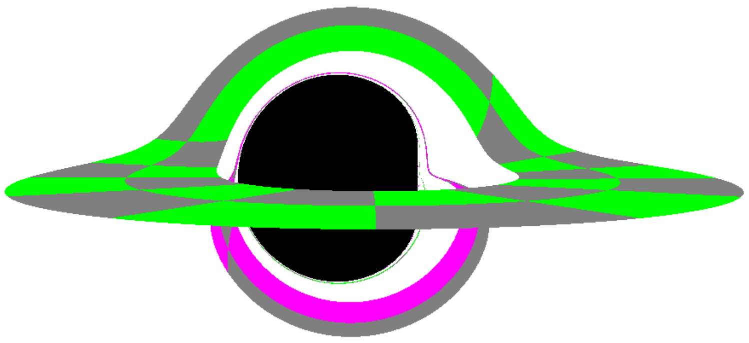


Figure 7.2.2: Kerr black hole with accretion disc. Zenith angle of observer: $\theta = 85°$.

Listing 7.2.1: Accretion disc of a Kerr black hole

```
---
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 Black Holes

Lastly, we would like to compare a *Reissner-Nordström black hole* ($a = 0$, $Q \neq 0$) with a Schwarzschild black hole.

Reissner-Nordström black holes do not differ much externally from Schwarzschild ones. The difference lies mostly within the event horizon: Reissner-Nordström black holes can, just like the general Kerr-Newman black holes, have two event horizons. We are however not concerned about the inner features of black holes, since `cuRRay` stops the integration of geodesics at the outer event horizon.

Sphere behind Reissner-Nordström Black Hole A visible difference is the smaller event horizon of Reissner-Nordström black holes compared to Schwarzschild black holes. This can be seen directly in equation (2.2.5).

Figure 7.3.1 shows the same scene as figure 7.1.10 only with $Q \neq 0$. Listing 7.3.1 is the corresponding scene file. The event horizon appears smaller than in figure 7.1.10.

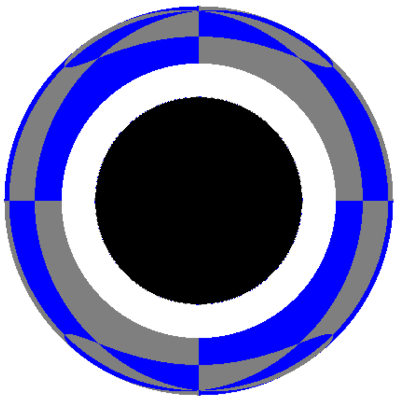


Figure 7.3.1: Sphere behind Reissner-Nordström black hole.

Listing 7.3.1: Sphere behind Reissner-Nordström black hole

```
---
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
```

Chapter 8

Discussion

Here, we will discuss several aspects of `cuRRay` and mention possible improvements. We begin with a discussion of the software's reliability and investigate potential software errors. Subsequently, we discuss performance, then possible extensions of the source code and finally we mention `GRay`, the software, from which `cuRRay` got the step size control.

8.1 Reliability of `cuRRay`

`cuRRay` was tested for many hours during and after development. All detected errors (including some wrong Christoffel symbols) were corrected.

We further tested the implemented algorithms in different ways to detect specific errors. For instance, we tested the raytracing algorithm with different parametrisations of the geodesics (which is of course allowed mathematically) and could detect and correct an erroneous equation of motion.

Certain errors are known but were not corrected, because they correspond to rare special cases and their fix would not be worth the effort. The following known errors can still occur:

Operating System-induced Errors Operating system errors, like running out of memory or hardware errors are very hard to detect and fix. Such errors generally crash the software. These errors can be prevented by complying with the system requirements in chapter 4.2.

Under Windows, it is possible (and even quite likely) that the CUDA version crashes due to the TDR timer. To prevent these crashes, the timer needs to be configured or disabled. See chapter E.5.

Faulty User Input If the scene file contains impossible settings (like $M^2 < a^2 + Q^2$ or observer inside event horizon), the raytracing might fail, as the behaviour is undefined. The user must manually ensure the correctness of the settings.

Numerical Imprecision in RK4 The numerical inaccuracy of the RK4 algorithm is very hard to measure, since the correct geodesics are unknown. However, accuracy increases, as the step size is decreased.

Equation (6.6.1) controls the step size used by the RK4 algorithm. The factor c allows the manual tweaking of the step size. The following manual procedure allows for accurate images: We draw the same frame multiple times with ever decreasing c. As soon as the pixels of two consecutive images no longer differ, the step size is accurate enough and we can use it in computations of similar scenes. We found the factor $c = 1/32$ (which is also the default value) to be sufficient for resolutions up to 2000 by 2000 pixels and most scene configurations. Even extreme situations like observers just outside the horizon were successfully tested with this value.

Coordinate singularities The polar axis ($\theta = 0$ and $\theta = \pi$ respectively) is a coordinate singularity, as θ and ϕ are discontinuous there. RK4 steps can only be calculated in a continuous region of the coordinate system. If a step of a ray crosses the polar axis, we expect the raytracing of that ray to yield wrong results.

Such errors quickly show, as coordinates reach invalid values ($r < 0$, $\theta < 0$ or $\theta > \pi$). `cuRRay` detects these errors and draws erroneous pixels using a configurable colour. Such errors can easily be avoided, as they only occur, when rays hit the axis perfectly. We can easily choose frame sizes or orientations of the observer, such that this does not happen. And of course, the observer should not be positioned on the polar axis. Figure 8.1.1 shows a Schwarzschild black hole where the settings have deliberately been chosen such as to generate faulty pixels.

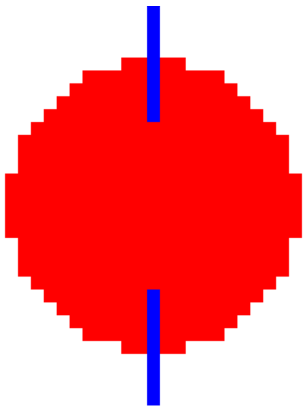


Figure 8.1.1: Schwarzschild black hole with errors (blue pixels) along the axis. The dimensions of the frame were chosen to be odd (31x31 pixels), to make sure that pixels in the center of the image perfectly hit the axis. To better show the effect, we used small frame dimensions and set the event horizon colour to red.

8.2 Efficiency of `cuRRay`

`cuRRay`'s performance was not compared with existing software.

Comparisons between the CUDA version and the CPU version showed the

CUDA version to be generally faster than the CPU version. The CUDA version was at least twice as fast as the CPU version when tested on one computer (Intel Core i7-4770 @ 3.4 GHz, ASUS GTX-760 DirectUI II OC) and even up to four times faster when tested on another machine (Intel Core i7-6700K @ 4.0 GHz, EVGA GTX-1070 Founder's Edition). The superiority of the CUDA version could be clearly demonstrated.

The performance of the software was greatly improved during development (an increase in speed of roughly 200 times since the first prototype). Yet, mostly the CUDA version could still be improved: Investigations with the NVIDIA Visual Profiler [1] showed that the graphics card is waiting on VRAM data roughly halve of the time. By using different types of VRAM, this waiting time could be minimised and the performance further increased.

8.3 Outlook

`cuRRay` can be used to visualise Kerr-Newman spacetime and could be employed as a learning aid for students of relativity. New functions could render the visualisations even more interesting:

For instance, one could store the single steps of a geodesic and plot them in a three-dimensional graph to get an idea of the shape of the geodesic. Or, one could extend raytracing to trajectories of other bodies like massive particles. This way one could compute the orbits of small asteroids. Lastly, one could use RK4 to perform raytracing in other spacetimes like *Kruskal-Szekeres spacetime* inside a Schwarzschild black hole.

8.4 Existing Software

We would like to point out `GRay` [CC13.1], a software with similar functionality as `cuRRay`: `GRay` is also a CUDA-accelerated raytracer for relativistic spacetimes around black holes which uses the RK4 algorithm. `cuRRay` employs the step size control of `GRay`.

1. See `https://developer.nvidia.com/nvidia-visual-profiler`.

Appendix A

First Order Geodesic Equations

Here, we simplify the geodesic equations for t and ϕ using the Killing vector fields of Kerr-Newman spacetime from second order equations to first order equations.

The Killing vector fields of Kerr-Newman spacetime give the following constants of motion:

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

and

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

See chapter 2.3.1. The equations (A.1) and (A.2) contain the velocity components $u^t = dt/d\lambda$ and $u^\phi = d\phi/d\lambda$. We can solve for those components and obtain first order equations for t and ϕ.

First we solve equation (A.2) for $d\phi/d\lambda$:

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

and insert into equation (A.1):

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

We now solve for $dt/d\lambda$:

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

Finally, we multiply by $g_{\phi\phi}$ and get the first order differential equation describing the evolution of the t coordinate.

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

Similarly, we obtain the equation for ϕ:

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

Appendix B

Christoffel Symbols of Kerr-Newman Spacetime

In this appendix, we will list the required Christoffel symbols for the equations (2.3.10) and (2.3.11). First, we show the procedure for calculating the symbols, then we demonstrate the derivation for one symbol and finally, we list all relevant symbols.

Relevant Christoffel Symbols The Christoffel symbols need to be computed from the metric components according to equation (1.7.9):

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

Before we invert or differentiate the components of the metric, we note that only the Christoffel symbols with $\alpha \in \{r, \theta\}$ need to be computed. This will simplify our work quite a bit.

Inverse Metric We require the components of the inverse metric tensor in equation B.1. They are given by equation (1.6.5). If we think of the components of the metric tensor as the components of a 4x4-matrix, then the components of the inverse metric tensor are the components of the inverse of said matrix.

According to equation (2.2.1), the metric of Kerr-Newman spacetime has the following structure:

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

The inverse metric is therefore:

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

Now we only need the inverse metric components with $\alpha \in \{r, \theta\}$. These are

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

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

Since only terms $g^{\alpha\beta}$ with $\alpha = \beta$ are required, we only need to sum over a single term in equation (B.1) for each Christoffel symbol.

Derivatives of the Metric Components The biggest challenge is to differentiate the metric components. But even here, some of the work can be spared by realising that the metric is independent of t and ϕ, that is $\partial_t g_{\alpha\beta} = \partial_\phi g_{\alpha\beta} = 0$. We can simplify the expressions for single Christoffel symbols by crossing out vanishing terms:

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

Next, we need to evaluate the expressions for the Christoffel symbols. We show as an example the evaluation of ${\Gamma^r}_{tt}$. The other Christoffel symbols are then calculated in similar fashion.

Example For ${\Gamma^r}_{tt}$ we get from (B.4),

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

Using the quotient rule for differentiation and simplifying:

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

List of Christoffel Symbols The Christoffel symbols required by `cuRRay` are the following:

$$
\begin{aligned}
{\Gamma^r}_{tt} =& \frac{\Delta}{\Sigma^3}\left[Mr^2 - Q^2r - Ma^2\cos^2\theta\right] && (B.9) \\
{\Gamma^r}_{rr} =& \frac{1}{\Delta\Sigma}\left[-Mr^2 + Q^2r + a^2r + (M-r)a^2\cos^2\theta\right] && (B.10) \\
{\Gamma^r}_{\theta\theta} =& -\frac{\Delta}{\Sigma}r && (B.11) \\
{\Gamma^r}_{\phi\phi} =& \frac{\Delta}{\Sigma^3}\sin^2\theta\left[\Sigma\left((r-M)a^2\sin^2\theta - 2r(r^2+a^2)\right)\right. \\
& \left. + r\left((r^2+a^2)^2 - \Delta a^2\sin^2\theta\right)\right] && (B.12) \\
{\Gamma^r}_{t\phi} =& \frac{\Delta}{\Sigma^3}a\cdot\sin^2\theta\left[-Mr^2 + Q^2r + Ma^2\cos^2\theta\right] && (B.13) \\
{\Gamma^r}_{r\theta} =& -\frac{1}{\Sigma}a^2\ \cos\theta\sin\theta && (B.14) \\
{\Gamma^\theta}_{tt} =& \frac{1}{\Sigma^3}a^2\sin\theta\cos\theta\left[Q^2 - 2Mr\right] && (B.15) \\
{\Gamma^\theta}_{rr} =& \frac{1}{\Sigma\Delta}a^2\sin\theta\cos\theta && (B.16) \\
{\Gamma^\theta}_{\theta\theta} =& -\frac{1}{\Sigma}a^2\sin\theta\cos\theta && (B.17) \\
{\Gamma^\theta}_{\phi\phi} =& \frac{1}{\Sigma^3}\sin\theta\cos\theta\left[(r^2+a^2)(\Delta a^2\sin^2\theta - (r^2+a^2)^2) + \Sigma\Delta a^2\sin^2\theta\right] \\
& && (B.18) \\
{\Gamma^\theta}_{t\phi} =& \frac{1}{\Sigma^3}a\cdot\sin\theta\cos\theta(2Mr - Q^2)(r^2+a^2) && (B.19) \\
{\Gamma^\theta}_{r\theta} =& \frac{1}{\Sigma}r. && (B.20)
\end{aligned}
$$

These expressions were all calculated by hand and later checked using the computer algebra system `wxMaxima` [1].

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

Appendix C

Gravitational Redshift in Kerr-Newman Spacetime

In this appendix, we derive equation (3.5.1) for the gravitational redshift in Kerr-Newman spacetime. To this end, we calculate the ratio of frequencies between sender and receiver. The sender is the light source, the receiver the observer.

Angular Frequency The angular frequency of a photon as seen by an observer with velocity v^α is according to equation (3.2.1)

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

The wave vector k^α is proportional to the velocity u^α of the photon, as we saw in chapter 3.2: $k^\alpha = a \cdot u^\alpha$, where a is the factor of proportionality. We get

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

The units of ω depend on the parametrisation of u^α and v^β.

Observer To prevent redshift due to the relativistic Doppler shift, the observer and the source must not move relative to each other. This is the case if both are stationary. For both, we thus set their velocities to $v^0 \neq 0$ and $v^a = 0$. This however restricts redshift calculation to sources and observers outside the ergosphere.

The frequency becomes

$$\omega = -a \cdot \left(g_{tt} u^t v^t + g_{t\phi} u^\phi v^t \right). \tag{C.3}$$

We have

$$\boldsymbol{v} \cdot \boldsymbol{v} = |\boldsymbol{v}|^2 = g_{\alpha\beta} v^\alpha v^\beta = g_{tt} v^t v^t, \tag{C.4}$$

that is

$$v^t = \sqrt{\frac{|\boldsymbol{v}|^2}{g_{tt}}} = \sqrt{-|\boldsymbol{v}|^2} \frac{1}{\sqrt{-g_{tt}}}. \tag{C.5}$$

In the last step we used the fact that $|\boldsymbol{v}|^2 < 0$ and $g_{tt} < 0$ at all times (outside the ergosphere).

Redshift The ratio of frequencies is

$$\frac{\omega_r}{\omega_s} = \frac{-a \cdot \sqrt{-|\boldsymbol{v}|^2}\left(u^t \frac{g_{tt}}{\sqrt{-g_{tt}}} + u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\right)\Bigg|_r}{-a \cdot \sqrt{-|\boldsymbol{v}|^2}\left(u^t \frac{g_{tt}}{\sqrt{-g_{tt}}} + u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\right)\Bigg|_s}. \tag{C.6}$$

This ratio only makes sense, if ω_r and ω_s are given in the same units, thus if u^α and v^α are equally parametrised for both the sender and the receiver. Given a parametrisation, $|\boldsymbol{v}|^2$ takes on a constant value for every object moving with less than the speed of light (a result from special relativity that we will not further discuss). $|\boldsymbol{v}|^2$ is thus the same for both observers. Also a is the same for similar reasons. We arrive at the final equation:

$$\frac{\omega_r}{\omega_s} = \frac{u^t\sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\Bigg|_r}{u^t\sqrt{-g_{tt}} - u^\phi \frac{g_{t\phi}}{\sqrt{-g_{tt}}}\Bigg|_s}. \tag{C.7}$$

The nominator of the right hand side is evaluated at the observer, the denominator at the light source. We get the values of u^α along the light ray using raytrcing, g_{tt} and $g_{t\phi}$ can be calculated using equation 2.2.1.

Appendix D

Derivation of the Cartesian-BL Transformation

Here we derive the transformation (6.4.1) that transforms vectors from local Cartesian coordinates to global BL coordinates:

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

Transformation of the Metric We know that our transformation acts on tensors according to equation (1.5.1). It thus has to transform the metric of the local inertial frame to the Kerr-Newman metric, evaluated at the observer according to

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

where $\hat{g}_{\gamma\delta}$ are the components of the Kerr-Newman metric at the observer and $\eta_{\alpha\beta}$ are the components of the Minkowski metric (flat spacetime in Cartesian coordinates). Note, how the transformation is applied in the opposite direction: This is due to the fact that the metric tensor transforms covariantely, while vectors transform contravariantly.

Transformation Matrix To better understand our transformation, we think of the components of both metric tensors as matrices. Our transformation then becomes a transformation matrix. Because matrices are Rank-$(1,1)$ tensor, but metric tensors have rank $(2,0)$, we have to be careful, when transforming the equations (D.1) and (D.2) in matrix notation.

First, we define the matrices for the metric tensors. The row numbers of such a matrix corresponds to the first index, the column numbers to the second:

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

The components of the matrix are simply the components of the metric. The same is true for the matrix $\hat{g}$, whose components are $\hat{g}_{\alpha\beta}$.

Equation (D.1) now becomes

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

and equation (D.2) leads to

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

Ψ is the transformation matrix:

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

The first index designates the row, the second the column.

Metric Matrices Let us have a look at the form of the metric matrices:

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

We see that $\hat{g}$ can be diagonalized and scaled into η.

Diagonalising We know from linear algebra, that we can *diagonalise* a square, invertible matrix A into a diagonal matrix (one, where all entries except on the diagonal are zero):

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

The column vectors of P are the *eigenvectors* of A. The components of the diagonal matrix D along the diagonal are the *eigenvalues* of A. For a symmetric matrix A, we can always find an *orthogonal* matrix P, because the eigenvectors of a symmetric matrix are orthogonal (see [BI78.1, S. 150]). If P is orthogonal, we have

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

Since $\hat{g}$ is invertible and symmetric, we have

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

P is the matrix of orthogonal eigenvectors of $\hat{g}$ and D the diagonal matrix containing the eigenvalues of $\hat{g}$ along the diagonal. We are free to choose the eigenvectors such that they are not just orthogonal, but also normalised to length one. Without restriction, we can find an *orthonormal* (orthogonal and normal) matrix P satisfying the previous equation.

We can find the eigenvalues of $\hat{g}$ by solving the *characteristic polynomial*:

$$\begin{aligned} \lambda_0 &= \frac{1}{2}\left(\hat{g}_{tt} + \hat{g}_{\phi\phi} + \sqrt{(\hat{g}_{tt} - \hat{g}_{\phi\phi})^2 + 4\hat{g}_{t\phi}^2}\right) \\ \lambda_1 &= \hat{g}_{rr} \\ \lambda_2 &= \hat{g}_{\theta\theta} \\ \lambda_3 &= \frac{1}{2}\left(\hat{g}_{tt} + \hat{g}_{\phi\phi} - \sqrt{(\hat{g}_{tt} - \hat{g}_{\phi\phi})^2 + 4\hat{g}_{t\phi}^2}\right). \end{aligned} \tag{D.11}$$

The eigenvectors are then

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

For convenience, we normalise the columns of P afterwards.

Scaling To get the flat metric, the diagonalised metric matrix needs to be scaled:

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

where S is the following scaling matrix:

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

The eigenvalues are all positive real numbers, outside the ergosphere. Since we only consider observers outside the ergosphere, this scaling matrix is well defined.

We can easily split S into two diagonal matrices, since S is diagonal itself. Also, the multiplication with diagonal matrices is commutative. We can thus construct a coordinate transformation Ω:

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

where we used the notation

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

Further Transformations Ω is not yet the transformation Ψ we are looking for. We can extend Ω by further transformations that leave the metric unchanged, but change vectors: We can add a rotation and a Lorentz transformation.

Rotation We could further rotate the observer. Rotations leave the lengths of vectors unchanged and thus do not change the metric. We need a rotation to align the axes according to figure 6.4.2.

Investigating the action of Ω on Cartesian basis vectors, we see that no additional rotation is required: The eigenvectors of $\hat{g}$ we chose earlier turns the action of P on spatial basis vectors into a simple scaling. This is exactly what we want: x points along r, y along θ and z along ϕ.

Lorentz Transformation A Lorentz transformation (see chapter 1.1) allows us to change the velocity of the observer relative to BL coordinates. We require the observer to be stationary relative to these coordinates. The velocity vector of the observer in BL coordinates is thus

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

Here we chose the time component such, that $\hat{g}_{\alpha\beta} V^{\alpha}_{\mathrm{BL}} V^{\beta}_{\mathrm{BL}} = -1$. This corresponds to a parametrisation using the observers proper time.

The squared length of a vector takes on the same value in every coordinate system. We therefore have in Cartesian coordinates:

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

and $\eta_{\alpha\beta} V^{\alpha}_{\mathrm{C}} V^{\beta}_{\mathrm{C}} = -1$.

Our coordinate transformation needs to fulfil

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

Ω however does not fulfil this condition: By applying Ω to a purely timelike vector, we get a vector with ϕ component other than zero. This happens because of the $dtd\phi$ cross term of the Kerr-Newman metric. Ω transforms the local coordinates of an observer moving along ϕ with a certain velocity to BL coordinates, not one who is stationary.

This motion can be corrected by a Lorentz transformation:

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

Λ is the Lorentz transformation. We apply the Lorentz transformation in Cartesian coordinates, before the rest of the transformation. We only need to correct the velocity in ϕ direction. This corresponds to a correction in the z direction of the local coordinates.

Inverse Lorentz Transformation It will be easier to work with the inverse Lorentz transformation Λ^{-1}. It has the following form

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

According to the equations (D.19) and (D.20), it fulfils the following condition:

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

where we have introduced the vector $\vec{X} = \Omega^{-1} \cdot \vec{V}_{\mathrm{BL}}$. Considering equation (1.1.3) and (D.21), we can now solve for v and calculate γ:

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

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

Here we used $\vec{X}_t^2 - \vec{X}_z^2 = 1$, since $\eta_{\alpha\beta} X^{\alpha} X^{\beta} = -1$. Inserting into equation (D.21) allows us to calculate Λ^{-1}.

Complete Transformation The complete transformation is given by

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

It fulfils the equations (D.4) and (D.2) and makes sure that the observer is stationary and oriented correctly.

Appendix E

cuRRay: User Manual

E.1 Command Line

`cuRRay` is executed from the command line. The following options can be supplied:

```
cuRRay  [-s <scene file>] [-o <output directory>]
        [-x <frame width>] [-y <frame height>] [-f <frame count>]
        [-t <output type>] [-c <system configuration file>]
        [-v <true/false>] [-q <true/false>] [-a <true/false>]
        [-i] [-h] [--version] [--]
```

The command line for the CPU version, `cuRRay_cpu`, is very similar:

```
cuRRay_cpu  [-s <scene file>] [-o <output directory>]
        [-x <frame width>] [-y <frame height>] [-f <frame count>]
        [-t <output type>] [-c <system configuration file>]
        [-v <true/false>] [-q <true/false>] [-a <true/false>]
        [-i] [-h] [--version] [--]
```

`-s, --scene_file <scene file>` Sets the scene file. Example: `-s scene.yml`.

`-o, --output_dir <output directory>` Sets the output directory. Example: `-o output`.

`-x, --x_res <frame width>` Sets the frame width. Example: `-x 1000`.

`-y, --y_res <frame height>` Sets the frame height. Example: `-y 1000`.

`-f, --frames <frame count>` Sets the frame count. Default: 1. Example: `-f 20`.

`-t, --output_type <output type>` Sets the output type. Can contain any of the following characters: `i`, `c`, `r` and `d`. Default: `icr`.

`i`: creates information text file (`info.txt`) containing a summary of the scene and rendered frames.

`c`: creates PNG files with the calculated pixel colours (enumerated with frame number: `c_0.png`, etc.).

`r`: creates PNG files with redshift data (enumerated with frame number).

`d`: creates CSV file with detailed frame information (enumerated with frame number).

`-c, --config <system configuration file>` Sets the system configuration file. Default: `config.yml`.

`-v, --verbose <true/false>` Enables verbose logging. Overrides setting in the system configuration file.

`-q, --quiet <true/false>` Mutes console output. Overrides setting in the system configuration file.

`-a, --auto_accept <true/false>` By default, `cuRRay` asks the user whether or not to proceed with the calculations after the scene file was loaded. Is this option set, the user is not asked. If the console output is quiet, the user is not asked no matter the value of this option. Overrides setting in system configuration file.

`-i, --info` Shows information about installed NVIDIA GPUs. See listing E.1 for a possible output.

Listing E.1: Possible output of `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)
Max. grid dimensions:
   (2147483647,65535,65535)

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

No scene file specified, no raytracing will be
   performed.
```

`-h, --help` Shows a list of all options and a short description.

`--version` Shows the version of the software (2.0 as of typesetting).

`--` All further text after this option is ignored.

E.2 System Configuration File (Sysconfig)

The *system configuration file* (*sysconfig*) contains all settings concerning GPU, CPU, output and raytracing algorithm. The default sysconfig file is `config.yml`. To load another file, see appendix E.1. The sysconfig file is a YAML file (see `http://yaml.org`). Note especially that YAML does not use tabs, but multiple spaces for indentation.

GPUs One can enable the GPUs being used by the CUDA version using settings in the sysconfig file. For instance:

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

If a system only contains one GPU, the second entry (starting with 1:) can be omitted. Likewise, one would add new entries if more GPUs are installed. The numbers of the GPUs used in the sysconfig file are the same as outputted by `cuRRay -i`.

CPU The amount of CPU threads used by the raytracer in the CPU version can be set accordingly:

```
cpu:
    threads: 8
```

Optimal performance is achieved, when this number is set to the number of cores (or virtual cores) of the CPU.

Raytracer The constants ϵ (see chapter 6.8) and c (see chapter 6.6) can be set accordingly:

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

Log The logging output is set accordingly:

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

`verbose` corresponds to `-v`.

`quiet` corresponds to `-q`.

`keep_log` cannot be set in the command line. It controls, whether a separate log file is created every time the software runs (`keep_log: true`), or whether the same log file is overwritten every time. If the software crashes, a separate log file with the crash report is created in any case. All log files including error reports are stored in `/log`.

Automatic Execution The option `auto_accept` corresponds to `-a`. For instance:

```
auto_accept: false
```

Example Listing E.1 shows the standard sysconfig file `config.yml`.

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

```yaml
# Default config file

---

# This configures the GPUs used by cuRRay.
# 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

To run the ratracer, at least the options `-s`, `-o`, `-x` and `-y` need to be set. Additionally, one might set `-f` and `-t`.

Example To use the scene file `scene.yml` and the output directory `output`, to create 20 frames with a resolution of 2000 by 2000 pixels each and to create colour and redshift files, one uses the following command line:

```
cuRRay -s scene.yml -o output -x 2000 -y 1000 -f 20 -t cr
```

E.4 Scene File

The scene file is a YAML file describing the scene used to create frames.

Angles Angles can be specified in three different units: degrees, radians and radians as multiples of π. The following four values are equivalent (90°):

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

The suffixes `deg` (degrees), `rad` (radians) and `pi` (radians as multiples of π) set the unit in which the numerical value of the angle is interpreted. If no suffix is specified, degree is chosen.

Colours RGB colours are specified as a list of the three colour components (each ranging from 0 to 255). Yellow ($R = 255$, $G = 255$, $B = 0$) for instance, would be written as

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

Animated Values Certain values can be animated over multiple frames. Animated values follow this format:

```
[linear, <start value>, <end value>]
```

`<start value>` and `<end value>` are the values taken on in the first and last frame respectively. For intermediate frames, the value is computed by linearly interpolating between start and end value. To interpolate an angle from 0 to 2π, one would use the following code:

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

Metric Every scene file has to contain the three parameters M, a, Q of the metric according the following example:

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

All three values can be animated.

Observer The observer is configured like in the following example:

```
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` and `phi` are the BL coordinates of the observer. They are mandatory and can be animated.

`roll`, `pitch` and `yaw` are the roll, pitch and yaw angles of the observer. They are optional and zero by default (then, the observer looks directly at the black hole). They can be animated.

`hfov` and `vfov` are the field of view angles of the observer. `hfov` is mandatory, `vfov` is optional. If the vertical angle is omitted, `cuRRay` calculates it from the horizontal angle and the frame dimensions, such that the aspect ratio of the pixels is one:

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

where h is the height and b the width of the frame.

Accretion Disc One can optionally configure an accretion disc which will be centred around $r = 0$ and lie in the plane $\theta = \pi/2$. For instance:

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

    resolution: [2, 12]

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

`color1` and `color2` are the accent colours of the top and bottom side of the disc respectively. Only `color1` needs to be set. If the second colour is not set, it is calculated from the colour components of the first colour according to:

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

`resolution` is the resolution of the chessboard pattern. The first value corresponds to the radial divisions and the second value corresponds to the number of sectors. These values are required.

`yaw` is the angle by which the pattern is rotated around the polar axis. This value is zero by default and can be animated.

`radius` are the two radii of the disc. The first value is the r coordinate of the inner edge, the second is the r coordinate of the outer edge. These values are required and can be animated.

Starry Sky Optionally, an image can be projected onto the sky sphere. For instance:

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

`image` is the image file containing the rectangular projection of the sky. This is required.

`boundary` is the the r coordinate R, outside of which spacetime is approximated to be flat (see chapter 6.10). This value is required and can be animated.

Spheres One can place up to eight spheres around the black hole. For every sphere, a separate section is added in the scene file. For instance:

```
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` is the required accent colour of the sphere.

`resolution` is the required resolution of the chessboard pattern. The first value corresponds to the amount of lines of latitude and the second to the amount of lines of longitude.

`r`, `theta` and `phi` are the BL coordinates of the sphere. They are required and can be animated.

`roll`, `pitch` and `yaw` allow the precise orientation of the sphere. All angles are optional and can be animated, their default value is zero. The sphere is rotated along the axes of a local Cartesian coordinate system (like the one used for the observer, same orientation of the axes relative to the BL axes) first by `roll` around the negative x axis, then by `pitch` around the negative z axis and finally by `yaw` around the negative y axis.

`radius` is the radius of the sphere in local Cartesian coordinates.

Colours One can also configure special colours in the scene file:

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

`sky_color` is the colour of the sky. The default is black.

`horizon_color` is the colour of the event horizon. The default is red.

`error_color` is the colour of erroneous pixels. The default is blue.

Examples For examples of scene files, see the scene files of chapter 7.

E.5 TDR-Timer under Windows

Depending on the configuration, raytracing might take several minutes. Using the CUDA version, the GPU is fully occupied during this time. If the display of the computer is connected to that very same graphics card, the screen freezes until `cuRRay` is done calculating.

To prevent this, Windows contains a built-in security, the *TDR timer* (timeout detection and recovery timer), which kills programs that use the graphics card for too long. This is of course not desired. Therefore, the TDR timer should be disabled if one chooses to run the CUDA version.

The TDR timer can be disabled via the NVIDIA Nsight monitor (if installed) [1]. Alternatively, one can disable it in the registry [2].

1. see `http://docs.nvidia.com/gameworks/content/developertools/desktop/nsight/timeout_detection_recovery.htm`
2. see `https://docs.microsoft.com/en-us/windows-hardware/drivers/display/tdr-registry-keys`

Appendix F

Git Repository

The Git repository at `https://gitlab.com/sebiG/cuRRay` hosts the `cuRRay` source code, builds for Windows 10 and Linux, sample images created with `cuRRay` and this document in German and English.

License `cuRRay` and everything pertaining to it is distributed under the MIT license [1]. A copy of the MIT license can be found on the repository in the end user license agreement file (`EULA.txt`).

Git Versioning The highest version indicated by the tags on the repository reflects the current `cuRRay` source code version. The master branch holds the latest stable release. Changes in additional files like builds or PDFs are not reflected by tags. The newest version of these files will be silently pushed to the master branch. To get the newest version, simply download these files from the master branch.

Further Information For further information (including animations created by `cuRRay`), visit the `cuRRay` sub page on the authors web page: `sebastiengarmier.ch/cuRRay?lang=en`.

1. See `https://opensource.org/licenses/MIT`

Appendix G

Compiling `cuRRay`

This appendix gives a quick overview of the compiling process under Windows and Linux. Similar information can also be found in the readme file (`README.txt`) on the repository.

G.1 General

`cuRRay` should only be compiled in 64-bit mode. It may be possible to compile for 32-bit, but this was never tested.

For both Windows and Linux, libraries are required (see chapter 4.3). The following libraries need to be installed and built for the compiler configuration to be used with `cuRRay`:

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`

G.2 Compiling under Windows (Visual Studio)

Visual Studio A version of Visual Studio [1] supporting the CUDA toolkit 8.0 and the platform toolset v120 needs to be installed. The CUDA toolkit [2] needs to be installed on top.

Environment Variables The following environment variables need to be set:

`INCLUDE` : Contains include directories of all required libraries.

`LIB` : Contains library directories (debug and release) of the required libraries. See the project options in the visual studio solution included in the source

1. See `https://www.visualstudio.com/`
2. See `https://developer.nvidia.com/cuda-toolkit`

code for further information on how the library files should be organised in folders.

Compiling The solution included in the source code on the repository (see appendix F) can simply be compiled in either debug or release mode. The configuration needs to set to 64 bit.

The solution contains projects for the CUDA version (`cuRRay_dev`) and the CPU version (`cuRRay_dev_cpu`).

Executing To execute `cuRRay`, the DLL files of all required libraries need to be copied into the directory of the executable.

The CUDA DLL (`cudart64_80.dll`) is automatically copied by Visual Studio (only required for CUDA version).

If the executable is run from outside Visual Studio, the C and C++ runtimes of the platform toolset v120 need to be copied as well (`msvcr120.dll` and `msvcp120.dll`). These files are included in the Visual Studio installation.

Also, `libpng16.dll` needs to be copied. This file is obtained by compiling libpng.

G.3 Compiling under Linux (g++-5)

GNU-make and g++ For compilation under linux, the repository contains GNU makefiles for g++ (only release, no debug). The makefiles were successfully tested using g++ 5 under Debian 8. By adapting the makefiles, one could possibly use other compilers. Before compiling, the CUDA toolkit needs to be installed.

CUDA Version The makefile `release.makefile` compiles the CUDA version.

CPU Version The makefile `release_cpu.makefile` compiles the CPU version.

Execution If the required libraries were all installed using the package manager, then the library files should be found automatically when executing `cuRRay`. Otherwise, the environment variable `LD_LIBRARY_PATH` needs to be set prior to execution.

List of Figures

List of Listings

Bibliography

[AB16.1] Abbott, B. P. et al., 2016: *Observation of Gravitational Waves from a Binary Black Hole Merger*, DOI: 10.1103/PhysRevLett.116.061102 (visited: 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 (visited: 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*, volume 322, 10, P. 891-921.

[EA16.1] Einstein, Albert, 1916: *Die Grundlage der allgemeinen Relativitätstheorie*, in: *Annalen der Physik*, volume 354, 7, P. 769 - 822.

[BS09.1] Brunier S.; European Southern Observatory (ESO), 2009: *The Milky Way panorama*, `https://www.eso.org/public/images/eso0932a/` (visited: 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*, volume 11, P. 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*, volume 6, P. 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, volume 26, P. 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 (visited: 17.09.16).

[RH16.1] Reissner, Hans, 1916: *Über die Eigengravitation des elektrischen Feldes nach der Einsteinschen Theorie*, in: *Annalen der Physik*, volume 355, 9, P. 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, P. 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, P. 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 (visisted: 06.01.17).

[WR84.1] Wald, Robert Manuel, 1984: *General Relativity*, Chicago: The University of Chicago Press.