Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model

Werner Benger
Dissertation

eingereicht am
Fachbereich Mathematik und Informatik
der Freien Universtität Berlin

im August 2004

Bibliographische Information der Deutschen Bibliothek:
Die deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Benger, Werner
Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model

Berlin, 2005, Lehmanns Media-LOB.de

ISBN 3-86541-108-8

Druck und Verarbeitung: Docupoint Magdeburg, www.docupoint-md.de

Betreuer:	Prof. Dr. Dr. h.c. Peter Deuflhard (FU Berlin) Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) Takustraße 7 D-14195 Berlin-Dahlem
Gutachter:	Prof. Dr. Dr. h.c. Peter Deuflhard (ZIB/FU Berlin) Prof. Dr. H. Edward Seidel (Center for Computation & Technology at Louisiana State University) Prof. Dr. Thomas Ertl (University of Stuttgart)
Datum der Disputation:	8. Juni 2005

Contents

Chapter 1

Introduction and Motivation

1.1 Motivation and Physical Background

Astrophysically realistic scenarios such as colliding black holes are beyond the scope of analytical solutions in general relativity. Numerical Relativity aims to solve Einstein's equation of the gravitational field by the means of computer simulations. In contrast to perturbation theory, this approach can also handle strong gravitational fields. Such simulations are essential for interpreting the data from gravitational wave detectors. Beside the support of observational data, numerical relativity is also important for theoretical studies of the structure of space and time itself. The immense amount of numerical data produced by these simulations can only be appreciated using appropriate visualization tools. The Einstein equation – shown explicitly in eqn. (2.43) – relates the energy and momentum of some matter distribution to the geometry of spacetime. It is a *tensor equation*, as neither the curvature of spacetime nor the properties of matter can be fully described by scalar or vector fields. This exposes the problem of how to explore these central quantities of general relativity: there are no off-the-shelf methods to visualize them. While many techniques exist to cast the intuitive imagination of for scalar and vector fields into computer graphics, for tensor fields such an intuitive vision does not even yet exist and rendering methods are rare.

The goal of this thesis is to develop methods for analyzing curved spacetimes, in particular those expressed in purely numerical form. This work has been done at the Zuse-Institute Berlin in cooperation with the Max-Planck Institute for Gravitational Physics in Potsdam; a review of our collaboration group's work can be found e.g. in the article by Allen et al. [AGL+99]. The primary tool for computing numerical spacetimes was CACTUS [MPIfGP03]. Existing methods have been investigated for their applicability to relativistic data, and new visualization algorithms have been conceived for the exploration of curved spacetime. Special attention has been given to intuitive perceptibility and high performance which enables interactivity. These efforts have required a mathematically grounded framework that supports the language of general relativity. For this purpose a new model for organizing numerical data has been developed. It has been kept general in form, giving it relevance and applicability across a wide range of scientific data beyond visualization of relativistic tensor fields. Moreover it will be demonstrated that some of the novel display algorithms can also be applied well to data originating from other scientific domains.

1.2 State of the Art

As geometry plays a key role in Einstein's concept of gravity, visualization of curved spacetime has a long history in general relativity. Spacetime diagrams are frequently used in special relativity to depict space and time while omitting one or two spatial dimensions. Light propagation appears as a cone originating at each point. Causality relationships in a general relativistic curved spacetime can be displayed by placing light cones at selected points. Penrose extended the idea of spacetime diagrams by employing special coordinate transformations to map infinite distances to finite ones. These Penrose diagrams allow the depiction of topological and causality relationships among parallel universes. Another common approach concentrating upon spatial geometry is the computation of "embedding surfaces" (see 2.3.2). These allow use of our intuitive notion of flat space to interpret the properties of a certain surface in curved space. The visualization of light paths and particle trajectories is useful for investigating special properties and non-local causality relationships. Beside visualizing geodesics within a coordinate frame (e.g. done by Bryson [Bry92]

in an interactive environment or Andrew and Fleming [AF92] using a CRAY), light-like geodesics are also used to study the visual appearance of a curved spacetime. This technique, known as relativistic raytracing, allows a direct comparison of astronomical observations with predictions of Gravitational Lens theory (a review can be found in Schneider, Ehlers and Falco [SEF99]).

The term "tensor field" is very general, and many kinds of tensor fields occur in general relativity. Our main focus will be on tensor fields of rank two. These also occur in other domains such as computational fluid dynamics and theory of elastic bodies. A straightforward approach involves the rendering of ellipsoids (p.128). Several iconic techniques have been developed to emphasize certain features, such as the Haber glyph [Hab90], p.135, and the Reynolds glyph [MSM95], p.135. Integrating eigenvectors is a frequent approach, as with Delmarcelle and Hesselink's method of hyperstreamlines [DH93]. In recent years new developments in medical image acquisition allow measurement of the so-called diffusion tensor field in a human brain, leading to a revived interest in appropriate visualization methods. Beside approaches to directly visualize the tensor data, e.g. via volume rendering as presented by Kindlmann et al. [KWH00], several methods have been specialized for tracking neuronal fibers.

The mathematical concept of a smooth manifold is used in many scientific simulations. Modeling it in a computer requires an appropriate discretization scheme. Frequently scientists approach this with ad-hoc, specific purpose implementations. This complicates reuse of algorithms that could profit from similarities in the discretization schemes. In particular, the description of data for transfer among applications suffers under incompatible approaches of modeling the same mathematical structures. The data generated by a numerical simulation needs to be organized in an appropriate way, such that some visualization system is able to interpret them. If we desire to involve more applications in the process, we face an n^2 problem, where we must equip n applications with each other's interface. Designing a common model is a question of active research, and no commonly accepted solution yet exists. This deficiency particularly troubled the U.S. ASCI project, where many different institutions had to cooperate on the common task of nuclear weapon design. As L.M. Cook and C.M. Matarazzo describe the problem [CM]: "Without a data model, data are just numbers, or, even worse, bits".

The mathematical language of general relativity is differential geometry. This allows the description of certain common manifolds as a bundle of a base space and a fiber space (see 2.1.3 for the mathematical definition). This concept has inspired Butler and Pendley [BP89] to conceive a data model for visualization purposes. Together with Bryson [BB92] they implemented a prototype. Their pioneering ideas were developed in the IBM Data Explorer [Res97], which is now known as OpenDX. Still, many visualization environments such as Amira [SWH04] and VTK [Kit] do not implement a generic data model, but follow an ad-hoc approach for each data type category. Butler himself continued classified work in on a common data model in the domain of the ASCI project, and some efforts were integrated into the design of the HDF5 [NCS03] file format.

1.3 Rationale for a New Approach

Relativistic visualization techniques for analytic solutions are of limited applicability for numerical relativity. Drawing light cones at one million points in a three-dimensional volume of a discretized manifold (a typical, but still rather small size for numerical simulations) does not lead to any visually interpretable result. Spacetime diagrams suffer from the need to omit spatial dimensions. The physical fraction of spacetime covered by a numerical simulation usually is too small to profit from Penrose diagrams. The differential equations leading to embedding surface do not necessarily have a solution at all. Raytracing a numerical spacetime (currently) requires too much computational time and memory to be used for interactive data exploration.

As seen from the theory of fiber bundles, the base space in general relativity is a four-dimensional differentiable manifold, and the fiber spaces are tangential spaces or their generalizations. The tangential spaces are the host of tensor fields of various flavors. Finding a unification scheme for the many discretization approaches of the underlying base space is a topic of research by itself. In the context of general relativistic tensor fields, we must also pay attention to non-trivial data structures on the fiber space, as there exist tensors with covariant and contra-variant indices, of rank two or more, and non-tensors such as the Christoffel symbols. It is common for numerical relativity simulations to have up to 100 quantities per point in space. For instance, the central tensor describing the curvature of spacetime – known as the Riemann curvature tensor, discussed on p. 34 – consists of 256 components in a four-dimensional spacetime.

For the purpose of visualizing general relativistic tensor fields, significant work has to put in

into operations on the fiber space. It is thus reasonable to implement these operations on a data model organized as a fiber bundle to avoid re-implementation efforts for changes of the base space, e.g. due to different discretization schemes. Here, we take up and extend Butler's ideas (reviewed in section 2.2.2). The concept of a fiber bundle is used in various ways, leading to a hierarchical data structure of five levels. Beyond the formulation of a manifold with tangential spaces as fiber bundle, the new model also utilizes fiber bundles to model data fields given on the skeletons (e.g. vertices, edges, faces, cells) of a computational grid. It introduces the idea of abstract identifiers when dealing with data entities instead of operating on the data itself, thereby achieving a level of abstraction similar to coordinate-free formulations in general relativity. Based upon this abstraction level, it allows unification of frequently required operations on computational grids such as *coordinate transformations*, *grid interpolations*, *grid evaluations*, and *coefficient evaluations* via a common interface. The distinction between base space and fiber space allows high reutilization of algorithms. It is demonstrated that by utilizing modern programming techniques such as C++ template metaprogramming the data model imposes no runtime overhead, whereas there is no previously published data model provides the same range of functionality. Its potential applicability extends well beyond just visualization of general relativistic tensor fields. It can be used as core architectural component of any visualization environment or numerical simulation, as well as for the structuring of file formats and network protocols. A common data model becomes especially crucial in the context of Grid computing where many applications are supposed to communicate. The issues of Grid computing in conjunction with numerical relativity are discussed in more detail by [BFN+99, BHM+00, HBM+00] and Allen et al. in [ABD+01a, ABD+01b]. Here, we will concentrate on the specific task of visualizing relativistic tensor fields with aspects of Grid computing as background perspective.

Building upon the suitable basis of an appropriate data model, the final goal of visualizing tensor fields by means of computer graphics is addressed via various newly conceived techniques, each of them specialized for certain purposes. Analytically expressed "simple" spacetimes are used to build an intuition for how various methods visualize curved spacetime. There is no claim in this thesis that an ultimate solution has been found, but a set of new or adapted techniques (among them those listed in fig. 1.1) is presented and analyzed for their applicability to general relativity. Many existing approaches only compute and visualize scalar and vector quantities derived from a tensor field, whereas the main goal of the methods conceived here is to display all tensorial quantities at the same time (see fig. 1.2 for a demonstration of the difference). Conceiving such rendering techniques for creating intuitively perceptible images extends in some respects into artistic dimensions, see also our contribution to the Conference on the Interaction of Science and Art [BHH00]. As an example of similar approaches, D. Laidlaw et al. [LAK+98] were inspired by the oil paintings of van Gogh when developing their tensor field visualization method. The novel technique of "Tensor Splats" is one of the most powerful techniques considering its ability to display intuitively interpretable images. It is well-suited for displaying evolving features in general relativistic tensor fields and enables to comprehend the geometrical properties of a numerical spacetime. Tensor splats can be rendered quickly enough even in a three-dimensional volume, thereby supporting interactive usage while still providing images of high quality.

1.4 Overview

The mathematical context of this dissertation will be reviewed in chapter 2, *Theoretical Background*. Chapter 3, *Data Model Design and Architecture*, presents the design of the newly conceived data model and techniques for efficient implementation using modern features of the C++ programming language. New and adapted rendering algorithms are discussed and compared in chapter 4, *Rendering Algorithms*. The applicability of the data model and tensor field rendering methods beyond general relativity is demonstrated in the appendix A.

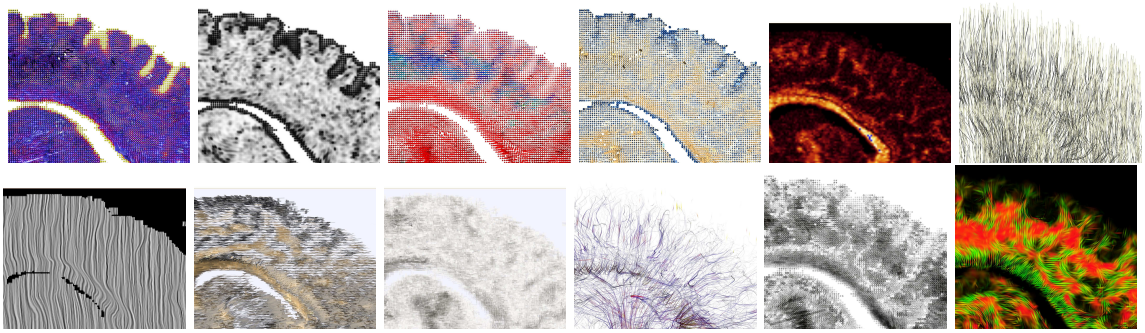


Figure 1.1: An overview of various tensor field visualization methods applied to a slice through an MRI-aquired "diffusion tensor" field of a human brain.

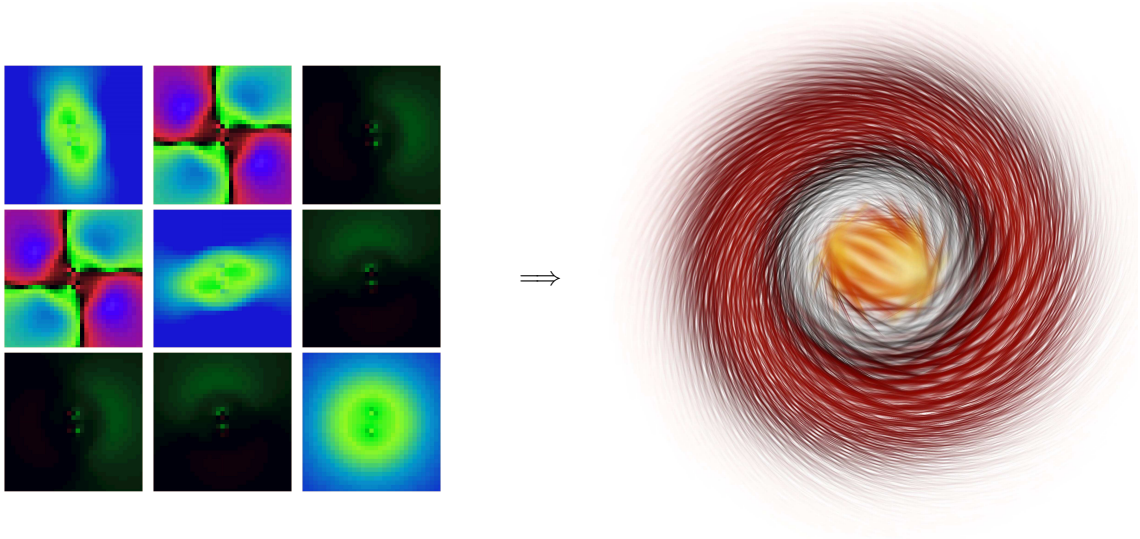


Figure 1.2: The maelstrom in spacetime produced by two colliding black holes – the output of a numerical simulation – is hard to see using standard "scalar field" visualization methods (left: components of the tensor field, see also fig. 4.70). It is easily perceptible using appropriate tensor field visualization methods (right: method of "Tensor Splats", see also fig. 4.47).

Chapter 2

Theoretical Background

In the first section the basic terms of differential and computational geometry are reviewed, starting from the ground up by defining topological spaces from sets with certain properties. These definitions are used when discussing discretized numerical spacetimes in the next section 2.2. The concept of a manifold with charts is a central part of general relativity. It motivates the formalism of coordinate-free expressions that allow highly abstract, powerful expressions. This abstraction provides an ideal "mathematical implementation" of the physical principle of covariance and therefore motivates the structure and terms of the data model as presented in the succeeding chapter as an appropriate basis for performing visualization tasks on tensor fields originating especially from general relativity. Here, we also derive the important term "acceleration" in a coordinate-free way using the non-standard, but simple and elegant concept of the "tangential transport" (not to be confused with the frequently used "parallel transport"), as it was invented by P. Wagner [Wag94]. This approach demonstrates that acceleration is not just "the second derivative by the time coordinate", but involves structures beyond tensor fields. A data model that is supposed to be appropriate for dealing with general relativistic data and their visualization must be able to support all the mathematical structures that are required for a correct description of acceleration (and other quantities, like curvature tensors). The concept of a fiber bundle is an appropriate framework and is presented at the end of this section. It has inspired Butler and Pendley [BP89] to conceive a data model for visualization purposes. Here, we take up on this idea and extend the original idea even further.

2.1 Mathematical Concept of a Continuous Spacetime

2.1.1 Manifolds and Tangential Spaces

We start with the assumption that the terms "set" and "power set" are known to the reader and may now straightforwardly define the term "topology":

Definition 1 *Let X be a set and $\mathcal{P}(X)$ be the power set of X. A subset $\mathcal{T} \subseteq \mathcal{P}(X)$ of the power set is a* ***topology*** *iff*

I.) arbitrary unions of elements of $\mathcal{T}$ are contained in $\mathcal{T}$, i.e. if I is an arbitrary set of indices and $\forall i \in I$: $U_i \in \mathcal{T}$, then $\bigcup_{i \in I} U_i \in \mathcal{T}$,

II.) finite intersections of elements of $\mathcal{T}$ are contained in $\mathcal{T}$, i.e. if $U_0, \ldots, U_n \in \mathcal{T}$ then $\bigcap_{i=0}^{n} U_i \in \mathcal{T}$ with $n \in \mathbb{N}$,

III.) the empty set and the set X itself are contained in $\mathcal{T}$, i.e. $\emptyset, X \in \mathcal{T}$

Definition 2 *The pair $(X, \mathcal{T})$ of a set together with a topology on this set is a* topological space *(possible implementations shown in 2.2.2 and 3.1.1). The elements of a topological space are called* points.

EXAMPLES:

1. The set $X = \{0, 1, 2, 3\}$ with the topology $\mathcal{T} = \{\emptyset, \{0\}, \{1, 2, 3\}, \{0, 1, 2, 3\}\}$ is a topological space.
2. The set $X = \{0, 1, 2, 3\}$ with the topology $\mathcal{T} = \{\emptyset, \{0\}, \{0, 1, 2, 3\}\}$ is a topological space as well.
3. The set $X = \{0, 1, 2, 3\}$ with the subset $\mathcal{T} = \{\emptyset, \{0\}, \{1\}, \{0, 1, 2, 3\}\}$ of the power set $\mathcal{P}(X)$ is **not** a topological space because the union $\{0, 1\}$ of two elements of $\mathcal{T}$ is not contained in $\mathcal{T}$.
4. The set $X = \{0, 1, 2, 3\}$ with the full power set $\mathcal{T} = \mathcal{P}(X)$ is a topological space. Its topology is called the *discrete topology* and is the *finest topology* possible. It contains $16 = 2^4$ elements in the given example. In general, the cardinality (number of elements) $\#\mathcal{P}(X)$ of a power set $\mathcal{P}(X)$ is related to the cardinality of the set $\#X$ via $\#\mathcal{P}(X) = 2^{\#X}$: each element of X can or cannot be part of $\mathcal{P}(X)$, which results in $2^{\#X}$ possibilities.
5. A set X with the topology $\{\emptyset, X\}$ is a topological space, its topology is called the *trivial topology* and is the *coarsest topology* possible. It always contains just 2 elements.
6. $\mathbb{R}$ with the set of open intervals $\mathcal{T} = \left\{ \bigcup_i (a_i, b_i) : a_i, b_i \in \mathbb{R}, a_i < b_i \right\}$ is known as the *standard topology* on $\mathbb{R}$.

Definition 3 *Let $(X, \mathcal{T})$ be a topological space. A subset $U \subset X$ is called **open** iff it is contained in the topology $U \in \mathcal{T}$, i.e.*

$$U \subset X \quad \textit{open} \quad \Longleftrightarrow \quad U \in \mathcal{T} \quad .$$

*A subset $V \subset X$ is **closed**, if its complementary set $X \backslash V$ is open.*

For the examples given before we find:

1. The set $\{0\}$ is open in example 1.), because it is contained in $\mathcal{T}$. It is also closed, because its complementary set $\{1, 2, 3\}$ is contained in $\mathcal{T}$ as well.
2. In the topological space from example 2.) the set $\{0\}$ is open and $\{1, 2, 3\}$ is closed. The set $\{0, 1\}$ is neither open nor closed, because it is not contained in the topology, nor is its complementary set $\{2, 3\}$.
3. The set $(1, 2)$ is open in example 6.), the set $[1, 2]$ is closed ($[1, 2]$ is not in $\mathcal{T}$).

The empty set $\emptyset$ and the full set X are always open and closed at the same time.

Definition 4 *A space X, where no other subsets than $\emptyset$ and X are open and closed at the same time, is called **connected**.*

EXAMPLE: $\mathbb{R}$ is connected, $\mathbb{R} \backslash \{0\}$ is not connected.

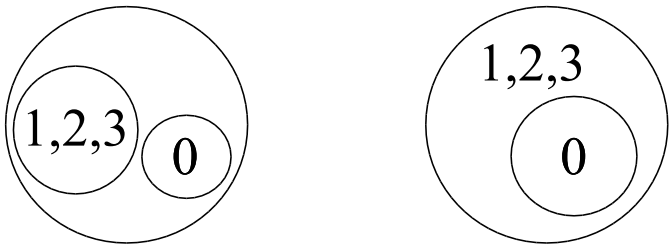


Figure 2.1: Different topologies on the set $\{0, 1, 2, 3\}$: The topological space of example 1.) (left figure) is not connected (it contains two disjoint sets), whereas 2.) (right figure) is connected (each set of the topology contains the set $\{0\}$).

Definition 5 *A subset $A \subseteq X$ of a topological space $(X, \mathcal{T})$ is a **neighborhood** of an element $p \in X$ iff it contains an element O of $\mathcal{T}$ that contains p:*

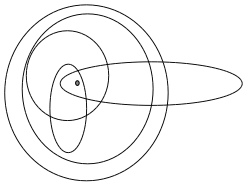
$$A \subseteq X \quad \textit{neighborhood of } p \Longleftrightarrow \exists O \in \mathcal{T} : p \in O, O \subseteq A$$

Note that a neighborhood is not necessarily contained in $\mathcal{T}$, i.e. a neighborhood is not necessarily an open or closed set.
EXAMPLES:

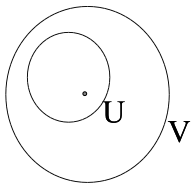
1. The set $\{0, 1, 2\}$ is a neighborhood of the point 0 in the topological space from example 1.), but it is not an element of the given topology. However, the set $\{0\}$ is a neighborhood of 0 that is contained in the topology.
2. The interval $[1, 3]$ is a neighborhood of the point 2, but is not contained in the standard topology of $\mathbb{R}$, whereas $(1, 3)$ is a neighborhood that is contained.

An alternative, less frequently used definition of topological spaces starts with the definition of (neighborhood) sets $\mathcal{U}(p) \subset \mathcal{P}(X)$ of neighborhoods $U \subseteq X$ and requires (note that $\mathcal{U}(p)$ is a set of sets - the "neighborhood set" or "set of neighborhoods", whereas U is a set, a "neighborhood"):

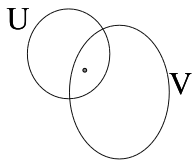
I.) Each element of the neighborhood set contains the point $p : \forall U \in \mathcal{U}(p) : p \in U$ and X always is a neighborhood, i.e. $X \in \mathcal{U}(p)$ (which can also be seen as a consequence of III.).
Consequently, the neighborhood set is never empty $\mathcal{U}(p) \neq \emptyset$.



II.) A superset of a neighborhood is a neighborhood, too: if $U \subset V$ and $U \in \mathcal{U}(p)$ then $V \in \mathcal{U}(p)$.

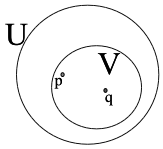


III.) The intersection of neighborhoods of the same point yields a neighborhood: if $U, V \in \mathcal{U}(p)$ then $U \cap V \in \mathcal{U}(p)$. More general, finite intersections of neighborhoods yield neighborhoods again: if $U_i \in \mathcal{U}(p)$, then $\bigcap_{i=0}^{n} U_i \in \mathcal{U}(p)$ with $n \in \mathbb{N}$. The zero intersection $n = 0$ is the set X, i.e. the set X is contained in the neighborhood sets (alternatively to I.).



IV.) If U is a neighborhood of p, then there exists a (usually smaller) neighborhood V such that U is a neighborhood of each point of V: $\forall U \in \mathcal{U}(p) \ : \ \exists V \in \mathcal{U}(p)$ such that $\forall q \in V \ : \ U \in \mathcal{U}(q)$.

EXAMPLE: If $U = [0, 5]$ a neighborhood of $p = 2$, then $V = [1, 4]$ is such a neighborhood of the point $q = 3$.



A topological space can thus be constructed by

- a (base) set X of elements ("points"), the *topological base*
- for each element $p \in X$ a set of neighborhoods $\mathcal{U}(p)$ whereby $\forall U \in \mathcal{U}(p) : p \in U$
- for each two neighborhoods $U, V \in \mathcal{U}(p)$ there exists another neighborhood W, which is contained in both: $\forall U, V \in \mathcal{U}(p) : \exists W \in \mathcal{U}(p) : W \subset U \cap V$

These rules motivate the construction of "`Topology` objects" as part of the fiber bundle data model as presented in the next chapter via "sets with neighborhood information".

Definition 6 *Two topological spaces X, Y are called **homeomorphic**, if there exists a bijective map $H : X \to Y$ such that open sets of X are mapped to open sets in Y and vice versa, i.e. the neighborhood relations must be sustained under this mapping. H is called a homeomorphism or topological map.*

A common task of algebraic topology is to determine if two spaces are homeomorphic. *Algebraic invariants* provide means to solve this problem. If the invariants of two spaces differ, then they cannot be homeomorphic. One of these invariants is the *Lebesgue covering dimension* of a topological space. Its definition is given here to show that the term "dimension" can be defined even for non-manifolds, such that we can assign a dimensional number to any topological space - the "dimension" will thus be one property of `Topology` objects, even if they do not describe manifolds.

Definition 7 *A cover of a set X is a set of nonempty subsets of X whose union is X, i.e. $C \in \mathcal{P}(X)$ is a cover iff $\bigcup_{U \in C} U = X$.*

A cover is called "open" (open cover of a topological space), if it is contained in the topology of the respective space.

Definition 8 *A topological space has **Lebesgue covering dimension** n if any open cover C has a second open cover D (the* refinement*), where each element $d \in D$ is a subset of an element in the first cover, i.e. $d \subset c$ with $c \in C$, such that no point is included in more than $n+1$ elements.*

If a topological space is homeomorphic to $\mathbb{R}^n$ with $n \in \mathbb{N}$, then its dimension is n. If a space does not have Lebesgue covering dimension n for any $n \in \mathbb{N}$, it is said to be infinite dimensional. The dimension of the empty set $\emptyset$ is defined as -1. An alternative definition of the dimension of a topological space is given by the Hausdorff dimension, which allows a generalization of the term "dimension" also to non-integer numbers (fractal dimensions). However, this branch will not be addressed here.

Definition 9 *Let X, Y be two sets. The **cartesian product** $X \times Y$ of the **sets** X and Y is defined as the set of pairs (x, y) of elements from both spaces with $x \in X, y \in Y$.*

This definition will be directly casted into "Product Arrays", as described in 3.1.2.

Definition 10 *The **cartesian product** $X \times Y$ of two **topological spaces** X, Y with the respective neighborhood sets $\mathcal{U}(x) \subset \mathcal{P}(X), \mathcal{V}(y) \subset \mathcal{P}(Y)$ of the points $x \in X, y \in Y$ is a topological space, if the neighborhood sets $\mathcal{U}(x, y)$ of the point $(x, y) \in X \times Y$ are given by $\mathcal{U}(x, y) = \{W : W \supset U \times V : U \in \mathcal{U}(x), V \in \mathcal{V}(y)\}$.*

The according topology is called the *product topology*. Multiple self-products of spaces are denoted as a power $X^n := \underbrace{X \times X \dots X}_{n}$.

Definition 11 *A set X together with a function $d : X \times X \mapsto \mathbb{R}$ is a **metric space** (X, d), if d fulfills the following three properties for all $x, y, z \in X$:*

I.) $d(x, y) = d(y, x)$ *(symmetric)*

II.) $d(x, y) = 0 \Leftrightarrow x = y$ *(not degenerated)*

III.) $d(x, z) \leq d(x, y) + d(y, z)$ *(triangle inequality, positive definite)*

*d is called a **distance function** or (topological) **metric**.*

Sometimes the property $d(x, y) \geq 0$ (positive definite) is explicitly included in literature, but it actually follows when inspecting the case $x = z$ in *III.)* due to $0 = d(x, x) \leq 2d(x, y)$, so this is redundant information and can be omitted. A metric which allows $d(x, y) = 0$ also for $x \neq y$, i.e. replacing condition II.) by the weaker condition $d(x, x) = 0$, is called a *pseudo-metric*. Note that the Minkowski metric and spacetime metrics used in general relativity violate the triangle inequality condition *III.)* and thus do not obey the definitions of a *topological metric* at all; this rarely discussed issue will be addressed in the next paragraph.

Definition 12 *Two metric spaces $(X, d_x), (Y, d_y)$ are **isometric**, if a bijective map f, called the **isometry**, between these two metric spaces exists that preserves distances, i.e.,*

$$f : (X, d_x) \to (Y, d_y) \text{ isometry} \quad \Longleftrightarrow \quad d_y(f(x_1), f(x_2)) = d_x(x_1, x_1) \quad \forall x_1, x_2 \in X$$

Definition 13 *Let (X, d) be a metric space. An **open (ε-)ball** at a point $p \in X$ is the subset $\{q \in X | d(p, q) < \varepsilon\}$, $\varepsilon \in \mathbb{R}^+$ of a metric space (X, d). A **closed ball** is the set $\{q | d(p, q) \leq \varepsilon\}$ with $\varepsilon > 0$.*

Note that the geometrical shape of a ball may be quite different from a sphere; it depends on the given metric. Using the standard Euclidean metric on $\mathbb{R}^n$, a ball has the intuitively expected spherical shape.

A certain metric d *induces* a topology on X, namely $\mathcal{T} = \{\bigcup_i B_i : B_i \text{ open } \varepsilon - \text{ball}\}$, with the ε-balls as topological base. This topology is called a *metric topology.*

Definition 14 *Two metrices d_1, d_2 on a given space X are called **equivalent**, if they sustain qualitatively the same distance relationships, i.e.*

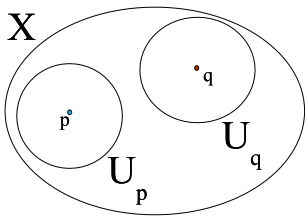
$$d_1, d_2 \text{ equivalent} \Longleftrightarrow \exists C_1, C_2 > 0 : \forall x, y \in X : C_2\, d_2(x,y) \leq d_1(x,y) \leq C_1\, d_2(x,y)$$

It follows that if two metrices are equivalent, then the topologies induced by them are identical, but the metric spaces are not necessarily isometric:

$$(X, d_1), (X, d_2) \text{ isometric} \overset{\not\Leftarrow}{\Rightarrow} d_1, d_2 \text{ equivalent} \overset{\not\Leftarrow}{\Rightarrow} \mathcal{T}_1 = \mathcal{T}_2$$

EXAMPLE: Equivalent metrices on $\mathbb{R}^n$:

1. L^1-norm: $d_1(x,y) = \sum_{i=1}^{n} |x_i - y_i|$ ..... ε-balls are diagonal n-cubes
2. L^2-norm: $d_2(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$ ..... ε-balls are n-spheres
3. L^p-norm: $d_p(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$
4. L^∞-norm: $d_\infty(x,y) = \max|x_i - y_i|$ ..... ε-balls are axis-aligned n-cubes



Definition 15 *A topological space X is a **Hausdorff** [Hau14] space iff for any two distinct points $x, y \in X$ there exist two distinct neighborhoods U_x, U_y such that $U_x \cap U_y = \emptyset$:*

$$X \text{ Hausdorff} \Longleftrightarrow \forall p, q \in X \text{ with } p \neq q \;:\quad \exists U_p \in \mathcal{U}(p)\ \exists U_q \in \mathcal{U}(q) \;:\quad U_p \cap U_q = \emptyset$$

EXAMPLES: The topological spaces from example 2.) and 3.) are not Hausdorff spaces, whereas example 4.) is a Hausdorff space.
REMARK: Finite topological spaces are Hausdorff spaces iff their topology is the full power set (i.e. the discrete topology). A set with the trivial topology is not Hausdorff (with the exception of sets that contain only one element, where no distinct points exist at all).

Definition 16 *A **manifold** is a Hausdorff space that is locally homeomorphic to Euclidean space $\mathbb{R}^n$.*

EXAMPLE: Two crossing lines form a topological Hausdorff space, which is not a one-dimensional manifold (although it can be embedded in a two-dimensional manifold).

Topology of Spacetime

The term "metric" is somewhat ambiguous when considering spacetimes; it still is a distance function, thus deserving the name "metric", but does not fulfill the requirements of a metric or pseudo-metric that are required to construct metric spaces. Strictly speaking, while a Riemannian metric (only positive distances allowed) is a (topological) metric in the sense of def. 11, a pseudo-Riemannian metric (which allows negative distances) is not even a (topological) pseudo-metric. This "spacetime metric" is no longer positive definite and thus the triangle inequality III.) no longer holds in a relativistic spacetime. For instance, a triangle may easily be constructed with two sides of zero length (which are light-like) and any other side which has non-zero length.

This apparent contradiction is accomplished by **not** using the metric topology that is induced by the spacetime (e.g. Minkowski) metric (which would not yield a topological space at all) on the underlying space, but to assign the Euclidean topology to Minkowski spacetime. The approach of constructing a metric space in general relativity is thus different to constructing a metric space from a (topological) metric function. The Einstein equations cannot be used to determine the topology of the spacetime manifold, but rather the topology needs to be imposed before a solution can be found. For instance, a spacetime containing multiple black holes may be constructed from different topologies: "Misner" initial data presumes that the throats connect a pair of asymptotically flat spacetimes which are identical to each other, whereas "Brill-Lindquist" initial data connects each throat to a separate asymptotically flat region.

Differentiable Manifolds

Definition 17 *A chart* $\{x^\mu\}$, $\mu = 0 \dots n-1$ *is a homeomorphic mapping assigning an n-tuple of real-valued numbers to each point* $p \in M$ *in a manifold* M*:*

$$\begin{aligned} \{x^\mu\} \quad &: \quad M \longrightarrow \mathbb{R}^n \\ &\qquad p \longmapsto (x^\mu(p)) \end{aligned}$$

The numbers $x^\mu(p)$ are the *coordinates* of the point p in the chart $\{x^\mu\}$ x^i is the i^{th} *coordinate function.*

A chart may also be defined on just an open subset of M and $\mathbb{R}^n$. Such a chart, which does not cover the full manifold M, is a *local chart.* Interpreting the term "chart" $x^\mu : M \to \mathbb{R}$ physically, it is actually a description how to measure certain numbers (the coordinates) for a given point $p \in M$. It is thus not possible to provide a further mathematical description of a chart function. Examples are the cartesian coordinates $\{x, y, z\}$ or the polar coordinates $\{r, \vartheta, \varphi\}$. E.g. the cartesian coordinate function x is defined by a length measurement, whereas the polar coordinate function ϑ is the measurement of an angle.

Once a chart is defined via its coordinate functions, further charts can be constructed from it via *coordinate transformations*: Let $\{x^\mu\}$ and $\{x^{\bar{\mu}}\}$ be charts. (Notation: for practical reasons we distinguish between charts via their indexing, i.e. $1 \neq \bar{1}$, $2 \neq \bar{2}$ etc.). The inverse mapping $\{x^\mu\}^{-1}$ of a chart $\{x^\mu\}$ (which exists because $\{x^\mu\}$ is bijective) is a map $\mathbb{R}^n \supseteq I \to M$. The $\bar{\mu}$-th coordinate function $x^{\bar{\mu}}(p), p \in M$ thus can be expressed via the coordinate function $x^\mu(p)$ as a composition:

$$\{x^{\bar{\mu}}\}(p) = \{x^{\bar{\mu}}\} \circ \{x^\mu\}^{-1} \circ \{x^\mu\}(p)$$

The resulting map $\{x^{\bar{\mu}}\} \circ \{x^\mu\}^{-1}$ is a map $\mathbb{R}^n \to \mathbb{R}^n$ (called the *transition functions*) and can thus be written in terms of algebraic expressions.

Definition 18 *A set of (local) charts which all together cover the full manifold* M *with respective coordinate transformations which are* $\mathcal{C}^\infty$*, is a (*$\mathcal{C}^\infty$*-)**atlas**.*

An atlas A is *compatible* with another atlas $\bar{A}$, if their respective coordinate transformations (i.e. the transition functions) are $\mathcal{C}^\infty$. The union of all atlases which are compatible to a certain atlas is a *maximal* or *complete atlas.* It contains all possible charts for a given manifold. On the other hand, a set of charts which contains the minimal number of charts that are required for a manifold, is a *minimal atlas.*

Definition 19 *A **differentiable manifold** is a manifold (def. 16) with an atlas.*

The *cartesian product* $A \times B$ of two manifolds A and B reveals a manifold $A \otimes B$, where for each pair of points $p \in A$ and $q \in B$ with coordinates (x^i) and (y^j) the point $p \times q$ is mapped to the coordinates (x^i, y^j).

Curves, Lines and Tangential Vectors

Definition 20 *A* **curve** q *is a continuous map* $q : \mathbb{R} \to M : s \mapsto q(s)$*, which provides a point* $q(s) \in M$ *for a number* $s \in \mathbb{R}$*.*

s is called the *curve parameter.* The composition of a coordinate function and a curve function yields n maps $q^\mu(s) : \mathbb{R} \to \mathbb{R}$ where n is the dimension of the manifold M. These n functions are the *representations* of the curve q in the chart $\{x^\mu\}$:

$$q^\nu(s) := x^\nu(q(s)) \equiv x^\nu \circ q(s) \tag{2.1}$$

The set of all points of a curve within a manifold is a *line*. A certain line can be described by an infinite number of curves with different parameterizations. The inverse map of coordinate functions $(x^\mu)^{-1}$ are maps $\mathbb{R} \to M$ and thus are curves as well. Their corresponding lines are called *coordinate lines*. They are the set of all points $p \in M$ with all coordinate values fixed except one. (The "inverse of a measurement description": do measure all points and mark those which yield the same number.)

Definition 21 *The **tangential vector** on a curve $q(s)$ is defined as the differential operation $\frac{d}{ds}$, which maps a function $f : M \to \mathbb{R}$ to a number*

$$\frac{d}{ds} f \equiv \frac{d}{ds} f(q(s)) := \lim_{h \to 0} \frac{f(q(s+h)) - f(q(s))}{h} \in \mathbb{R} \tag{2.2}$$

for each value s.

Note that $f \circ q : \mathbb{R} \to \mathbb{R}$, so the derivation can be computed algebraically. In a given chart $\{x^\mu\}$ this operation can be written as:

$$\begin{aligned} \frac{d}{ds} f(q(s)) = \frac{d}{ds} f(x^0(q(s)), x^1(q(s)), x^2(q(s)), \dots) = \\ = \frac{\partial f}{\partial x^\mu} \frac{dx^\mu(q(s))}{ds} = \frac{dx^\mu(q(s))}{ds} \frac{\partial}{\partial x^\mu} f = \frac{dq^\mu(s)}{ds} \frac{\partial}{\partial x^\mu} f =: \dot{q}^\mu(s) \frac{\partial}{\partial x^\mu} f \end{aligned} \tag{2.3}$$

Here we made use of Einstein's sum convention, that tells to implicitly sum over repeated covariant [lower] and contravariant [upper] indices, as if $\sum_\mu$ were written above. Any tangential vector can be written as a linear combination of derivatives by the coordinate functions:

$$\frac{d}{ds} = \dot{q}^\mu(s) \frac{\partial}{\partial x^\mu} =: \dot{q}$$

The functions $\dot{q}^\mu$ are the **components** of the tangential vector $\dot{q}$ in the chart (see also Hawking and Ellis [HE73], p.15f). If f is a coordinate function x^ν, then $\frac{d}{ds}$ yields the ν-th component $\dot{q}^\nu$:

$$\frac{d}{ds} x^\nu(q(s)) = \dot{q}^\mu(s) \frac{\partial x^\nu}{\partial x^\mu} = \dot{q}^\mu(s) \delta^\nu_\mu = \dot{q}^\nu(s)$$

Definition 22 *The partial derivatives by the coordinate functions fulfill the vector space axioms:*

$$(a \frac{\partial}{\partial x^\mu} + b \frac{\partial}{\partial x^\nu}) f = a(\frac{\partial}{\partial x^\mu} f) + b(\frac{\partial}{\partial x^\nu} f)$$

*(with f being a function $M \to \mathbb{R}$ and $a, b \in \mathbb{R}$), and thus span a vector space, which is called the **tangential space** $T_p(M)$ of the manifold M at the point p.*

The coordinate functions x^μ induce a basis $\{\frac{\partial}{\partial x^\mu}\}$ in $T_p(M)$. We will use the notation

$$\partial_\mu := \frac{\partial}{\partial x^\mu}$$

for the basis vectors of a tangential space. Sometimes the notation $\vec{\partial}_\mu$ is used to emphasize the vector property of the partial derivative.

Chart Transition of Tangential Vectors The basis vectors of the tangential spaces $\{\vec{\partial}_\mu\}$ in a chart$\{x^\mu\}$ are related to their representation $\{\vec{\partial}_{\bar{\mu}}\}$ in another chart $\{x^{\bar{\mu}}\}$ via a linear combination:

$$\vec{\partial}_{\bar{\mu}} \equiv \frac{\partial}{\partial x^{\bar{\mu}}} = \frac{\partial x^\mu}{\partial x^{\bar{\mu}}} \cdot \frac{\partial}{\partial x^\mu} =: \alpha^\mu_{\bar{\mu}} \cdot \vec{\partial}_\mu$$

$\alpha^\mu_{\bar{\mu}}$ are abbreviations for the inner derivatives of the transition functions:

$$\alpha^\mu_{\bar{\mu}} := \frac{\partial x^\mu(x^{\bar{\mu}})}{\partial x^{\bar{\mu}}} \qquad \alpha^{\bar{\mu}}_\mu := \frac{\partial x^{\bar{\mu}}(x^\mu)}{\partial x^\mu} \tag{2.4}$$

A tangential vector $v \in T_p(M)$ can be represented in various charts:

$$v = v^\mu \partial_\mu = v^{\bar{\mu}} \partial_{\bar{\mu}} = v^{\bar{\mu}} \left(\alpha^\mu_{\bar{\mu}} \partial_\mu\right)$$

Consequently, the components of a tangential vectors are transformed inversely as the basis vectors:

$$v^{\mu} = v^{\bar{\mu}} \alpha^{\mu}_{\bar{\mu}} \qquad \partial_{\mu} = \alpha^{\bar{\mu}}_{\mu} \partial_{\bar{\mu}}$$

This transformation rule guarantees that operations on the fundamental geometric object v are independent from the used chart. The physical idea behind is

Definition 23 *The **principle of covariance**: The laws of physics have to be formulated independently of the observer and must not depend on the choice of a certain chart.*

EXAMPLE: The Newtonian law $F = m \cdot a$ is valid in any coordinate system, not only in so-called "inertial systems". The force F between physical objects is clearly independent from the observer, as is the mass m, too (in Newtonian physics). Accordingly, the acceleration has to be a quantity that is independent from the observer. However, not for all observers the acceleration is simply the second derivative $\ddot{q}$ of a curve q, as we will see in def. 46. If for a certain observer the second derivative of a line is identical to its acceleration, he is called "inertial". Other observers, e.g. those in a rotating coordinate system, which wrongly interpret themselves as being inertial, will find non-physical ("virtual") forces (e.g. "centrifugal force", "coriolis force") to occur, i.e. forces which depend on the motion of the observer, but not on the (kinematic, dynamic) properties of the objects under investigation themselves. We will discuss a coordinate-free definition of the term "acceleration" on pp.29.

The inner derivatives $\alpha^{\mu}_{\bar{\mu}}$ can be written as $n \times n$ coordinate transformation matrices. To handle the transition between two charts, actually four such coordinate transformation matrices are required. Each of the two transformation matrices (for both transition directions) needs to be provided in both charts.

EXAMPLE: Transformation matrices in $\mathbb{R}^3$, with cartesian $\{x^{\mu}\} \rightarrow$ polar $\{x^{\bar{\mu}}\}$ in cartesian coordinates:

$$\alpha^{\mu}_{\bar{\mu}}(x,y,z) = \begin{pmatrix} \frac{\partial x}{\partial r} = \frac{x}{\sqrt{x^2+y^2+z^2}} & \frac{\partial x}{\partial \vartheta} = \frac{x \cdot z}{\sqrt{x^2+y^2}} & \frac{\partial x}{\partial \varphi} = -y \\ \frac{\partial y}{\partial r} = \frac{y}{\sqrt{x^2+y^2+z^2}} & \frac{\partial y}{\partial \vartheta} = \frac{y \cdot z}{\sqrt{x^2+y^2}} & \frac{\partial y}{\partial \varphi} = x \\ \frac{\partial z}{\partial r} = \frac{z}{\sqrt{x^2+y^2+z^2}} & \frac{\partial z}{\partial \vartheta} = -\sqrt{x^2+y^2} & \frac{\partial z}{\partial \varphi} = 0 \end{pmatrix}$$

$$\alpha^{\bar{\mu}}_{\mu}(x,y,z) = \begin{pmatrix} \frac{\partial r}{\partial x} = \frac{x}{\sqrt{x^2+y^2+z^2}} & \frac{\partial r}{\partial y} = \frac{y}{\sqrt{x^2+y^2+z^2}} & \frac{\partial r}{\partial z} = \frac{z}{\sqrt{x^2+y^2+z^2}} \\ \frac{\partial \vartheta}{\partial x} = \frac{x \cdot z}{\sqrt{x^2+y^2 \cdot (x^2+y^2+z^2)}} & \frac{\partial \vartheta}{\partial y} = \frac{y \cdot z}{\sqrt{x^2+y^2 \cdot (x^2+y^2+z^2)}} & \frac{\partial \vartheta}{\partial z} = -\frac{\sqrt{x^2+y^2}}{x^2+y^2+z^2} \\ \frac{\partial \varphi}{\partial x} = -\frac{y}{x^2+y^2} & \frac{\partial \varphi}{\partial y} = \frac{x}{x^2+y^2} & \frac{\partial \varphi}{\partial z} = 0 \end{pmatrix}$$

Transformation matrices polar $\{x^{\bar{\mu}}\} \rightarrow$ cartesian $\{x^{\mu}\}$ in polar coordinates:

$$\alpha^{\mu}_{\bar{\mu}}(r,\vartheta,\varphi) = \begin{pmatrix} \frac{\partial x}{\partial r} = \sin\vartheta \ \cos\varphi & \frac{\partial x}{\partial \vartheta} = r \ \cos\vartheta \ \cos\varphi & \frac{\partial x}{\partial \varphi} = -r \ \sin\vartheta \ \sin\varphi \\ \frac{\partial y}{\partial r} = \sin\vartheta \ \sin\varphi & \frac{\partial y}{\partial \vartheta} = r \ \cos\vartheta \ \sin\varphi & \frac{\partial y}{\partial \varphi} = r \ \sin\vartheta \ \cos\varphi \\ \frac{\partial z}{\partial r} = \cos\vartheta & \frac{\partial z}{\partial \vartheta} = -r \ \sin\vartheta & \frac{\partial z}{\partial \varphi} = 0 \end{pmatrix}$$

$$\alpha^{\bar{\mu}}_{\mu}(r,\vartheta,\varphi) = \begin{pmatrix} \frac{\partial r}{\partial x} = \sin\vartheta \ \cos\varphi & \frac{\partial r}{\partial y} = \sin\vartheta \ \sin\varphi & \frac{\partial r}{\partial z} = \cos\vartheta \\ \frac{\partial \vartheta}{\partial x} = \frac{\cos\varphi \ \cos\vartheta}{r} & \frac{\partial \vartheta}{\partial y} = \frac{\sin\varphi \ \cos\vartheta}{r} & \frac{\partial \vartheta}{\partial z} = -\frac{\sin\vartheta}{r} \\ \frac{\partial \varphi}{\partial x} = -\frac{\sin\varphi}{r \ \sin\vartheta} & \frac{\partial \varphi}{\partial y} = \frac{\cos\varphi}{r \ \sin\vartheta} & \frac{\partial \varphi}{\partial z} = 0 \end{pmatrix}$$

Covectors

Let $v \in T_p(M)$ be a vector and $f : M \rightarrow \mathbb{R}$ a real-valued function on the manifold. Then applying the tangential vector v on f yields a number:

$$v(f) \in \mathbb{R}$$

In def. 21 we examined this expression for a fixed vector v and arbitrary function f. Alternatively, we can study it for arbitrary vectors v and a fixed function f. This way it defines a function df which maps a tangential vector v to a number $df(v) \in \mathbb{R}$:

$$\boxed{\begin{array}{lccl} df: & T_p(M) & \longrightarrow & \mathbb{R} \\ & v & \longmapsto & df(v) := v(f) \end{array}} \tag{2.5}$$

The function df is called a **1-form**. 1-forms fulfill the vector space axioms:

$$(a \cdot df + b \cdot dg)(v) = v(a \cdot f + b \cdot g) = a \cdot v(f) + b \cdot v(g) = a \cdot df(v) + b \cdot dg(v)$$

and thus span a vector space, called the cotangential space $T^*_p(M)$; its elements, called *covectors* (a direct visualization is given in 4.1.2), are the linear maps $T_p(M) \rightarrow \mathbb{R}$.

Coordinate Expression In a chart $\{x^\mu\}$ eq.(2.5) can be written in terms of the basis vectors ∂_μ, i.e. the partial derivatives by the respective coordinate functions:

$$df(v) = v(f) = v^\mu \partial_\mu(f) = v^\mu \frac{\partial f}{\partial x^\mu} =: v^\mu f_{,\mu} \tag{2.6}$$

Thus in a chart of a manifold of dimension n a covector is constructed by n numbers, like a tangential vector. In matrix notation[1], a tangential vector and covectors are written as a column and row vectors, such that their product yields a number:

$$v^\mu f_{,\mu} = \begin{pmatrix} f_{,1} & f_{,2} & \cdots & f_{,n} \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \in \mathbb{R}$$

If we take v as the μ-th basis vector ∂_μ, then we directly get the derivative of the function f by the μth coordinate function:

$$df(\partial_\mu) = \partial_\mu(f) = \frac{\partial f}{\partial x^\mu} = f_{,\mu}$$

On the other hand, if we let f be the μ-th coordinate function x^μ, we directly get the μth vector component:

$$dx^\mu(v) = dx^\mu(v^\nu \partial_\nu) = v^\nu \frac{\partial x^\mu}{\partial x^\nu} = v^\nu \delta^\mu_\nu = v^\mu$$

Therefore we see that a covector just "sees" the projection of a tangential vector in which the covector points. Any covector can be written as a linear combination of such "projections" along the coordinate functions:

$$df(v) = v^\mu f_{,\mu} = dx^\mu(v) \cdot f_{,\mu} = f_{,\mu} \cdot dx^\mu(v)$$

In other words, the coordinate functions $\{x^\mu\}$ induce a basis $\{dx^\mu\}$ in the cotangential space $T^*_p(M)$:

$$\boxed{df = f_{,\mu} dx^\mu \equiv \frac{\partial f}{\partial x^\mu} dx^\mu} \tag{2.7}$$

Equation (2.7) is also called the *total differential* of a function f. Often the notation of a scalar product is used for the application of a covector on a vector:

Definition 24 *The scalar product of a covector and a vector is defined as the action of the covector on the vector:*

$$< df, v >:= df(v)$$

This operation is also called an index contraction, because in a chart it appears as a cancellation of a lower index with upper index:

$$< df, v > = < f_{,\mu} dx^\mu, v^\nu \partial_\nu > = f_{,\mu} v^\nu < dx^\mu, \partial_\nu > = f_{,\mu} v^\nu \delta^\mu_\nu = f_{,\mu} v^\mu$$

The basis vectors of the tangential space ∂_μ are orthogonal to the basis vectors of the cotangential space dx^μ, i.e. the cotangential space $T^*_p(M)$ and the tangential space $T_p(M)$ are *dual*. With $V \in T^*_p(M)$ and $u \in T_p(M)$, then $< V, u > \in \mathbb{R}$ yields a number. Alternatively, one can interpret this operation as the application of a tangential vector on a covector. Thus the tangential space can also be interpreted as the space of linear functions $T^*_p(M) \to \mathbb{R}$, or $T^{**}_p(M) \equiv T_p(M)$ with "$*$" the duality operation.

Covector Chart Transition Covectors transform inversely as compared with the tangential vectors:

$$df = f_\mu dx^\mu = f_{\bar\mu} dx^{\bar\mu} = f_{\bar\mu}(\alpha^{\bar\mu}_\mu dx^\mu) \quad ,$$

thus

$$f_\mu = \alpha^{\bar\mu}_\mu f_{\bar\mu} \quad .$$

[1] Note that the order of expressions is not relevant in index notation, i.e. $v^\mu f_{,\mu} \equiv f_{,\mu} v^\mu$, whereas in matrix notation the order is crucial.

Application Example To support the concept of covectors and coordinate-free formulations, a very practical computational problem will be given here: the ray/surface intersection problem for a spherical height field. This task is to be solved when e.g. some digital elevation model is given for a planetary surface which shall be rendered using the raytracing technique. The problem is stated as follows

1. Given a ray $g(\lambda) = O + \lambda\vec{d}$ with O the ray's starting point, λ the ray's parameter and $\vec{d}$ the ray's direction,
2. and $h(\vartheta, \varphi)$ a height field function on a spherical surface,
3. find the ray parameter λ such that the radial coordinate of $g(\lambda)$ is identical to $h(\vartheta, \varphi)$ for its spherical coordinates ϑ, φ.

Clearly, each point on the ray has unique polar coordinates, which can thus be formulated as functions $r(\lambda), \vartheta(\lambda), \varphi(\lambda)$ of the ray parameter λ. The problem of finding a ray/surface intersection is then formulated as searching the minimal solution $\lambda > 0$ of the equation

$$f(\lambda) := r(\lambda) - h\left(\vartheta(\lambda), \varphi(\lambda)\right) = 0 \quad . \tag{2.8}$$

Since $h(\vartheta, \varphi)$ is an arbitrary function, this equation is best solved by a combination of the Newton method and interval bisection. For doing so, we need the derivative $f'(\lambda)$. From (2.8) we get:

$$f'(\lambda) = \frac{dr}{d\lambda} - \frac{\partial h}{\partial \vartheta}\frac{d\vartheta}{d\lambda} - \frac{\partial h}{\partial \varphi}\frac{d\varphi}{d\lambda} \tag{2.9}$$

We find that $(\frac{dr}{d\lambda}, \frac{d\vartheta}{d\lambda}, \frac{d\varphi}{d\lambda})$ is just the direction of the ray $g(\lambda)$ in polar coordinates. It can be computed from the ray's cartesian direction $\vec{d}$ by multiplication with the appropriate transformation matrix (2.4). Since $f'(\lambda)$ is a number depending linearly on the ray direction, we may interpret it as covector of the form $V = (1, -h_{,\vartheta}, -h_{,\varphi})$ such that $f'(\lambda) \equiv < V, \vec{d} >$ along the ray. As (2.8) can be evaluated at any point p via $F(p) := r(p) - h\left(\vartheta(p), \varphi(p)\right)$, were $r(p), \vartheta(p), \varphi(p)$ are the polar coordinates of the point p, we see that the covector V is just the total derivative $V = dF$ of the "potential function" F, whereby $f(\lambda) = F(g(\lambda))$.

In the Newton method of finding roots of an unknown, but "sufficiently linear" function, we write the potential function F as Taylor series of first order in the ray parameter and expect that an appropriate step $\delta\lambda_{n+1}$ brings us closer to the requested point where $f = 0$:

$$f\left(\lambda_{n+1}\right) = f\left(\lambda_n\right) + \delta\lambda_{n+1} f'\left(\lambda_n\right) \approx 0$$

By assuming that the next point $n+1$ already is the exact intersection point $F = 0$, we get an estimate for the step width (using $f' =< dF, \vec{d} >$):

$$\delta\lambda_{n+1} = -\frac{F\left(g(\lambda_n)\right)}{< dF, \vec{d} >} \quad . \tag{2.10}$$

This operation now needs to be repeated until $|F|$ falls below a given tolerance. Note that eq.(2.10) is written completely coordinate-free, so it can directly be used for height fields given on arbitrary two-dimensional manifolds, as long as coordinate transformations from three-dimensions onto appropriate manifold coordinates and transformation matrices are given. Even more, this method also works for arbitrary curves, not just straight lines in cartesian coordinates, as we may write eq.(2.10) with an arbitrary curve $q(s)$:

$$\boxed{\delta\lambda_{n+1} = -\frac{F\left(q(\lambda_n)\right)}{< dF, \dot{q} >}} \tag{2.11}$$

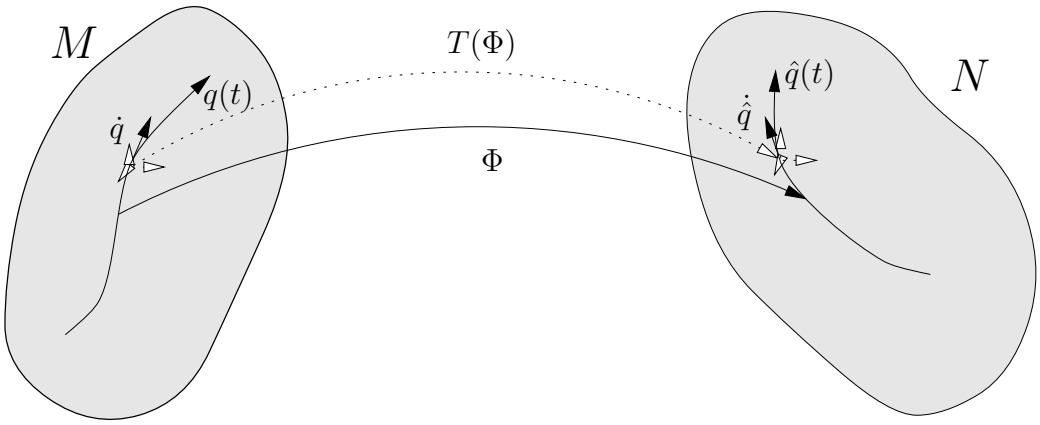
Finally we also show the coordinate expression, where q is given in a chart $\{x^\mu\}$ as the components $q^\mu(\lambda_n)$ with according tangential vector $\dot{q}^\mu(\lambda_n)$ and the potential function F is given in another chart $\{x^{\bar{\mu}}(\{x^\mu\})\}$ with transformation matrix $\alpha^{\bar{\mu}}_{\nu}$ (2.4):

$$\delta\lambda_{n+1} = -\frac{F\big(x^{\bar{\mu}}\left(q^\mu(\lambda_n)\right)\big)}{F_{,\bar{\mu}}\,\alpha^{\bar{\mu}}_{\nu}\dot{q}^\nu} \tag{2.12}$$

Note that transforming the covector dF into cartesian coordinates or the tangential vector $\dot{q}$ into height field coordinates is equivalent, so either can be done, depending on numerical choices and performance considerations.

Tangent map

Let M, N be manifolds and $q(t) : \mathbb{R} \to M$ a curve M. Moreover let $\Phi : M \to N$ be a map from M to N. By virtue of Φ a curve $\hat{q}(t) : \mathbb{R} \to N$ is defined as $\hat{q}(t) := \Phi(q(t))$.



Let $\dot{q}(t) \in T_{q(t)}(M)$ denote the tangential vector on q and $\dot{\hat{q}}(t) \in T_{\hat{q}(t)}(N)$ denote the tangential vector on $\hat{q}(t)$. Using the two charts $\{x^\nu\}$ on M and $\{x^\mu\}$ on N we find a linear relationship between both tangential vectors:

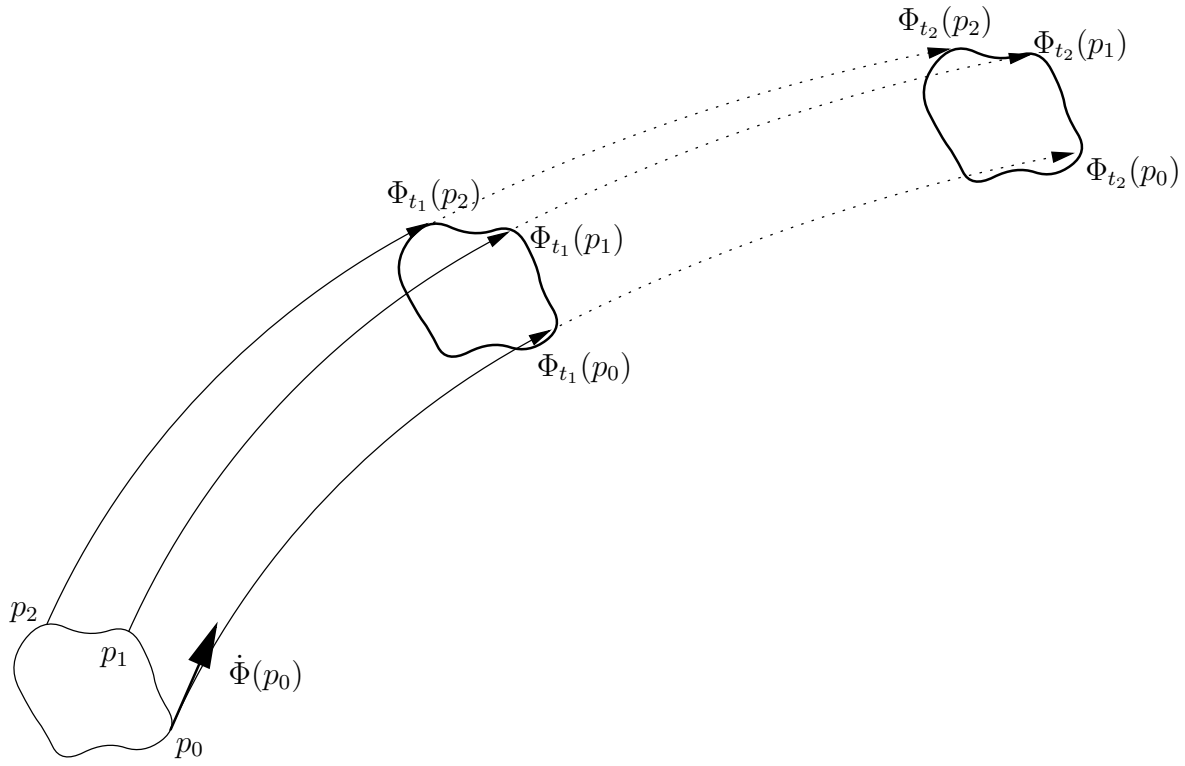
$$\dot{\hat{q}}(t) = \dot{\hat{q}}(t)^\nu \partial_\nu = \frac{d\hat{q}(t)^\nu}{ds}\partial_\nu = \frac{d\Phi^\nu(q^\mu(t))}{ds}\partial_\nu = \frac{\partial \Phi^\nu}{\partial x^\mu}\frac{dq(t)^\mu}{ds}\partial_\nu =: T(\Phi)\left(\dot{q}(t)\right)$$

This map $T(\Phi) : T_{q(t)}(M) \to T_{\hat{q}(t)}(N)$ is called the *tangent map* or differential of the map Φ.

Flow maps

A vector field $v \in \mathcal{T}^1(M)$ defines an array of integral curves Φ by equating the tangential vector on Φ with the vector at each point, i.e. $\dot{\Phi}(p) = v(p)\ \forall p \in M$. This array of integral curves defines the *flow map* $\Phi_t(p) : \mathbb{R} \times M \to M$ which fulfills the properties:

$$\begin{aligned} \Phi_0(p) &= p \\ \Phi_s(\Phi_t(p)) &= \Phi_{s+t}(p) \\ \tfrac{d}{dt}\Phi_t(p)\big|_{t=0} &= \dot{\Phi}_0(p) = v(p) \end{aligned}$$



We might use the short notation $\dot{\Phi} := \dot{\Phi}_0$ in the succeeding text.

Lie-Derivative

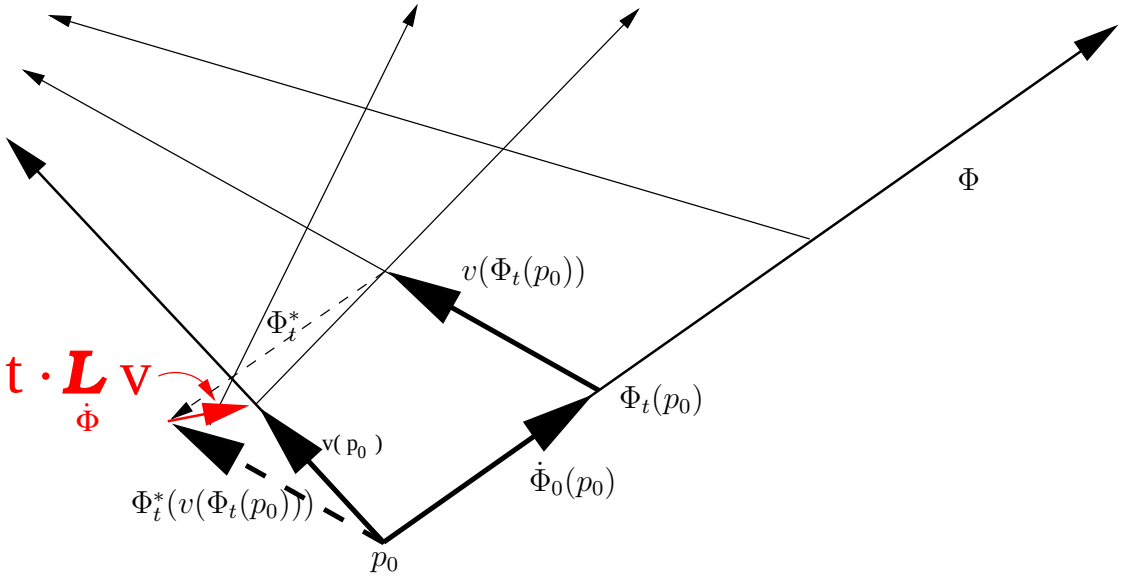
The Lie derivative is a coordinate-dependent operation on vector fields which does not rely on any underlying structure of the manifold. In contrast, the partial derivative of a vector field is not a tensor (i.e. it is not coordinate-independent), and the directional derivative of a vector field (see 2.1.2) involves additional structures, especcially requires knowledge of the manifold's curvature. The Lie derivative is a more fundamental structure and is directly related to the flow map as will be shown now.

Let $u, v \in \mathcal{T}^1(M)$ be vector fields. Then the vector field u defines a flow map $\Phi_t(p)$ with $\dot{\Phi} = u$. The *pullback map* Φ_t^* transports vectors from the tangential space at the transported point $\Phi_t(p_0)$ to the original point p_0, i.e. $\Phi_t^* : T_{\Phi_t(p_0)}(M) \to T_{p_0}(M)$. It is defined via the tangent map:

$$\Phi_t^*(\ v(\Phi_t(p))\ ) := T(\Phi_t^{-1})(\ v(\Phi_t(p))\ )$$

By virtue of the pullback map we may now compare this transported vector with the value of the vector field at the point p_0, which leads to the definition of the *Lie-derivative* as the infinitesimal change of a vector field under the pullback map:

$$\mathfrak{L}_u v := \frac{d}{dt}\Phi_t^*(\ v(\ \Phi_t(p_0)\ )\ )\Big|_{t=0} := \lim_{t\to 0}\left[\frac{1}{t}(\Phi_t^*\left(v(\Phi_t(p_0))\right) - v(p_0))\right]$$



In a chart $\{x^\mu\}$ the Lie derivative becomes:

$$\begin{aligned}\mathfrak{L}_u v &= \frac{d}{dt}\left(\frac{\partial(\Phi_t^{-1})^\mu}{\partial x^\nu} v^\nu(\Phi_t(p))\partial_\mu\right)\Big|_{t=0} \\ &= \underbrace{\frac{\partial(\Phi_t^{-1})^\mu}{\partial x^\nu}\Big|_{t=0}}_{\delta^\mu_\nu} \frac{\partial v^\nu}{\partial x^\sigma} \underbrace{\frac{d(\Phi_t(p_0))^\sigma}{dt}\Big|_{t=0}}_{u^\sigma(\Phi_t(p))|_{t=0}} \partial_\mu + \left[\frac{\partial}{\partial x^\nu}\left(\frac{d(\Phi_t^{-1})^\mu}{dt}\right)\Big|_{t=0}\right] v^\nu(\ \Phi_t(p_0)\ )|_{t=0}\,\partial_\mu = \dots\end{aligned}$$

Whereby we may use that the inverse flow map is given by just inversing the flow parameter, i.e. the inverse flow map is given by simply transporting into the opposite direction:

$$\frac{d(\Phi_t^{-1})(p)}{dt}\Big|_{t=0} = \frac{d(\Phi_{-t})(p)}{dt}\Big|_{t=0} = -\frac{d(\Phi_s)(p)}{ds}\Big|_{\substack{t=0\\ s=-t}} = -\dot{\Phi}(p) = -u(p)$$

And thus we finally find:

$$\begin{aligned}\mathfrak{L}_u v = \dots = \delta^\mu_\nu \frac{\partial v^\nu}{\partial x^\sigma}\ u^\sigma(\Phi_t(p))|_{t=0}\,\partial_\mu \quad + \quad &\left[\frac{\partial}{\partial x^\nu}\left(-u^\mu(p_0)\right)\right] v^\nu(p_0)\partial_\mu \\ &= v^\mu{}_{,\sigma}\, u^\sigma \partial_\mu - u^\mu{}_{,\nu}\, v^\nu \partial_\mu = u(v) - v(u) =: [u, v] \qquad ,\end{aligned}$$

i.e. the Lie derivative is given by the commutator of two vector fields.

Tensor Fields

> Es seien gewisse Dinge ("Tensoren") in Bezug auf jedes Koordinatensystem definiert durch eine Anzahl Raumfunktionen, welche die "Komponenten" des Tensors genannt werden.
>
> Foundation of General Relativity, Albert Einstein, 1916, [Ein16]

Definition 25 *An $m \times n$ **tensor** F is a multilinear mapping from cartesian products of n tangential spaces $T_p(M)$ and m cotangential spaces $T_p^*(M)$ into $\mathbb{R}$ at some point p of a manifold M:*

$$F : (T_p(M))^n \times \left(T_p^*(M)\right)^m \quad \overset{multilinear}{\longrightarrow} \quad \mathbb{R}$$

$n \times m$ is the *rank* of the tensor. The vector space spanned by tensors is called the tensor product space

$$\left(T_p^*(M)\right)^{\otimes n} \otimes (T_p(M))^{\otimes m} := \left\{(T_p(M))^n \times \left(T_p^*(M)\right)^m \quad \overset{\text{multilinear}}{\longrightarrow} \quad \mathbb{R}\right\} \quad , \tag{2.13}$$

whereby $X^{\otimes n}$ denotes the n^{th} tensor product of the space X with itself, similar to the power of a set constructed via the cartesian product from def. 10. Due to the duality relations $T_p^*(M) \leftrightarrow T_p(M)$, the tensor product space can be seen as the space of linear functions that map elements from the dual tensor product space to a number:

$$\left(T_p^*(M)\right)^{\otimes n} \otimes (T_p(M))^{\otimes m} = \left\{(T_p(M))^{\otimes n} \otimes \left(T_p^*(M)\right)^{\otimes m} \quad \overset{\text{linear}}{\longrightarrow} \quad \mathbb{R}\right\} \tag{2.14}$$

With V, U two tensor product spaces, we see

$$V \otimes U \overset{(2.13)}{=} \left\{V^* \times U^* \quad \overset{\text{multilinear}}{\longrightarrow} \quad \mathbb{R}\right\} \overset{(2.14)}{=} \left\{V^* \otimes U^* \quad \overset{\text{linear}}{\longrightarrow} \quad \mathbb{R}\right\}$$

and

$$V^* \otimes U^* \overset{(2.13)}{=} \left\{V \times U \quad \overset{\text{multilinear}}{\longrightarrow} \quad \mathbb{R}\right\} \overset{(2.14)}{=} \left\{V \otimes U \quad \overset{\text{linear}}{\longrightarrow} \quad \mathbb{R}\right\} \quad .$$

It follows from (2.14) that $V^* \otimes U^* \equiv (V \otimes U)^*$. Equivalence relationships of this kind constitute the basis of the Sophus library [MKH95], which is an interesting approach of employing coordinate free mathematics in scientific computing.

Definition 26 *An $m \times n$ **tensor field** $\mathcal{F}$ is a map from each point p of a manifold to an $m \times n$ tensor:*

$$\mathcal{F} : M \longrightarrow \left(T_p^*(M)\right)^{\otimes n} \otimes (T_p(M))^{\otimes m}$$

A tensor field of rank 0×0 is a scalar field $M \to \mathbb{R}$. A tensor field of rank 0×1 is a *vector field* $v : M \to T_p(M)$. A relevant type of tensors are symmetric tensors of rank 2×0, i.e. with $G \in T_p^*(M) \otimes T_p^*(M)$, $v, w \in T_p(M)$ tangential vectors:

$$\begin{array}{rcl} G : T_p(M) \times T_p(M) & \to & \mathbb{R} \\ (v, w) & \mapsto & G(v, w) \end{array} \quad ,$$

i.e. each pair of tangential vectors v, w is mapped into a number at each point in the manifold. A tensor is *symmetric*, if $G(v, w) = G(w, v)$. Symmetric tensors span a vector subspace which is sometimes denoted using the symbol Ⓢ such as $T_p^*(M) \,Ⓢ\, T_p^*(M)$. Elements $U \otimes V$ are written as $u \otimes v$ with $u \in U$ and $v \in V$. For symmetric products, the tensor product symbol is omitted such that $uv \in U Ⓢ V$. This notation is usually only seen for base vectors like $du\,dv$ with du a basis vector of U and dv a basis vector of V, otherwise it might be confused with the dot product of two vectors. Given a tensor product $u \otimes v$ of two tensors u, v, the symmetric component can be constructed easily as $uv \equiv u Ⓢ v = \frac{1}{2}(u \otimes v + v \otimes u)$.

For a fixed vector field v, a 2×0 tensor G defines a mapping $T_p(M) \to \mathbb{R}$:

$$\begin{array}{rcl} G(v, -) : T_p(M) & \to & \mathbb{R} \\ w & \mapsto & G(v, w) \end{array} \tag{2.15}$$

This way we have defined tensor fields as a map which assigns to each point of a manifold an object that carries vector space properties. Alternatively, we can omit the notion of objects living in the tangential space and directly define a tensor field as functions on a manifold which possess certain vector space properties:

Definition 27 *A* **vector field** v *on a manifold* M *is a function from* $C^\infty(M)$ *to* $C^\infty(M)$ *satisfying the following properties:*

$$\begin{aligned} v(f+g) &= v(f)+v(g) && (2.16)\\ v(a\cdot f) &= a\cdot v(f) && (2.17)\\ v(f\cdot g) &= v(f)\cdot g + f\cdot v(g) \quad \textit{(Leibniz rule)} && (2.18)\end{aligned}$$

for all $f, g \in C^\infty(M)$ *and* $a \in \mathbb{R}$ *[BM94].*

To see that this definition is equivalent with the previous ones, we may think of a vector field as the tangents of a set of curves $\{q(s) : \mathbb{R} \to M\}$ that cover each point of the manifold. We now inspect the properties of the derivatives of functions $f, g \in C^\infty(M)$ by the curve parameters to find:

$$\begin{aligned} \frac{d}{ds}\Big(f\big(q(s)\big)+g\big(q(s)\big)\Big) &= \frac{d}{ds}(f)+\frac{d}{ds}(g) && (2.19)\\ \frac{d}{ds}(a\cdot f) &= a\cdot\frac{d}{ds}(f) && (2.20)\\ \frac{d}{ds}(f\cdot g) &= \left(\frac{d}{ds}f\right)\cdot g + f\cdot\left(\frac{d}{ds}g\right) && (2.21)\end{aligned}$$

Clearly, the operation $\frac{d}{ds}$ possesses the same properties as the vector field definition above, but def. 27 does not depend on the definition or existence of curves (e.g. integral curves are not necessarily existent or contiguous on the entire manifold), but completely covers the entire manifold.

Chart Transition An $n \times m$ tensor is written in a chart $\{x^\mu\}$ as

$$T = T^{\nu_1,\nu_2,\ldots,\nu_m}_{\mu_1,\mu_2,\ldots,\mu_n} \quad \partial_{\mu_1}\otimes\partial_{\mu_2}\otimes\cdots\otimes\partial_{\mu_n} \quad \otimes \quad dx^{\nu_1}\otimes dx^{\nu_2}\otimes\cdots\otimes dx^{\nu_m} \quad .$$

The $T^{\nu_1,\nu_2,\ldots,\nu_m}_{\mu_1,\mu_2,\ldots,\mu_n}$ are the components of the tensor T in the chart $\{x^\mu\}$. The chart transition rules can be directly read off from the transition rules of the basis vectors and is just the multiplication of each tensor component with an appropriate chart transition matrix (2.4) such that the indices cancel in the resulting expression:

$$T^{\bar\nu_1,\bar\nu_2,\ldots,\bar\nu_m}_{\bar\mu_1,\bar\mu_2,\ldots,\bar\mu_n} = T^{\nu_1,\nu_2,\ldots,\nu_m}_{\mu_1,\mu_2,\ldots,\mu_n} \quad \alpha^{\mu_1}_{\bar\mu_1}\alpha^{\mu_2}_{\bar\mu_2}\ldots\alpha^{\mu_n}_{\bar\mu_n} \quad \alpha^{\bar\nu_1}_{\nu_1}\alpha^{\bar\nu_2}_{\nu_2}\ldots\alpha^{\bar\nu_m}_{\nu_m} \tag{2.22}$$

Often the definition of tensors and tensor fields starts with the chart transition rule and it is said, that every object that obeys the chart transition rule (2.22) is a tensor. This approach of defining tensors hides the view to the underlying idea of coordinate-free formulation because it starts with the view of tensor fields in certain (although arbitrary) charts. However, charts are not required when speaking about tensor fields. Many properties can be discussed without looking at the coordinate expression at all. Dealing with coordinate expressions is only required when actually performing numerical computations, but when dealing with mathematical equations coordinate expressions contain the danger of becoming coordinate-dependent.

Metric Tensor Field

In def. 11 we made use of a function that maps two elements of a topological space to a number in a certain way. Now considering elements of the tangential space, i.e. tangential vectors, we may use a linear function to provide a clue of the distance between two tangential vectors; such a function conforms to the definition of a tensor field:

Definition 28 *The* ***metric tensor field*** g *is a symmetric tensor field*

$$g : T_p(M)\times T_p(M) \to \mathbb{R} \quad .$$

It maps two tangential vectors to a number (symmetric bilinear form).

Metric tensor fields possess certain coordinate-invariante properties which determine the properties of the underlying manifold:

Definition 29 *Let* λ_i *be the signs of the eigenvalues of a metric tensor. The difference of the number of positive and negative components is the* ***signature*** *of the metric tensor. It is written as an (ordered)* $n-$*tuple of signs, eg.* $(+,+,-,-)$.

A metric tensor field has constant signature for the entire manifold. The signature does not depend on the choice of coordinates, the required eigenvalues can be computed in any chart.

In special and general relativity, a pseudo-Riemannian metric tensor field of signature $(+,-,-,-)$ is used (the alternative, but equivalent convention $(-,+,+,+)$ can also be found in literature sometimes). Note that a pseudo-Riemannian metric is no longer a metric in the topological sense of def. 11 and cannot be used to construct metric spaces, as discussed in 2.1.1. As a distance function on elements of the tangential space, it allows to distinguish tangential vectors into three categories:

- $g(v,v) > 0 \quad \longleftrightarrow \quad v$ "timelike"
- $g(v,v) = 0 \quad \longleftrightarrow \quad v$ "lightlike"
- $g(v,v) < 0 \quad \longleftrightarrow \quad v$ "spacelike"

The metric tensor allows to define the length of a timelike tangential vector (which has a physical meaning of time)

$$|v| := \sqrt{g(v,v)} \tag{2.23}$$

as well as the length of a spacelike tangential vector:

$$|v| := \sqrt{-g(v,v)} \tag{2.24}$$

and the angle between a pair of spacelike tangential vectors:

$$\cos\alpha(u,v) := \frac{-g(u,v)}{|u|\ |v|} = \frac{-g(u,v)}{\sqrt{g(u,u)\ g(v,v)}} \tag{2.25}$$

or even timelike tangential vectors

$$\cosh\alpha(u,v) := \frac{g(u,v)}{|u|\ |v|} = \frac{g(u,v)}{\sqrt{g(u,u)\ g(v,v)}} \quad . \tag{2.26}$$

The angle of lightlike tangential vectors is undefined; because the metric length $g(v,v)$ of a lightlike tangential vector $v \neq \vec{0}$ is null, it can be said that a lightlike tangential vector is "orthogonal to itself" at the same time as it is "tangential to itself", such that angular relationships of lightlike vectors do not make much sense at all. It would only make sense to compare the angles of their spatial projections, i.e. the directions of infalling light rays – which is then coordinate-dependent, because a spatial projection can only be done using a certain chart.

Definition 30 *A line is **timelike** if its tangent vectors are timelike at any point of the curve:*

$$q \quad timelike \quad \Longleftrightarrow \quad g\left(\dot{q}(s), \dot{q}(s)\right) > 0 \quad \forall s$$

Such a line is also called a worldline. Analogously a spacelike line can be defined. However, spacelike lines are not as relevant in physics as timelike lines.

Definition 31 *The metric tensor defines the **length** of a curve (or line) q:*

$$l_{12} = \int_{s_1}^{s_2} \sqrt{\pm g(\dot{q}(s), \dot{q}(s))} ds$$

Definition 32 *The length of a timelike curve is called the proper time, the length of a spacelike curve is the proper distance.*

The length of a lightlike curve is always zero. The length of a curve which is partially timelike, spacelike and/or lightlike has no physical meaning.

By the means of timelike vectors the four-velocity is defined for a pseudo-Riemannian manifold:

Definition 33 *The **four-velocity** of a wordline is defined as:*

$$u := \frac{\dot{q}}{\sqrt{g(\dot{q},\dot{q})}}$$

The norm of the four-velocity $|u| = \sqrt{g(u,u)}$ is always 1, as can be seen easily. It is a timelike vector.

Coordinate Expression and Chart Transition In a chart $\{x^\mu\}$ we write tangential vectors as linear combinations of their basis vectors and make use of the linearity properties of the metric tensor field to derive a coordinate expression:

$$g(u,v) = g(u^\mu \partial_\mu, v^\nu \partial_\nu) = u^\mu v^\nu g(\partial_\mu, \partial_\nu) =: u^\mu \cdot v^\nu \cdot g_{\mu\nu}$$

$g_{\mu\nu}$ are called the *components* of the metric tensor g in the chart $\{x^\mu\}$. Using basis vectors of $T_p^*(M) \times T_p^*(M)$ it is also often written as the advance of a small distance ds:

$$ds^2 = g_{\mu\nu} dx^\mu dx^\nu$$

The matrix

$$(g_{\mu\nu}) = \begin{pmatrix} g_{00} & g_{01} & \cdots & g_{0n} \\ g_{10} & g_{11} & \cdots & g_{1n} \\ \vdots & \vdots & & \vdots \\ g_{n0} & g_{n1} & \cdots & g_{nn} \end{pmatrix}$$

is the *representation* of the metric tensor G in the chart $\{x^\mu\}$. Due to the symmetry of g, it follows for the components that $g_{\mu\nu} = g_{\nu\mu}$ and only $\frac{n(n+1)}{2}$ of the n^2 components in an n-dimensional space are independent.

As a special case of (2.22), the chart transition of the metric tensor's components becomes:

$$g_{\bar{\mu}\bar{\nu}} = g_{\mu\nu} \alpha^\mu_{\bar{\mu}} \alpha^\nu_{\bar{\nu}}$$

EXAMPLE: The Euclidean metric tensor η is a unit matrix $\eta_{\mu\nu} = \delta_{\mu\nu}$ in a Euclidean chart, with δ the Kronecker symbol. Using the transformation matrices for transition between cartesian and polar coordinates (see p.16) and a little algebra we find that the Euclidean metric tensor in a polar chart $\{r, \vartheta, \varphi\}$ becomes

$$(g_{\mu\nu}) = \begin{pmatrix} 1 & & \\ & r^2 & \\ & & r^2 \sin^2 \vartheta \end{pmatrix} \tag{2.27}$$

or

$$ds^2 = dr^2 + r^2 d\vartheta^2 + r^2 \sin^2 \vartheta d\varphi^2 \quad .$$

EXAMPLE: Given a spatial velocity vector $\vec{v} = v^a \vec{\partial}_a$ with $a = 1, 2, 3$ in a Minkowskian chart with metric tensor $\eta = dt^2 - dx^2 - dy^2 - dz^2$, the corresponding four-velocity is given by

$$u = \frac{(1, \vec{v})}{\sqrt{1 - |\vec{v}|^2}} = \gamma(1, \vec{v}) \qquad \text{with } \gamma := \frac{1}{\sqrt{1 - |\vec{v}|^2}} \geq 0 \quad .$$

We see easily that $\eta(u,u) = (u^0)^2 - |\vec{u}|^2 = \gamma^2(1 - |\vec{v}|^2) \equiv 1$ and $|\vec{v}| < 1$ is required. For $\vec{v} = 0$ the 4-velocity is $(1,0,0,0)$. The length of a timelike 4-vector is coordinate invariant. The time component $u^0 = \gamma$ of worldlines with $|\vec{v}| > 0$ will appear scaled by a factor γ as compared to worldlines which correspond to particles at rest relative to some observer, which is known as the Lorentz time dilation of moving observers.

Co-Metric If a metric $(g_{\mu\nu})$ can be inverted (not degenerated metric), then a co-metric $(g^{\nu\lambda})$ can be defined on the covectors:

$$g_{\mu\nu} g^{\nu\lambda} = \delta^\lambda_\mu$$

EXAMPLE: The Euclidean co-metric in polar coordinates becomes:

$$(g^{\mu\nu}) = \begin{pmatrix} 1 & & \\ & \frac{1}{r^2} & \\ & & \frac{1}{r^2 \sin^2 \vartheta} \end{pmatrix}$$

REMARK: In Newtonian spacetime (J.Rothleitner [Rot99] has presented an elaborated formulation in modern differential geometry) the co-metric has the signature $(0,+,+,+)$, thus it is degenerated and a (four-dimensional) metric does not exist. The explicit specification of an exact 1-form Z, called the "time-one-form", in addition to the spacetime co-metric is required to allow the computation of the length of vectors. The term "timelike" or "spacelike" is defined via Z:

- $v \in T_p(M)$ spacelike $\iff \quad < Z, v >= 0$

- $v \in T_p(M)$ timelike $\Longleftrightarrow \quad < Z, v > \neq 0$

Z defines slices in the spacetime which are connected by spacelike vectors, and any function $t : M \longmapsto \mathbb{R}$ with $Z = dt$ may be used as a coordinate function. Such a coordinate function is called the "absolute time", which is uniquely defined by Z except for a constant t_0 (if t is an absolute time with $Z = dt$, then also $\bar{t} + t_0$ is an absolute time coordinate because $Z = d\bar{t} = dt$). The time-one-form describes the concept of an "*ether*", consisting of an absolute time which is identical for all points in space. The four-velocity, see def. 33, in a Newtonian spacetime is defined as $u = \dot{q}/ < Z, \dot{q} >$.

Musical Isomorphisms A metric allows to define isomorphic mappings between tangential and cotangential spaces. This is required for identifying vector and covectors, like it is done in Euclidean space or to formulate operations on cotangential spaces in tangential spaces and vice versa. These mappings appear as "index lowering" and "index raising" when viewing these operations at the coordinate components. With respect to the lowering and raising operators of halftones, these maps are called "musical":

Definition 34 *The index lowering* $\flat : T_p(M) \to T_p^*(M)$ *is defined via the identity*

$$< V, u >= g(\flat v, u)$$

In a chart $\{x^\mu\}$:

$$V_\mu = g_{\mu\nu} v^\nu =: (\flat v)_\mu$$

Definition 35 *The index raising* $\sharp : T_p^*(M) \to T_p(M)$ *is defined via the identity*

$$< V, u >= g(V, U)$$

In a chart $\{x^\mu\}$:

$$v^\mu = g^{\mu\nu} V_\nu =: (\sharp V)^\mu$$

Remark: In Euclidean space the metric is represented by the unit matrix, thus the components of a covector and a vector appear numerically identical under the musical isomorphisms and covectors are not distinguished from vectors in Euclidean geometry.

Application example: By the means of the musical isomorphism, the well-known operation of the gradient (a tangential vector) of a scalar function is defined, making use of (2.7) and def. 35:

$$\vec{\nabla} f := \sharp df \tag{2.28}$$

The coordinate expression $\sharp df = g^{\mu\nu} f_{,\nu} \vec{\partial}_\mu$ can be read off easily.

2.1.2 Velocity and Acceleration

Concept of Time

This section contains non-standard concepts which have been inspired by P. Wagner [Wag94].

In Newtonian mechanics, time is an absolute quantity and is like a constant flow at each point of the universe, uniquely defined for each event. All events that happen at the same time constitute the "present". Events which happen at a later time, form the "future", earlier events the "past". The notion of "past", "future" and "present" is uniquely defined via the time difference of two events.

In relativistic physics, time is a relative quantity which depends on the motion (special relativity) and/or the location (general relativity). Various coordinate systems with different notions of time may co-exist. The time difference in one coordinate system (defining a "future" in this coordinate system) is not sufficient to determine their causal connectivity. What resides in a future relative to one coordinate system, might be in the past relative to another coordinate system. An "absolute" notion of the term "future" thus requires a definition which is not based on time coordinates, but on causality among events:

Definition 36 *The set of all points of a Riemannian manifold* M *which can be reached via a timelike line(def.30) from a point* p *is called the (absolute)* ***non-present*** *of* p*, denoted as* $N(p)$*:*

$$N(p) = \{x \in M | \exists q : [0,1] \longrightarrow M, q(0) = p, q(1) = x, \forall s \in [0,1] : g(\dot{q}(s), \dot{q}(s)) > 0\}$$

To distinguish between the terms "future" and "past" we need an additional structure on the given space beyond the metric. We call this the "time orientation", denoted by $\uparrow$ or $\downarrow$. Since the metric has signature $(+,-,-,-)$ the double cone $\{v \in T_p(M) : g(v,v) > 0\}$ consists of two connected components $V_1, V_2 \subset M$.

Definition 37 *M is called **time-orientable** if we can chose one of the half-cones V_1, V_2 as the "future cone" in a continuous way, i.e. if we can chose a continuous map*

$$\uparrow: \{v : v \in T_p(M), g(v,v) > 0\} \longrightarrow \{-1, +1\}$$

such that $\uparrow(-v) = -\uparrow v$. We set $\downarrow v = -\uparrow v$. For lightlike non-null vectors w we define $\uparrow w$ by continuity. For spacelike vectors the time orientation is undefined.

The notion of "time-orientation" is usually neglected when speaking about manifolds, but it is nevertheless an important property of a physically acceptable spacetime. O'Neill [O'N83], p. 145, defines time-orientability in a similar way. A lemma proven there tells that a manifold M is time-orientable if and only if there exists a timelike vector field on M.

The choice of a time orientation $\uparrow$ can be interpreted as the choice of the positive or negative square root when computing the length of a vector from its given square $g(v,v)$. Choosing a negative value as length is mathematically equivalent, but by convention the positive value is taken. The time orientation operator makes such an implicit choice explicit. It allows the distinction among the (mathematically equivalent) future and past:

Definition 38 *Let M be a time oriented manifold. A timelike or non-null lightlike vector is called **future(past)-directed** iff its time orientation is positive(negative):*

$$v \quad \textit{future-directed} \quad \Longleftrightarrow \quad v \in T_p(M), g(v,v) \geq 0, \uparrow v > 0$$

$$v \quad \textit{past-directed} \quad \Longleftrightarrow \quad v \in T_p(M), g(v,v) \geq 0, \downarrow v > 0$$

Definition 39 *A metric space (def.11) together with a time orientation is a **physical space** $(M, G, \uparrow)$ or **spacetime**. The points within a physical space are called **events**.*

Exchanging the time orientation operator maps the future to the past and vice versa. At each point of Minkowski spacetime actually two sheets of spacetime co-exist like parallel universes with respectively reversed time orientation. The double sheets of flat Minkowski spacetime never touch. However, the corresponding sheets of a curved spacetime may touch, e.g. at the event horizon of a static black hole: here a negatively time oriented asymptotically flat spacetime (approaching Minkowski spacetime far from the black hole) touches with a positively time orientated asymptotically flat spacetime. Both regions are outside the event horizon. The opposite direction of time becomes obvious when computing the flow of time in Schwarzschild coordinates in the "parallel universe". The Kruskal coordinate diagram (fig. 2.2) visualizes the reversion of the Schwarzschild time coordinate quite clear: consecutive time slices are straight lines through the center of the coordinate origin with various inclination angles, whereby 45^o corresponds to the event horizon). Increasing Schwarzschild time in our universe (region I, right of the Kruskal coordinate origin) corresponds to decreasing Schwarzschild time in the parallel universe (region I', left of the Kruskal coordinate origin). Within the event horizon (region II), both directions coincide, because time is "inward-directed" there. The time reversal is given by the white hole solution (region II''). The inner region of the white and black hole (white: things may only come out, black: things may only fall in) are isometric (lines and curves sustain there length under isometry operations) in a metric space, but not in a physical space (i.e. taking into account the time orientation as well, future-directed curves become past-directed curves and vice versa).

Interesting things may happen in exotic spacetimes like the very inner regions of a rotating black hole (the so-called *Kerr* solution (2.53) of Einstein's equations): A rotating black hole has two event horizons. In the innermost region behind the inner event horizon, the "rotational pull" of the spacetime is such that the tangential base vector for the azimuthal coordinate, ∂_φ becomes timelike, i.e.

$$g(\partial_\varphi, \partial_\varphi) = g_{\varphi\varphi} > 0$$

Now, as φ is the azimuthal angle with the periodic property that $\varphi + 2\pi \equiv \varphi$, this means that after a period of proper length 2π (which is a *time* here, see def.32) along a (timelike!) world line $q(s) = \varphi(s)$ an astronaut would arrive at exactly the same event in the manifold as where he started! Such a world line is called a *closed timelike line* (CTL). If they exist in nature, they would allow time travel. However, it is not yet known if CTL's can exist outside of event horizons or if there is always an event horizon separating "normal" space from regions containing CTL's (similar to the hypothesis of "cosmic censorship", which supposes that any singularity is surrounded by an event horizon and is thus invisible to the outside).

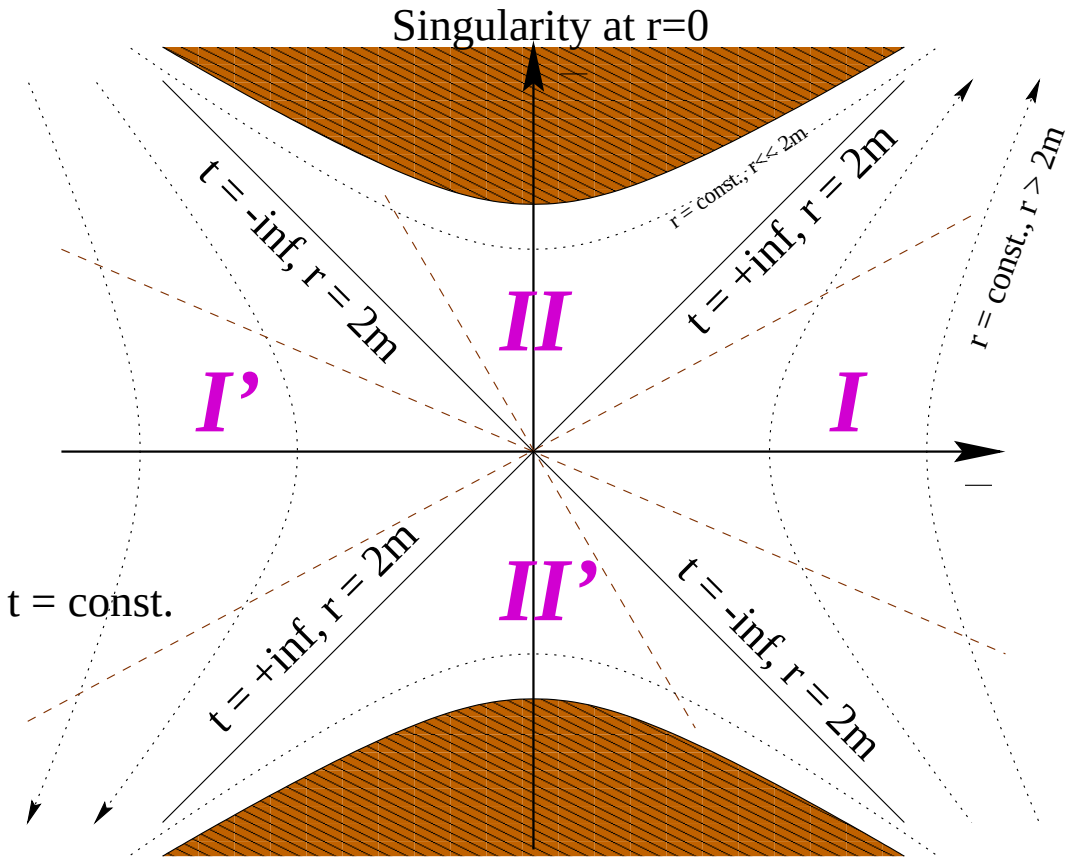


Figure 2.2: Geodesically complete Schwarzschild spacetime in Kruskal coordinates. Note the reversal of the Schwarzschild time on the left side, i.e. $t < 0$ close to the black hole in the parallel universe I'. [Ben97] contains a detailed explanation with derivation.

Geodesics

Definition 40 *An **extremal line** is the shortest or longest (i.e. most extreme) connection between two points. It is determined by the metric.*

A curve $q(s)$ is the most extreme connection between two points $A = q(s_1)$ and $B = q(s_2)$ iff

$$\int_{s_1}^{s_2} \sqrt{|g(\dot{q}(s), \dot{q}(s))|}\, ds = \text{minimum}$$

This minimization principle is similar to the Lagrange formalism within mechanics. Here $q(s)$ is the true path of a particle iff

$$\int_{s_1}^{s_2} \mathfrak{L}(q^k(s), \dot{q}^k(s))\, ds \stackrel{!}{=} \text{extr.}$$

From

$$\delta \int \mathfrak{L}(q^k(s), \dot{q}^k(s))\, ds = 0$$

we get the well-known Euler-Lagrange-equations:

$$\frac{\partial \mathfrak{L}}{\partial q^k} - \frac{d}{ds}\frac{\partial \mathfrak{L}}{\partial \dot{q}^k} = 0 \tag{2.29}$$

Now if we take the length of a curve's tangential vector as the Lagrange function:

$$\mathfrak{L}(q^k(s), \dot{q}^k(s)) = \sqrt{g(\dot{q}(s), \dot{q}(s))} = \sqrt{\dot{q}^\mu \dot{q}^\nu g_{\mu\nu}}$$

then its solutions are extremal lines. They are independent from the parameterization of the curves (proof not shown here). If we take the square of the length of the tangential vectors as Lagrange function:

$$\mathfrak{L}(q^k(s), \dot{q}^k(s)) = g(\dot{q}(s), \dot{q}(s)) = \dot{q}^\mu \dot{q}^\nu g_{\mu\nu} \tag{2.30}$$

then the parameterization of the curve becomes fixed and the solutions are extreme lines which are parameterized by their affine parameters.

Definition 41 *A **geodesic** curve is an extremal line with its affine parameter.*

Geodesics are determined from a metric tensor field via the geodesic equation (2.35).

Deriving the Geodesic Equation The first term of Euler-Lagrange equations (2.29) with the lagrange function (2.30) becomes:

$$\frac{\partial \mathfrak{L}}{\partial q^\alpha} = \dot{q}^\mu \dot{q}^\nu \frac{\partial g_{\mu\nu}}{\partial x^\alpha} = \dot{q}^\mu \dot{q}^\nu g_{\mu\nu,\alpha} \tag{2.31}$$

Derivation of (2.30) by the velocity $\dot{q}^\alpha$ of the curve yields:

$$\frac{\partial \mathfrak{L}}{\partial \dot{q}^\alpha} = \frac{\partial \left(\dot{q}^\mu \dot{q}^\nu g_{\mu\nu}\right)}{\partial \dot{q}^\alpha} = \dot{q}^\nu g_{\mu\nu} \delta^\mu_\alpha + \dot{q}^\mu g_{\mu\nu} \delta^\nu_\alpha = \dot{q}^\nu g_{\alpha\nu} + \dot{q}^\mu g_{\mu\alpha} = 2 g_{\mu\alpha} \dot{q}^\mu \tag{2.32}$$

Derivation by the curve parameter leads to the second term of (2.29):

$$\begin{aligned} \frac{d}{ds}\frac{\partial \mathfrak{L}}{\partial \dot{q}^\alpha} \underset{(2.32)}{=} \frac{d}{ds}\left(2 g_{\mu\alpha} \dot{q}^\mu\right) = 2\left(\frac{d}{ds} g_{\mu\alpha}\right)\dot{q}^\mu + 2 g_{\mu\alpha}\frac{d}{ds}\dot{q}^\mu \underset{(2.3)}{=} \\ 2\left(\frac{\partial g_{\mu\alpha}}{\partial x^\nu}\frac{d}{ds} q^\nu\right)\dot{q}^\mu + 2 g_{\mu\alpha}\ddot{q}^\mu = 2\left(g_{\mu\alpha,\nu}\, \dot{q}^\nu \dot{q}^\mu + g_{\mu\alpha}\ddot{q}^\mu\right) \end{aligned} \tag{2.33}$$

From this we may read off the equations of geodesics:

$$0 = \frac{\partial \mathfrak{L}}{\partial q^\alpha} - \frac{d}{ds}\frac{\partial \mathfrak{L}}{\partial \dot{q}^\alpha} = \dot{q}^\mu \dot{q}^\nu g_{\mu\nu,\alpha} - 2\left(g_{\mu\alpha,\nu}\, \dot{q}^\nu \dot{q}^\mu + g_{\mu\alpha}\ddot{q}^\mu\right) = -2 g_{\mu\alpha}\ddot{q}^\mu - 2\left(g_{\mu\alpha,\nu} - \frac{1}{2} g_{\mu\nu,\alpha}\right)\dot{q}^\nu \dot{q}^\mu$$

Multiplying by $-\frac{1}{2} g^{\alpha\lambda}$ reveals:

$$\ddot{q}^\lambda + \underbrace{\frac{1}{2} g^{\alpha\lambda}\left(2 g_{\mu\alpha,\nu} - g_{\mu\nu,\alpha}\right)}_{=: A^\lambda_{\mu\nu}} \dot{q}^\nu \dot{q}^\mu = 0$$

From the abbreviation expression $A^\lambda_{\mu\nu}$ only the symmetric part contributes:

$$-\ddot{q}^\lambda = A^\lambda_{\mu\nu} \dot{q}^\nu \dot{q}^\mu = A^\lambda_{\nu\mu} \dot{q}^\mu \dot{q}^\nu = A^\lambda_{\nu\mu} \dot{q}^\nu \dot{q}^\mu = \underbrace{\frac{1}{2}\left(A^\lambda_{\mu\nu} + A^\lambda_{\nu\mu}\right)}_{=: \Gamma^\lambda_{\mu\nu}} \dot{q}^\nu \dot{q}^\mu$$

These are the so-called **Christoffel symbols** $\Gamma^\lambda_{\mu\nu}$. They only depend on the metric and its first partial derivatives:

$$\Gamma^\lambda_{\mu\nu} := \frac{1}{2} g^{\lambda\alpha}\left(\frac{1}{2}\left(2 g_{\mu\alpha,\nu} - g_{\mu\nu,\alpha}\right) + \frac{1}{2}\left(2 g_{\nu\alpha,\mu} - g_{\nu\mu,\alpha}\right)\right) = \frac{1}{2} g^{\lambda\alpha}\left(g_{\mu\alpha,\nu} + g_{\nu\alpha,\mu} - g_{\mu\nu,\alpha}\right) \tag{2.34}$$

The abbreviated form of the geodesic equation in a chart $\{x^\mu\}$ thus can be written as:

$$\ddot{q}^\lambda + \Gamma^\lambda_{\mu\nu} \dot{q}^\mu \dot{q}^\nu = 0 \tag{2.35}$$

Definition 42 *A manifold is **geodesically complete** if it contains all points which can be reached with a finite parameter by an arbitrary geodesic starting within the manifold.*

In general, it is not possible to cover a geodesically complete manifold with a single chart. E.g. the outer region of a black hole is not geodesically complete. It is complete regarding spacelike geodesics (their proper length becomes infinite when touching the event horizon), but not regarding lightlike or timelike geodesics (which stay finite at the event horizon - an astronaut may cross the event horizon in a finite amount of proper time, although the Schwarzschild chart is not able to cover the event horizon itself and a static rescue rope down to the event horizon would have to be infinitely long). The domain of the so-called Eddington-Finkelstein chart goes beyond the event horizon and covers the interior of a black hole as well as the outside, but still is not geodesically complete. It does not tell where geodesics originate which do not come from spatial infinity $r = \infty$, and they clearly cannot come from the black hole itself. The geodesically complete extension of the Schwarzschild manifold is only visible in the so-called Kruskal-Skeresz coordinates; it covers the black hole solution together with the white hole solution and the parallel universe together with the "home" universe. All these four regions touch at the event horizon.

The lucky case of finding a single chart for a geodesically complete manifold is not common. E.g. in the case of the rotating black hole, it is not possible to find a single chart for the geodesically complete manifold, it can only be covered by multiple charts. For instance, here the singularity at $r = 0$ is actually a ring, and there exists a parallel universe at $r < 0$. There is no way to visualize a region $r < 0$ in usual polar coordinates.

Chart Transition Given the Christoffel symbols in a chart $\{x^\mu\}$ their representation in another chart $\{x^{\bar\mu}\}$ is given by

$$\Gamma^{\bar\sigma}_{\bar\mu\bar\nu} = \Gamma^{\sigma}_{\mu\nu}\alpha^{\bar\sigma}_{\sigma}\alpha^{\mu}_{\bar\mu}\alpha^{\nu}_{\bar\nu} + \alpha^{\kappa}_{\bar\mu,\bar\nu}\,\alpha^{\bar\sigma}_{\kappa} \quad , \tag{2.36}$$

where α are the chart transition matrices as in def. 2.4. The first term in the sum is called the *tensorial part*, whereas the existence of the second one makes clear that the Christoffel symbols are not components of a tensor (the generalization of tensors are called *geometrical objects*). For instance, all of the Christoffel symbols may be zero in one chart, $\Gamma^{\sigma}_{\mu\nu} = 0$, but non-zero in another chart $\Gamma^{\bar\sigma}_{\bar\mu\bar\nu} \neq 0$. For tensor fields, this cannot happen. If a tensor field is zero in all of its components in one chart, it is zero in all other charts as well. This property of the Christoffel symbol leads to the following definition:

Definition 43 *A chart $\{x^\mu\}$ is said to be **inertial**, if the Christoffel symbols vanish in this chart.*

General relativity states that inertial coordinate systems do not necessarily exist. However, at any event in a spacetime a coordinate system can be found locally such that the Christoffel symbols vanish there. Such a coordinate system is called *locally Minkowskian* and corresponds to a freely falling observer. An infinite number of locally Minkowskian coordinate systems exists, which are all related via the Lorentz transformation (freely falling observers may have arbitrary velocity less than the speed of light and move in any direction). The coordinates of a locally Minkowskian chart are called *Riemannian normal coordinates*. Finding Riemannian normal coordinates is identical to diagonalizing the metric at a particular event. Having a locally Minkowskian chart means to find a chart such that the first derivatives of the metric vanish in this chart; this can always be achieved by a proper choice of coordinates. However, the second derivatives of the metric cannot be made zero by coordinate transformations. Thus the coordinate system of a freely falling observer will be inertial only in first order, but he will still experience effects due to the second order derivatives of the metric, which are actually tidal forces of the gravitational field which cannot be transformed away by proper motion.

Another consequence is that the energy of the gravitational field, which shows up in the second derivatives of the metric, is not localizable, because any gravitational field appears locally Minkowskian. The energy of a gravitational field can only be defined via integration over some finite volume, but it is not possible to define some energy density of the gravitational field like in hydrodynamics.

Note that while the Christoffel objects itself are not tensors, the difference of any two Christoffel objects yields a tensor, because the non-tensorial parts in the chart transition rules (2.36) cancel. This is important for the definition of the curvature tensors like the Riemann, Ricci and Weyl tensors, which are constructed from derivatives and differences of Christoffel objects.

Straight Lines and Connections

The partial derivative of a tensor field is no longer a tensor field. Actually it is not even trivial to define what the derivation of a tensor field means, because derivation requires to compare elements from different tangential spaces. We may try to define the derivation of a tensor field V (along a curve q) like we defined the derivation along a curve, def. 2.2, which resulted in the definition of tangential vectors

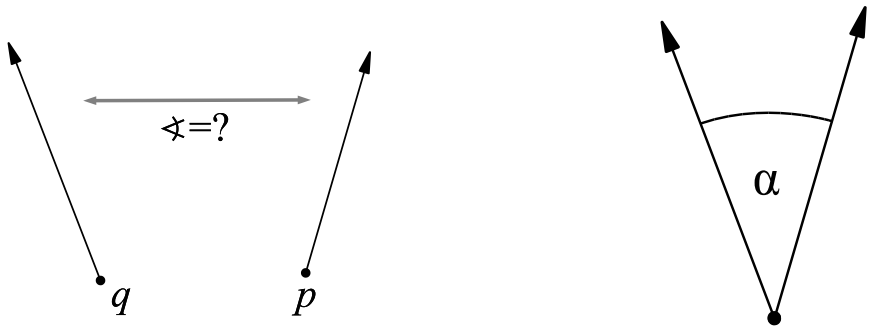


Figure 2.3: What is the "distance" between two (tangential) vectors at different points p and q? We can only compare vectors (which can be seen as "directions with speed information") if they originate at the same point in space (the same point p of a manifold M). Only then (i.e. if both reside in the same tangential space $T_p(M)$), we can make use of def. 28 and define the distance as the "dot product" of the two vectors.

def. 21:

$$\partial_q V \equiv \frac{dV}{ds} := \lim_{h\to 0} \frac{V(q(s+h)) - V(q(s))}{h}$$

At this point, we have a problem with the pretendedly simple subtraction operation (see fig. 2.3 for illustration): namely – let us limit ourselves to vector fields for now – $V(q(s+h)) \in T_{q(s+h)(M)}$ and $V(q(s)) \in T_{q(s)(M)}$ are elements from different spaces and therefore can not be simply compared or subtracted! Of course, in a given chart $\{x^\mu\}$ we may just determine the numerical values of the vector components and compute the numerical difference, but the resulting number (the *partial derivative*) then depends on the choice of the chart. Using another chart $\{x^{\bar{\mu}}\}$ would give another result. Thus we first need an operation which transports objects from $T_{q(s+h)(M)}$ into $T_{q(s)(M)}$ (fig. 2.4):

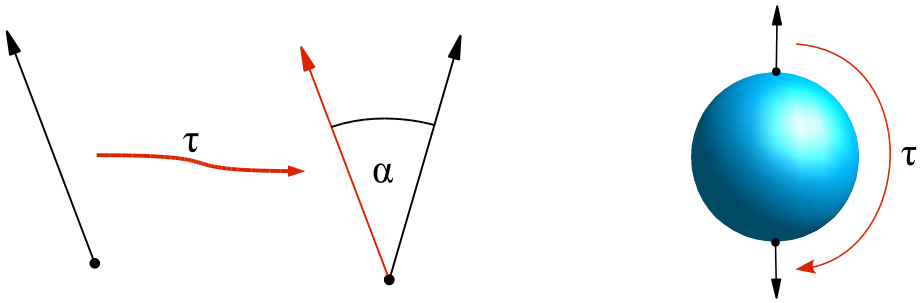


Figure 2.4: To compare two vectors at different locations in space, we need to transport one to the location of the other (def. 44). This operation is only trivial if the underlying manifold is flat, Euclidean space. Think e.g. about a vector field on a sphere, where we are not interested in the direction change of a vector in the embedding flat space around, but relative to the surface.

Definition 44 *Let $p_0, p \in M$ be points in a manifold M and $v \in \mathcal{T}^1(M)$ a vector field. The **tangential transport** τ is a linear map from the tangential space at the point p into the tangential space at point p_0:*

$$\begin{aligned} \tau_{p_0}(p,-): \quad T_p(M) &\longrightarrow T_{p_0}(M) \\ v(p) &\longmapsto \tau_{p_0}(p, v(p)) \end{aligned}$$

with the property that at the point p_0 itself the tangential transport is the identical map, i.e.

$$\tau_{p_0}(p_0, v(p_0)) = v(p_0)$$

The elegant concept of a tangential transport was invented by P. Wagner [Wag94] and is not identical with the more common concept of parallel transport, but is equivalent in first order. Linearity means that for $\lambda, \mu \in \mathbb{R}$ and $v, w \in T_p(M)$:

$$\tau_{p_0}(p, \lambda \cdot v(p) + \mu \cdot w(p)) = \lambda \cdot \tau_{p_0}(p, v(p)) + \mu \cdot \tau_{p_0}(p, w(p))$$

Thus we may write in a chart $\{x^\mu\}$:

$$\tau_{p_0}(p, v^\mu(p)\ \partial_\mu|_p) = v^\mu(p)\tau_{p_0}(p,\ \partial_\mu|_p) =: v^\mu(p)\tau^\nu_\mu(p_0,p)\ \partial_\mu|_{p_0}$$

The matrix elements τ^ν_μ are the *components* of the tangential transport from p to p_0 in the chart $\{x^\mu\}$. It depends on the distance between the points p and p_0 (as measured by the curve parameter of an arbitrary connecting curve, note that we don't need a metric here) and becomes the identical matrix for $p = p_0$. We may discover the structure of the coordinate expression τ^ν_μ further if we write the tangential transport τ of a vector field $v : M \to T_p(M)$ along a curve q as a Taylor series for small curve parameters s (i.e. close to the point $q(0)$):

$$\begin{aligned}\tau_{q(0)}\left(q(s), v(q(s))\right) = v^\mu(q(s))\ \partial_\mu|_{q(0)} + s\cdot\frac{\partial\,\tau_{q(0)}\left(q(s), v(q(s))\right)}{\partial\, x^\mu}\cdot\frac{d}{ds}q^\mu(0) + \\ \frac{s^2}{2}\cdot\frac{\partial^2\,\tau_{q(0)}\left(q(s), v(q(s))\right)}{\partial\, x^\mu\ \partial\, x^\nu}\cdot\frac{d}{ds}q^\mu(0)\frac{d}{ds}q^\nu(0) + \cdots = \\ v^\mu(q(0))\ \partial_\mu|_{q(0)} + s\cdot v^\nu(q(0))\underbrace{\frac{\partial\tau_{p_0}\left(q(0), \partial_\nu(q(0))\right)}{\partial\, x^\mu}}_{=:\nabla^\sigma_{\mu\nu}\partial_\sigma}\cdot\dot{q}^\mu(0) + \frac{s^2}{2}\cdots = \\ v^\mu\partial_\mu + s\cdot v^\nu\nabla^\sigma_{\mu\nu}q^\mu\partial_\sigma + \frac{s^2}{2}\ldots\end{aligned}$$

The tangential transport can thus be written in first order as

$$\tau_{q(0)}\big(q(s), v(q(s))\big) = v^\mu\tau^\nu_\mu\partial_\nu = v^\mu\underbrace{\left(\delta^\nu_\mu + s\cdot\nabla^\nu_{\sigma\mu}\dot{q}^\sigma\right)}_{\approx\ \tau^\nu_\mu}\partial_\nu \qquad , \tag{2.37}$$

or, in words: we may write the distance-dependent "tangential transport matrix" τ in a linear approximation as the unit matrix plus the distance times a "transportation matrix increase" for each direction. The expression $\nabla^\nu_{\sigma\mu}\dot{q}^\sigma$ gives the increase of the transportation matrix in the direction $\dot{q}$, which can be written as linear combination of the differences $\nabla^\nu_{\sigma\mu}$ along the coordinate line of x^σ. Now we are able to define the derivative of a vector field:

Definition 45 *The change of a vector field $v \in \mathcal{T}^1(M)$ under the action of a tangential transport τ along a curve q is called the* ***directional derivative*** *∇ of the vector field v in the direction of the tangential vector $\dot{q}$:*

$$\nabla_{\dot{q}}v := \frac{d}{ds}\left[\tau_{q(0)}\left(q(s), v(q(s))\right)\right]_{s=0} = \lim_{s\to 0}\frac{\tau_{q(0)}\big(q(s), v(q(s))\big) - v(q(0))}{s} \tag{2.38}$$

Now let us inspect the coordinate expression of this operation. (2.38) is of the form

$$\frac{d\, f(g(x), h(g(x)))}{d\, x} = \frac{\partial f(g,h)}{\partial g}\frac{d\, g(x)}{d\, x} + \frac{\partial f(g,h)}{\partial h}\frac{d\, h(g(x))}{d\, x}$$

therefore it can be written in a chart $\{x^\mu\}$ as

$$\begin{aligned}\nabla_{\dot{q}}v := \frac{d}{ds}\left[\tau_{q_0}\left(q_s, v(q_s)\right)\right]_{s=0} \\ = \left.\frac{\partial\,\tau_{q_0}\left(q_s, v(q_s)\right)}{\partial\, x^\mu}\right|_{s=0}\left.\frac{d\, q^\mu_s}{ds}\right|_{s=0} + \left.\frac{\partial\,\tau_{q_0}(q_s, v(q_s))}{\partial\, v}\right|_{s=0}\left.\frac{d\, v(q_s)}{ds}\right|_{s=0}\end{aligned} \tag{2.39}$$

whereby we know from $\tau_p(p,-) \equiv id$ that

$$\left.\frac{\partial\,\tau_{q_0}(q_s, v(q_s))}{\partial\, v}\right|_{s=0} = \frac{\partial\,\tau_{q_0}\left(q_0, v(q_0)\right)}{\partial\, v} = \frac{\partial\, v(q_0)}{\partial\, v} = 1$$

And with the linearity of the derivation we may write using def. 2.2:

$$\left.\frac{d\, v(q_s)}{ds}\right|_{s=0} = \left.\frac{d\,(v^\sigma\partial_\sigma)}{ds}\right|_{s=0} = \left.\frac{d\, v^\sigma}{ds}\right|_{s=0}\partial_\sigma|_{s=0} = \frac{\partial v^\sigma}{\partial x^\mu}\left.\frac{dq^\mu_s}{ds}\right|_{s=0}\partial_\sigma = v^\sigma{}_{,\mu}\,\dot{q}^\mu\partial_\sigma$$

The first term in (2.39) becomes:

$$\left.\frac{\partial\,\tau_{q_0}\,(q_s, v(q_s)))}{\partial\,x^\mu}\right|_{s=0} = \left.\frac{\partial\,\tau_{q_0}\,(q_s, v^\nu(q_s)\partial_\nu)}{\partial\,x^\mu}\right|_{s=0}$$
$$= v^\nu(q_s))|_{s=0}\left.\frac{\partial\,\tau_{q_0}\,(q_s, \partial_\nu)}{\partial\,x^\mu}\right|_{s=0} = v^\nu(q_0)\frac{\partial\tau_{q_0}(q_0,\partial_\nu)}{\partial x^\mu}$$
$$= v^\nu(q_0)\nabla^\sigma_{\mu\nu}(q_0)\partial_\sigma$$

Finally by inserting into (2.39) we get the coordinate expression for the directional derivative:

$$\nabla_{\dot q} v = v^\nu\nabla^\sigma_{\mu\nu}\partial_\sigma\dot q^\mu + v^\sigma{}_{,\mu}\,\dot q^\mu\partial_\sigma = \left(v^\nu\nabla^\sigma_{\mu\nu} + v^\sigma{}_{,\mu}\right)\dot q^\mu\partial_\sigma =: v^\sigma_{;\mu}\dot q^\mu\partial_\sigma \tag{2.40}$$

Whereby $v^\sigma_{;\mu} := v^\nu\nabla^\sigma_{\mu\nu} + v^\sigma{}_{,\mu}$ is called the **covariant derivative** of the vector field v (written in components). This derivation procedure involves an external structure, the tangential transport (which is usually just provided in first order by specification of the Christoffel symbols), but is independent from a specific coordinate system.

The definition of the covariant derivative can be extended to arbitrary tensor fields T. It always yields another tensor field ∇T. ∇ is called the **connection** or connection operator. Note that ∇ itself is *not* a tensor. In a chart $\{x^\mu\}$ we get along a curve:

$$\nabla_{\dot q}\dot q = \dot q^\sigma_{;\mu}\dot q^\mu\partial_\sigma = (\dot q^\nu\nabla^\sigma_{\mu\nu} + \dot q^\sigma{}_{,\mu})\dot q^\mu\partial_\sigma = \dot q^\nu\dot q^\mu\nabla^\sigma_{\mu\nu}\partial_\sigma + \ddot q^\sigma\partial_\sigma \tag{2.41}$$

$\nabla^\sigma_{\mu\nu}$ are the components of the connection in a coordinate representation. These are n^3 numbers for a n-dimensional manifold. The connection coefficients obey the same transformation rule (2.36) as the Christoffel symbols.

Definition 46 *The **acceleration** of a worldline is the directional derivative of the wordline's four-velocity (def. 33) along the worldline, i.e.:*

$$a := \nabla_u u = \nabla_{\dot q}\dot q$$

whereby the last expression is valid for curves which are parameterized by their arc length.

The norm of the acceleration $|a| \equiv g(a,a) = 0$ is zero, i.e. the acceleration is a *spacelike* vector and it is orthogonal to the four-velocity, i.e. $g(a,u) = 0$ (see e.g. (6.2) in Misner,Thorne and Wheeler [MTW73]).

Note that this definition supports the physical principle of covariance, def. 23. An example is Newton's law $F = m \cdot a$: some "force" is a physical action which is independent from an observer. Clearly, the force which might disrupt an object must not depend on any observers viewing this event, especially disregarding their motion. "Force" is an absolute quantity, like the mass of an object is independent from its observers (we view Newton's law in a non-relativistic context here!). Thus, consequently, also the acceleration is a absolute quantity. However, *the acceleration as measured by some observer is **not** the second derivative of a curve along its parameter* $\ddot q$, as it is frequently believed and wrongly taught. The special case where $a = \ddot q$ is only true for observers who perform their measurements in a coordinate system where the connection components $\nabla^\sigma_{\mu\nu}$ vanish, according to (2.41). These observers are called *inertial*. Other observers will need to "know" their connection components to correctly determine the acceleration and thus the true (physical, observer-independent) force. If they fail to do so and misleadingly interpret $\ddot q$ as the object's acceleration, then they will find components of the "force" which are not due to physical forces, but only due to the observer's measurement method. This are the so-called virtual forces like the coriolis "force" and the centrifugal "force". They actually are the components of the connection coefficients in a rotating coordinate system, being interpreted as part of the left side of Newton's law instead of the right side. A detailed calculation demonstrating the connection coefficients in a rotating coordinate system is given in [Rot99], pp.44.

It is an often heard misconception (or, more diplomatic, "alternative interpretation") that "the laws of nature appear the same to all observers, as long as they are not accelerating" (example cited from Betts [Bet98], which was emended in the subsequent article by Weiskopf [DW00]) and sometimes, even worse, that the theory of special relativity cannot handle acceleration. This is simply not true. The standard book for general relativity "Gravitation" by Misner, Thorne, Wheeler [MTW73] even dedicates its entire chapter six to accelerated observers in special relativity. The misconception arises when identifying the term "acceleration" with "second time derivative", which introduces coordinate-dependence by this assumption. However, if the coordinate-free definition of acceleration is employed, we can simply drop the accentuation of a special group of observers and state the theorem as "the laws of nature appear the same to all observers".

EXAMPLE: ACCELERATION IN A ROTATING COORDINATE FRAME. Instead of presenting a detailed computation, we give a short alternative derivation using more familiar concepts from Euclidean vector space calculus. Assume we know the path of some object in some inertial system, i.e. $\vec{r}(t) : \mathbb{R} \to \mathbb{R}^3$ is known. We may split this vector into its norm and unit direction vector

$$\vec{r} = |r| \cdot \vec{e}_r$$

and intend to compute the acceleration $\ddot{\vec{r}}$ along this curve. The velocity as the first derivative becomes (writing $r = |r|$):

$$\dot{\vec{r}} = \underbrace{\dot{r}\ \vec{e}_r}_{\vec{v}_r} + r\ \underbrace{\dot{\vec{e}}_r}_{=:\vec{\omega}\times\vec{e}_r} = \dot{r}\vec{e}_r + r\,(\vec{\omega}\times\vec{e}_r)$$

We model the change of the tangential velocity an angular velocity vector $\vec{\omega}$ around the coordinate origin. Now the second derivative becomes:

$$\ddot{\vec{r}} = \ddot{r}\ \vec{e}_r + \dot{r}\ \underbrace{\dot{\vec{e}}_r}_{\vec{\omega}\times\vec{e}_r} \quad + \quad \dot{r}\,(\vec{\omega}\times\vec{e}_r) + r\left(\dot{\vec{\omega}}\times\vec{e}_r\right) + r(\vec{\omega}\times\underbrace{\dot{\vec{e}}_r}_{\vec{\omega}\times\vec{e}_r}) =$$

$$\ddot{r}\ \vec{e}_r + 2\dot{r}\vec{\omega}\times\vec{e}_r + r\dot{\vec{\omega}}\times\vec{e}_r + r\omega^2\vec{e}_r \equiv \ddot{r}\ \vec{e}_r + 2\vec{\omega}\times\vec{v}_r + \dot{\vec{\omega}}\times\vec{r} + \omega^2\vec{r}$$

Now for an observer who is rotating around the coordinate origin with $\dot{\vec{\omega}} = 0$ the acceleration is the second derivative of the radial distance plus some additional terms, which are known as the coriolis and the centrifugal acceleration. Any physical forces on some mass m couple with the absolute acceleration $\ddot{\vec{r}}$ via Newton's law $m\ddot{\vec{r}} = \vec{F}$. If the rotating observer unknowingly interprets the second derivative of his coordinate change $\ddot{r}\ \vec{e}_r$ as "the acceleration", he will find the coriolis and centrifugal terms as some spooky forces without physical cause, because a physical force F will not couple on $m\ddot{r}\ \vec{e}_r \neq \vec{F}$. Instead, the second derivative of his coordinate location $\ddot{r}$ is determined by $m\ddot{r}\ \vec{e}_r = \vec{F} - m2\vec{\omega}\times\vec{v}_r - m\omega^2\vec{r}$, hereby interpreting parts of the acceleration (the expressions involving $\vec{\omega}$ are actually the components of the connection in a rotating frame) as force.

Newton also stated that an object without (physical!) force acting on it will move on a straight line:

Definition 47 *A line is **straight** if the acceleration is all zero along its curves, i.e.*

$$q \quad \textit{straight} \qquad \Longleftrightarrow \qquad \nabla_{\dot{q}(s)}\dot{q}(s) = 0 \quad \forall s$$

If we compare this definition with the geodesic equation (2.35) we see, that a connection may also be used to define geodesics by setting the connection coefficients $\nabla^\sigma_{\mu\nu} = \Gamma^\sigma_{\mu\nu}$. In this case, the shortest/longest lines (geodesics) are also straight lines. If Christoffel symbols exist, they induce a connection, which is called the *Levi-Civita-connection* (see fig. 2.5). However, this requires that the underlying metric field

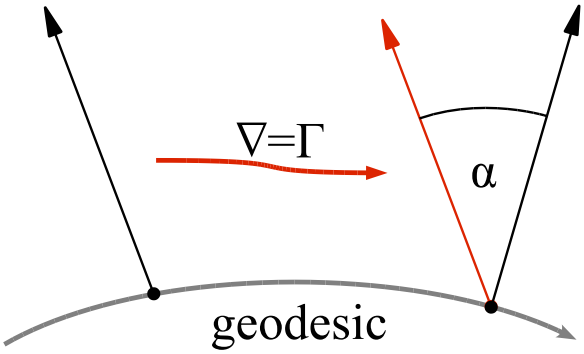


Figure 2.5: The tangential transport may be constructed by transporting tangential vectors along geodesics (e.g. the shortest connecting line on a surface). This special tangential transport is called the Levi-Civita Connection.

is invertible (see (2.34)) - which is, for instance, not the case in the Newtonian spacetime [Rot99]. Here the connection needs to be chosen independently from the metric with the constraint to be compatible, i.e. $\nabla g = 0$. This choice is similar to the choice of a time 1-form, i.e. is the introduction of an *ether*. Nevertheless, in special and general relativity the metric is non-degenerated and we may use the Christoffel symbols as a connection.

REMARK: A straight line will no longer appear as a straight line for non-inertial observers, who interprets "straight" as a linear coordinate expression of the form $q^\mu(s) = q^\mu(0) + s\dot{q}^\mu$, the solution of $\ddot{q}^\mu = 0$. But still, they are physically straight lines, even if some observation reveals a "curve".

Now general relativity questions the existence of inertial observers. The concept of inertial observers is troublesome, because a non-inertial observer needs to know his relation to an inertial one, but so how can any observer know that he is inertial if any one needs an "anchor"? What is the "root anchor"? Thus general relativity abandons the concept of inertial observers completely and states that the connection is the Levi-Civita connection of the underlying spacetime metric, which is no longer flat and can be measured. The spacetime metric is determined by the mass distribution in space and time, objects moving in space and time do not interact with other massive objects through spooky forces (like in newtons' theory of gravitation), but they feel the geometry of spacetime and move along geodesics. Consequently, the orbit of a satellite around earth or of planets around the sun is a straight line in the notion of general relativity and any object in free fall is *unaccelerated*. For instance, a particle orbiting earth is unaccelerated, whereas an object resting on the surface of earth is accelerated (upwards, like in a rocket which is just hovering at constant height).

An interesting aspect is that the simple theorem from special relativistic electrodynamics "an accelerated charge emits electromagnetic radiation" is no longer valid in the general relativistic context: an orbiting, but unaccelerated electrically charged particle will emit electromagnetic radiation, while a charged particle sitting at rest at the surface of earth does not emit electromagnetic radiation – where should the energy of the electromagnetic radiation come from in this static, time-independent situation? In contrast, the charged particle orbiting the electrically neutral earth would loose energy via its electromagnetic radiation and thus follow another trajectory than a neutral particle (i.e. it will obey some acceleration due to the back-reaction of the emitted radiation). Einstein's principle of equivalence (all masses behave identical in a gravitational field) is thus only valid "locally" or for electrically neutral masses, but violated once electrical charges come into account (as confirmed by J. Ehlers [WE]). This issue is rarely heard in recent presentations, but was discussed by Bondy and Gold [BG55], and recently mentioned as one of the alleged paradoxes in relativity also by W. Rindler [Rin93].

Riemann Curvature Tensor and Einstein Equation

When computing the second covariant derivative, we find that in contrast to the second partial derivative this operation does no longer commute. This difference in the commutation of the second covariant derivative is expressed by a tensor, known as the Riemann curvature tensor R:

$$K(u,v)w = \left[\left(w^\nu_{;\mu;\sigma} - w^\nu_{;\sigma;\mu}\right) + w^\nu_{;\kappa}\left(\nabla^\kappa_{\sigma\mu} - \nabla^\kappa_{\mu\sigma}\right)\right] v^\mu u^\sigma \partial_\nu$$

The Riemann curvature tensor is defined coordinate-free via the relationship:

$$K(u,v)w := \nabla_u\nabla_v w - \nabla_v\nabla_u w - \nabla_{[u,v]}w \tag{2.42}$$

Its coordinate expression involves differences and derivatives of the Christoffel symbols:

$$K^\kappa_{\mu\lambda\beta} = \nabla^\kappa_{\lambda\mu,\beta} - \nabla^\kappa_{\beta\mu,\lambda} + \nabla^\rho_{\lambda\mu}\nabla^\kappa_{\beta\rho} - \nabla^\rho_{\beta\mu}\nabla^\kappa_{\lambda\rho}$$

A detailed derivation of the Riemann tensor based on the notion of the tangential transport can be found in the diploma thesis "Voids" [Ben97]. The Riemann curvature tensor obeys certain symmetry and anti-symmetry relationships, which leads to just $\frac{1}{12}n^2(n^2-1)$ independent components in n dimensions instead of n^4 (i.e. 20 independent components on a four-dimensional spacetime). The only non-vanishing contractions define the symmetric Ricci tensor:

$$R_{\mu\nu} := K^\kappa{}_{\mu\kappa\nu}$$

In the case of a Levi-Civita connection $\Gamma = \nabla$ the Ricci-tensor $R(v,v)$ with $v \in T_P(M)$ computes the sum of the Gaussian curvatures $\mathfrak{K}$ of $n-1$ mutually orthogonal surfaces that are span by geodesics that contain v:

$$R(v,v) = g(v,v)\sum_{i=1}^{n-1}\mathfrak{K}_i$$

This interpretation is due to Ricci himself, as pointed out e.g. by Eisenhart [Eis49], p.113. Finally, the *Einstein field equation* relates the Ricci tensor to the momentum tensor $T_{\mu\nu}$ which describes the distribution of matter in space:

$$R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}\left(R_{\sigma\rho}R^{\sigma\rho}\right) = T_{\mu\nu} \tag{2.43}$$

and is the central equation of general relativity. For vacuum solutions, like black holes, the Ricci tensor has to vanish: $R = 0$.

2.1.3 Fiber, Vector, and Tangential Bundles

Definition 48 *The **fibers of a function** $f : X \longrightarrow Y$ are the pre-images or inverse images of the points $y \in Y$, i.e. the sets of all elements $x \in X$ with $f(x) = y$:*

$$f^{-1}(y) = \{x \in X \mid f(x) = y\}$$

is a fiber of f (at the point y).

A fiber can also be the empty set. The union set of all fibers of a function is called the *total space*.

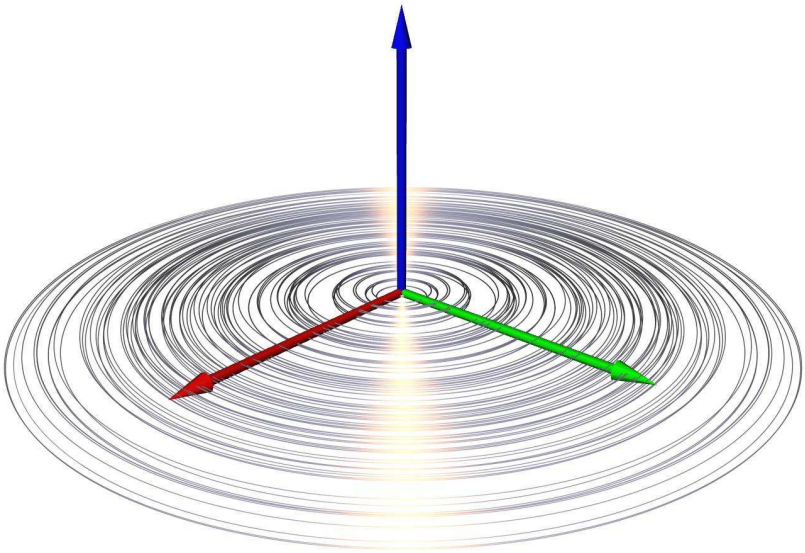


Figure 2.6: The fibers of the function $f : \mathbb{R}^2 \longrightarrow \mathbb{R} : f(x, y) \longmapsto x^2 + y^2$ are concentric circles ($f > 0$), the origin ($f = 0$) or empty sets ($f < 0$).

Definition 49 *The **projection map** pr_1 is a function that maps each element of a product space to the element of the first space:*

$$\begin{array}{rcl} pr_1 : X \times Y & \longrightarrow & X \\ (x, y) & \longmapsto & x \end{array}$$

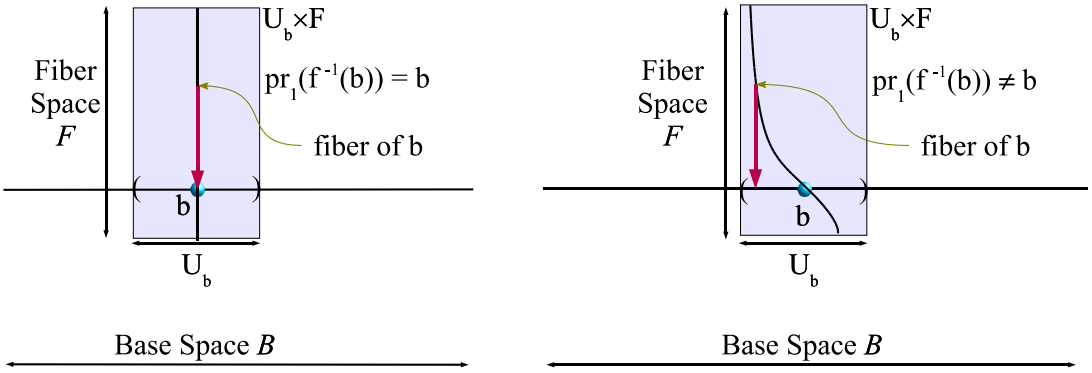
Definition 50 *Let E, B be topological spaces and $f : E \longrightarrow B$ a continuous map. (E, B, f) is called a **(fiber) bundle** if there exists a space F such that the union of fibers of a neighborhood $U_b \subset B$ of each point $b \in B$ are homeomorphic to $U_b \times F$ such that the projection pr_1 of $U_b \times F$ is U_b again (see fig. 2.7 for illustration):*

$$(E, B, f : E \to B) \quad \textit{bundle} \qquad \Longleftrightarrow \qquad \begin{array}{c} \exists F : \forall b \in B : \exists U_b : f^{-1}(U_b) \overset{\text{hom}}{\simeq} U_b \times F \\ \textit{and} \quad pr_1(U_b \times F) = U_b \end{array}$$

E is called the total space *E, B is called the* base space *and $f : E \longrightarrow B$ the* projection map.

The space F is called the *fiber type*[2] of the bundle or simply the **fiber** of the bundle. In other words, the total space can be written locally as a product space of the base space with some space F. The notation $\mathcal{F}(B) = (E, B, f)$ will be used to denote a fiber bundle over the base space B. It is also said that the *space F fibers over the base space B*.

[2] Note that the fibers of the bundle *projection* $f : E \to B$ is a subset of the total space. The term "fiber" occurs in two different meanings: as the "fiber of the projection map" and as "fiber space of the bundle". The first term refers to a subset of the total space (which includes the base space), the last term refers to the fiber space F (which does not include the base space).



(a) A fiber bundle with appropriate homeomorphism.

(b) A homeomorphism into $U_b \times F$, which does not preserve the projection, thus not revealing a fiber bundle.

Figure 2.7: The constraints on the fiber homeomorphism: the projections of the fibers of a point must reveal the same point (in the base space). The right example shows a space that is locally homeomorphic to $U_b \times F$, but is not a fiber bundle over U_b, whereas the left example is a fiber bundle.

An important case is the tangent bundle:

Definition 51 *The union of all tangent spaces $T_p(M)$ on a manifold M together with the manifold is called the* ***tangent bundle*** *$\mathcal{T}(M)$:*

$$\mathcal{T}(M) := \{(p, v) : p \in M, v \in T_p(M)\} \tag{2.44}$$

Every differentiable manifold possesses a tangent bundle $\mathcal{T}(M)$. The dimension of $\mathcal{T}(M)$ is twice the dimension of the underlying manifold M, its elements are points plus tangential vectors. The mapping $\pi : \mathcal{T}(M) \to M$ is called a *natural projection*. The inverse image of a point $p \in M$ is the tangent space $T_p(M)$. $T_p(M)$ is the fiber of the tangent bundle over the point p.

Definition 52 *If a fiber bundle over a space B with fiber F can be written as $B \times F$ globally, then it is called a* ***trivial bundle*** *$(B \times F, B, pr_1)$.*

Examples for non-trivial fiber bundles are the Möbius strip or the Hopf bundle that consists of total space $\mathbb{S}^3$ with base space $\mathbb{S}^2$ and fiber $\mathbb{S}^1$ (a nice discussion can by found on J. Baez' web site [Bae99]). If a manifold can be written as a product space, then its tangential bundle can be written as product of the respective tangential bundles:

$$\mathcal{T}(X \times Y) = \mathcal{T}(X) \times \mathcal{T}(Y)$$

It follows, that if a manifold M is a trivial bundle, i.e. $M = B \times F$, then its tangential bundle is trivial as well:

$$M \text{ trivial} \quad \Rightarrow \quad \mathcal{T}(M) \text{ trivial} \quad ,$$

also the negation is true:

$$\mathcal{T}(M) \text{ non-trivial} \quad \Rightarrow \quad M \text{ non-trivial} \quad .$$

However, a non-trivial manifold may still posses a trivial tangential bundle:

$$M \text{ non-trivial} \quad \not\Rightarrow \quad \mathcal{T}(M) \text{ non-trivial} \quad .$$

For various points $b_1, b_2 \in B$ their neighborhoods U_{b_1}, U_{b_2} overlap, i.e. $U_{b_1} \cap U_{b_2} \neq \emptyset$ (strictly speaking: such neighborhoods can be found that...). It is interesting to investigate how the fibers of both point neighborhoods are related in the overlap. In general this relationship is a diffeomorphism (a differentiable homeomorphism) on F, written $Diff(F)$, but in certain cases these diffeomorphisms can be reduced to a special group structure. A fiber bundle with a group structure $G \subset Diff(F)$ is called a *G-bundle*. For instance, the fibers of a tangential bundle are the tangential spaces, which are related via coordinate

transformation matrices (2.4). The coordinate transformation matrices are members of the group of general linear invertible functions Gl_n, thus the tangential bundle is a Gl_n-bundle:

$$\begin{array}{ccc} \mathcal{T}(U_1 \cap U_2) & \xrightarrow[\{x^\mu\}]{} & U_1 \cap U_2 \times \mathbb{R}^n \\ \| & & \downarrow \alpha^\mu_{\bar{\mu}} \in Gl_n \\ \mathcal{T}(U_1 \cap U_2) & \xrightarrow[\{x^{\bar{\mu}}\}]{} & U_1 \cap U_2 \times \mathbb{R}^n \end{array} \tag{2.45}$$

An interesting question is whether the structure group of a tangential bundle can be reduced further to the group of special linear invertible functions $Sl_n := \{A \in Gl_n : \det A = 1\}$, i.e. can the entire manifold be described by charts whose respective transformation matrices (2.4) have determinant one? In general, this is not possible. Manifolds whose tangential bundle has the structure group Sl_n are called *orientable*.

Definition 53 *If the fiber of a bundle is a vector space and its structure group is Gl_n, then it is called a **vector bundle**.*

Examples:

- The tangential bundle of a manifold is a vector bundle.
- The infinite Möbius strip and the infinite cylinder $\mathbb{S}^1 \times \mathbb{R}$ are vector bundles, because the fiber $\mathbb{R}$ is a vector space.
- The Hopf bundle is not a vector bundle, because the fiber $\mathbb{S}^1$ is not a vector space.

Any tensor is an element of a vector space, thus all tensor fields can be described via vector bundles. In contrast, the Christoffel symbols are not tensors and thus need to be described via the more general fiber bundles.

Definition 54 *A map $\sigma : B \to E$ is called a **section** of a fiber bundle $(E, B, p : E \to B)$ if $p \circ \sigma : B \to B = id$.*

A section gives an element of the fiber over every point in B.
Example: A vector field is a section of the tangent bundle. The *zero section* of a vector bundle is the submanifold of the bundle that consists of all the zero vectors. In general, a vector bundle of rank (i, j) is spanned locally by $i + j$ independent sections.

A "generalization" (with limitations) of fiber bundles are **sheaves**. They are also subject of research for data models and will thus be shortly described here. According to Kodairo's book [Kod86] a fiber sheave is defined as follows:

Definition 55 *A topological space E is called a **sheaf** over a space B if*

I.) a local homeomorphism $f : E \longrightarrow B$ exists,

II.) $\forall b \in B$ the fiber $f^{-1}(b)$ is a vector space on $\mathbb{R}^n$, whereby $n \in \mathbb{N}$ or $n = \infty$ (in most cases),

III.) for two numbers $c_1, c_2 \in \mathbb{R}$ and $p, q \in E$ the linear combination $c_1 p + c_2 q$ with $f(p) = f(q)$ depends continuously on p, q.

A sheaf can be imagined as a fiber space F that can change at each point of the base space B, but in general a sheaf is not a fiber bundle. There is some overlap among both concepts when the fiber is a discrete set with the discrete topology (i.e. the topology is the full power set, see p.10). The fibers of a sheaf are called **stalks**.

Example:

- Let $f^{-1}(p)$ be the set of equivalence classes of functions on p; two functions f, g are equivalent on a point p if there exists a neighborhood U_p on p such that $\forall x \in U_p : f(x) = g(x)$.

2.2 Modeling Discretized Manifolds

2.2.1 Vertices, Cells and Complexes

For computational purposes, a topological space is modeled by a finite set of points. These points naturally carry a discrete topology (the topology constructed by the full power set, see example 4 on p.10 for definition) by themselves, but one usually considers embeddings in a space that is homeomorphic to Euclidean space to define various structures describing their relationships. Based on the definitions of a metric space, def. 11, and k-balls, def. 13 the definition of a *cell* is as follows:

Definition 56 *A subset $c \subset X$ of a Hausdorff space X (def. 15) is a **k-cell** if it is homeomorphic to an open k-dimensional ball (def. 13) in $\mathbb{R}^n$.*

The dimension of the cell is k. 0-cells are called vertices, 1-cells are edges, 2-cells are faces or polygons, 3-cells are polyhedra. An n-cell within an n-dimensional space is just called a "cell". $(n-1)$-cells are sometimes called "facets" and $(n-2)$-cells are known as "ridges". For k-cells of arbitrary dimension, we may define incidence and adjacency relationships

Definition 57 *Two cells c_1, c_2 are **incident** if $c_1 \subseteq \partial c_2$, where ∂c_2 denotes the border of the cell c_2.*

Two cells of the same dimension can never be incident because $dim(c_1) \neq dim(c_2)$ for two incident cells c_1, c_2. c_1 is a *side* of c_2 if $dim(c_1) < dim(c_2)$, which may be written as $c_1 < c_2$. The special case $dim(c_1) = dim(c_2) - 1$ may be denoted by $c_1 \prec c_2$.

Definition 58 *Two $k-$cells c_1, c_2 with $k > 0$ are called **adjacent** if they have a common side, i.e.*

$$\text{cell } c_1, c_2 \textbf{ adjacent} \iff \exists \text{ cell } f: \ f < c_1, f < c_2 \quad .$$

For $k = 0$ two 0-cells (i.e. vertices) v_1, v_2 are said to be adjacent if there exists a 1-cell (edge) e which contains both, i.e. $v_1 < e$ and $v_2 < e$.

Incidence relationships form an incidence graph. A path within an incidence graph is a cell-tuple:

Definition 59 *A **cell-tuple** $\mathcal{C}$ within an n-dimensional Hausdorff space is an ordered sequence of k-cells $(c_n, c_{n-1}, \ldots, c_1, c_0)$ of decreasing dimensions such that $\forall 0 < i \leq n : c_{i-1} \prec c_i$.*

These relationships allow to determine topological neighborhoods: Adjacent cells are called *neighbors*. The set of all $k+1$ cells which are incident to a k-cell forms a neighborhood of the k-cell. The cells of a Hausdorff space X constitute a topological base, leading to the following definition:

Definition 60 *A ("closure-finite, weak-topology") **CW-complex** $\mathcal{C}$, also called a **decomposition** of a Hausdorff space X, is a hierarchical system of spaces $X^{(-1)} \subseteq X^{(0)} \subseteq X^{(1)} \subseteq \cdots \subseteq X^{(n)}$, constructed by pairwise disjoint open cells $c \subset X$ with the Hausdorff topology $\bigcup_{c \in \mathcal{C}} c$, such that $X^{(n)}$ is obtained from $X^{(n-1)}$ by attaching adjacent n-cells to each $(n-1)$-cell and $X^{(-1)} = \emptyset$.*

The respective subspaces $X^{(n)}$ are called the n-skeletons of X. A CW-complex can be understood as a set of cells which is glued together at their subcells. It generalizes the concept of a graph by adding cells of dimension greater than 1.

Up to now, the definition of a cell was just based on a homeomorphism of the underlying space X and $\mathbb{R}^n$. Note that a cell does not need to be "straight", such that e.g. a 2-cell may be constructed from a single vertex and an edge connecting the vertex to itself, as e.g. illustrated by J.Hart [Har99]. Another approach toward the definition of cells is more restrictively based on isometry to Euclidean space, defining the notion of "convexity" first:

Definition 61 *A subset K of the Euclidean space $\mathbb{R}^n$ is **convex**, if for any two points p, q the connection line is contained within K.*

E.g. each half space of $\mathbb{R}^n$ is convex. The intersection of convex sets is convex again. Any subset S of $\mathbb{R}^n$ is a subset of a convex set of $\mathbb{R}^n$. The intersection $\bigcap K_n$ of all convex sets $K_n \subseteq \mathbb{R}^n$ which contain $S \subseteq K_n$ is called the **convex hull** of S. Based on the notion of convexity, simplices are defined:

Definition 62 *The convex hull of a set of $k+1$ points $p_0, p_1, \ldots, p_k$ within a Euclidean space of n dimensions with $k \leq n$, whereby these $k+1$ points are not contained within a k-dimensional subspace, is called a **simplex** of dimension k or k-simplex.*

The points $p_0, p_1, \ldots, p_k$ are called the **vertices** of a k-simplex, which is consistent with the former definition of a 0-cell as vertex. A CW-complex built by simplices of an Euclidean space is a called a **simplicial complex**. The dimension of a simplicial complex is defined as the maximum dimension of its simplices: $dim(C) := max\{dim(S) : S \in C$.

Definition 63 *A topological space is **triangulable** if it is homeomorphic to a simplicial complex.*

Such a topological space is also called a **polyeder**. The decomposition of a topological space into a simplicial complex is called a triangulation.

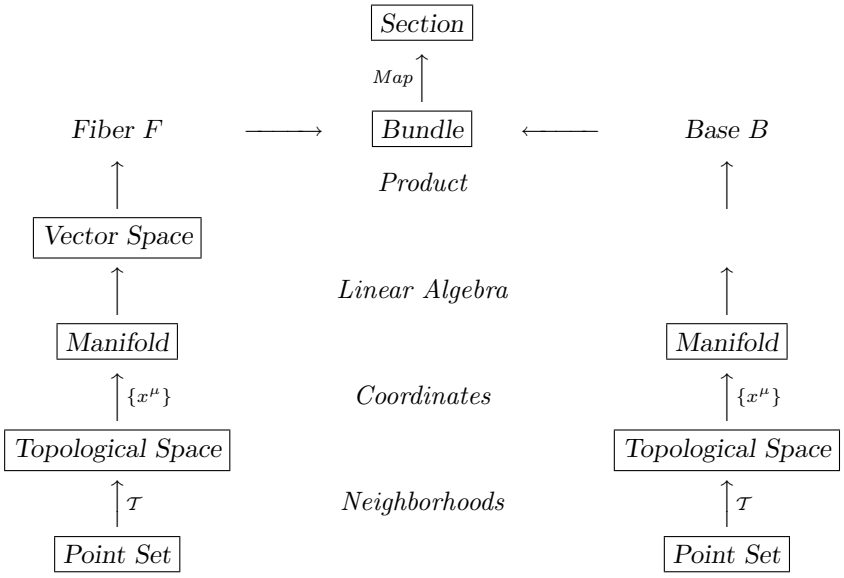
2.2.2 Butler's Vector Bundle Approach

> The proper abstractions for scientific data are known.
> We just have to use them. *David Butler, 1992 [BB92].*

When dealing with the visualization of discretized manifolds, one encounters many different implementations. The desire to make use of common properties is obvious. This task of finding a generalization scheme to handle all these branches within the same framework was tackled by Butler and Pendley [BP89] in an inspiring approach: They proposed to use the abstractions provided by the concept of a vector bundle for deducting of a class hierarchy. Their recipe was to follow the construction of manifolds via repeated aggregation of simpler objects to form a layered structure.

At the lowest level, both the base and fiber are point sets, just collections of points X. For the next layer, the notion of neighborhoods is added to obtain a topological space $T := (X, \mathcal{T})$ like in def. 1. Then the notion of coordinates and differentiability is added to get a manifold $M := (T, \{\{x^\mu\}\})$, like in def. 19. The fiber of interest here is a vector space, which can be considered as a manifold again, but with an additional layer of structure, i.e. the structure of linear algebra in the case considered here. The next layer then aggregates the base space B and the fiber F into a bundle (B, F). Finally the bundle is aggregated with a map, which allows to specify values in each fiber to yield a section.

The layer concept as presented in [BP89] is demonstrated in the following diagram:



(2.46)

It is limited to trivial bundles. Nontrivial bundles like the Möbius strip would require additional structures which allow to describe some "twisting". A non-trivial fiber bundle is no longer a cartesian product of base and fiber space globally, but just locally. Butler only considered trivial fiber bundles as relevant for visualization purposes.

Operations on the vector bundle data model were categorized into the three categories of *constructor*, *mapping* and *section* operations. These constructors are:

1. Base constructors: operations which deal with the structure of the base space
2. Fiber constructors: operations that specify the structure of the fiber
3. Bundle constructors: operations which glue base and fiber together
4. Section constructors: operations that deal with the values of a section

Mappings are used to create objects from existing ones. Example mappings are:

1. Bundle restriction: the base space is changed (decreased/increased) while the fiber is kept constant.
2. Subbundle: the base space is kept constant, but the fiber changes (i.e. parts of the fiber are removed or the fiber is extended)
3. Boolean operations: similar to the methods of constructive solid geometry, such mappings create union, intersection or difference sets of the point sets of two underlying base spaces. The respective fibers are available in the result.
4. Slicing: a certain kind of bundle restriction, where the base space is reduced by one dimension.
5. Stacking: the inverse of slicing, where a family of lower-dimensional bundles is composed into an higher-dimensional one.
6. Component projection: reducing the dimension of the fiber by extracting a single component, e.g. one of three from a 3D vector field.
7. Component direct product: a bundle formed by taking the direct product of the fibers of two bundles which are given over the same base; it can be used as the inverse operation of a component projection.

Section operations involve all properties of a section. Examples are:

1. Approximation: interpolating or smoothing a section
2. Linear algebra: multiplying sections by a constant, adding or subtracting two sections defined on the same bundle
3. Calculus: integrating or differentiating a section.

Finally, a classification scheme for scientific data was derived from the vector bundle model. It is the triple (d_b, d_f, m), where d_b is the dimension of the base space, d_f the dimension of the fiber and m the multiplicity. E.g. a "component projection" of a three-dimensional vector field given on a plane would be an operation $(d_b = 2, d_f = 3, m = 1) \mapsto (d_b = 2, d_f = 1, m = 3)$ in this scheme of vector bundle sections.

Implementation

In the succeeding paper Butler and Bryson gave [BB92] a prototype implementation in the Eiffel programming language. It demonstrated how to construct a class hierarchy and added the concept of multiple charts on a manifold and its fiber space. An application example, the computation of streamlines of a vector field was given to demonstrate the applicability and simplicity of the data model for visualization purposes.

The implementation from the paper is shown here as an experimental translation into C++ . Due to the different abstraction capabilities in the languages Eiffel and C++ , it cannot be an exact translation, but the code fragments are tried to be as close to the spirit of the Eiffel source as possible. Hereby efficiency and other implementation aspects have been neglected, as only the intention of the code shall be shown. The code by itself is not meant to be fully syntactically correct. The reader is assumed to be familiar with conventional C++ ; no template techniques are used, but the definition of abstract classes via pure virtual functions is a central functionality.

The class hierarchy starts with the definition of a point set X, which is modeled as a finite set of yet unspecified elements, such that only the number of elements (with consecutive indexing) is specified:

```
struct  PointSet
{
        int     count;
};
```

The original code also included an iterator class for the point set which is omitted here. A topological space $T := (X, \mathcal{T})$, def. 2, is a point set with neighborhood information:

```
struct  TopologicalSpace : PointSet
{
        int []  neighborhood(int i) = 0;
};
```

The neighborhood information assigns a set of points to a given point. A differentiable manifold (def. 19) $M := (T, \{x^\mu\})$ is a topological space with coordinate patches ("charts", def. 17) and coordinate transformations (coordinate transition rules). A Chart object is formulated as an object that allows to determine the coordinate representation of a point on a manifold, together with the inverse mapping and a range determination:

```
struct Chart
{
        // coordinates of point p
        virtual float [] coords(int p) = 0;
        // returns point with coordinates r
        virtual int      coordinv(float r []) = 0;
        // returns true if coordinates are in the range of the chart
        virtual bool     in_range(float r []) = 0;
        // index of adjoining chart if in_range() has returned false
        int      adjoining;
};
```

The "Transition" object (a similar implementation will be shown in 3.2.3) allows to evaluate the coordinate transition function $\{x^{\bar{\mu}}\} \circ \{x^\mu\}^{-1}$ at a certain point:

```
struct Transition
{
        Chart   *domain, *range;
        // returns coordinates in chart 'range' corresponding
        // to coordinates r in chart 'domain', i.e.
        virtual float [] value(float r []) = 0;
};
```

The Manifold object then is a topological space with a set of charts:

```
struct Manifold : TopologicalSpace
{
        int             dimension;
        int             ncharts;
        // return the nth chart {x^mu}
        virtual Chart       chart(int which) = 0;
        // return the coordinate transition functions {x^mubar} -> {x^mu}^-1
        virtual Transition  transition (int from, int to) = 0;
};
```

Note that the `Chart` and `Transition` object mentioned above are bound to a manifold object. They would ideally be formulated as nested classes, but this implementation enhancement is omitted here.

A vector bundle $\mathcal{F}(B) = (E, B, f)$ has a total space E, a base space B, a projection function f, coordinate patches and transition functions.

```
struct VectorBundle
{
        int     dim_base, dim_fiber, dim_total, dim_bundle;
        Manifold*base;
        int     ncharts;
        virtual VecChart            chart(int which) = 0;
        virtual VecTransition  transition (int from, int to) = 0;
};
```

The classes `VecChart` and `VecTransition` are similar to the definition of `Chart` and `Transition` on a manifold, but their return values are designated for the vector space components only, i.e. they return the representation of a vector field in a certain chart (2.3) and the transformation matrix def. 2.4. The section class finally delivers the field values for each chart and offers the vector space operations of addition and scalar multiplication:

```
struct VectorBundleSection
{
        // The base space, as obtained from the bundle
        Manifold*base;
        // The bundle to which this section belongs
        VectorBundle*range;
        // Returns the data values of the current field in the given chart.
```

```
        virtual VecChart            chart(int which) = 0;
        // Multiplies the section by a scalar function
        VectorBundleSection scaleby(float scalar(int) );
        // Adds the current section to another one and returns the result
        VectorBundleSection scaleby(VectorBundleSection&other);
};
```

As an example of a neighborhood implementation, some neighborhood iterator may compute a point's neighborhood from multidimensional indices within a regular block of data, whereby the index by itself is just a linear offset. Multiple blocks are handled by covering each block by its own chart. The computation of a streamline of a vector field is demonstrated in the following code example; hereby the second vector bundle section P carries the "physical coordinates" or a grid point, whereby the multidimensional indices are used as "computational coordinates", as they are returned by the manifold's chart functions:

```
void streamline(VectorBundleSection*V, VectorBundleSection*P)
{
int     c=1;
Manifold Base = VectorBundleSection->base;
        r1 = Base->chart(c)->coord(p1);
        v1 = V->chart(c)->coord(p1);
        r2 = r1 + dt * v1;

        while(c>0)
        {
                if (Base->chart(c).in_range(r2) )
                {
                        p2 = Base->chart(c).coordinv(r2);
                        x1 = P->chart(c).coord(p1);
                        x2 = P->chart(c).coord(p2);
                        draw_line(x1,x2);
                        p1 = p2;
                        r1 = Base->chart(c)->coord(p1);
                        v1 = Base->chart(c)->coord(p1);
                        r2 = r1 + dt * v1;
                }
                else
                {
                        cadj = Base->chart(c).adjoining;
                        v1 = V->transition(c, cadj).value(r1, v1);
                        r1 = Base->transition(c, cadj).value(r1);
                        r2 = r1 + dt * v1;
                        c  = cadj;
                }
        }
}
```

The concept is to cover a manifold, which cannot be described with a single chart (e.g. due to some coordinate singularities), with a number of overlapping coordinate patches (i.e. an *atlas*), such that computing integral curves and similar quantities is possible on the entire manifold by choosing an appropriate chart at each point.

Critics

Although Butler's approach was path-leading, it does not handle certain requirements that arise in practical usage. Especially, there is no concept of cells and skeletons on a discretized manifold. The use of cells instead of vertices is of particular importance when interpolating data on an arbitrary location in coordinate space. In the context of general relativistic tensor field visualization, it is moreover of central importance to distinguish among covariant and contravariant tensors (lower and upper indices). Christoffel symbols, which are not tensors, must be distinguishable from tensorial objects. In other words, the three-dimensional classification scheme (d_b, d_f, m) with d_b the dimension of the base space, d_f the dimension of the fiber space and m the multiplicity is not sufficient to describe all properties of the fiber.

2.2.3 Numerical Relativity

3+1 Decomposition

While the Einstein equation (2.43) nicely describes a complete spacetime at once, this equation is not appropriate in its general form for numerical computations. Here, the four-dimensional field equations are often split into spatial and temporal components, which is known as the $3+1$ formalism: We assume that a spacetime $(M, g, \uparrow)$ admits a slicing into spacelike subspaces Σ, i.e. there is a diffeomorphism $\phi : M \to \Sigma \times I$ with $I \subset \mathbb{R}$ such that the manifolds $\Sigma_t = \phi^{-1}(\Sigma \times \{t\})$ are spacelike and the curves $\phi^{-1}(\{x\} \times I)$ are timelike (see Straumann [Str96] for a detailed discussion). These timelike curves define a vector field ∂_t, which can be decomposed into normal and parallel components relative to the slicing $\vec{\partial_t} = \alpha\vec{n} + \vec{\beta}$. $\vec{n}$ is a unit normal vector on Σ and $\vec{\beta}$ is tangent to the slices Σ_t; it is called the *shift vector field*. α is called the *lapse function*.

A coordinate system $\{x^i\}$ on Σ induces natural coordinates $\{t, x^i\}$ on M. In these coordinates the Einstein equations become a system of coupled differential equations, a first system describing some constraints on the three-dimensional space:

$$\begin{aligned} R + \mathrm{tr}^2 K - K^{ab}K_{ab} &= 16\pi\varrho \\ \left(K^{ab} - \gamma^{ab}\mathrm{tr}K\right)_{;b} &= 8\pi p^a \quad , \end{aligned}$$

and another system the evolution of the space quantities in time:

$$\begin{aligned} g_{ab,t} &= -2\alpha K_{ab} + \beta_{b;a} + \beta_{a;b} \\ K_{ab,t} &= -\alpha_{;b;a} + \alpha\left(R_{ab} - 2K_{ac}K^c_b + K_{ab}\mathrm{tr}K\right) + \beta^c K_{ab;c} + K_{ac}\beta^c_{;b} + K_{cb}\beta^b_{;a} \ . \end{aligned}$$

The quantities and operations will not be explained here, a review can be found e.g. in Bruegmann [Bru00, BMSW98]. The four-dimensional metric g becomes

$$ds^2 = \left(\alpha^2 - \beta_i\beta^i\right)dt^2 - 2\beta_i dx^i dt - \gamma_{ij}dx^i dx^j = \alpha^2 dt^2 - \gamma_{ij}(dx^i + \beta^i dt)(dx^j + \beta^j dt) \quad (2.47)$$

or alternatively in matrix notation

$$(g_{\mu\nu}dx^\mu\, dx^\nu) = \begin{pmatrix} \alpha^2 - \beta^i\gamma_{ij}\beta^j & -\beta_i \\ -\beta_i & -\gamma_{ij} \end{pmatrix} \quad . \quad (2.48)$$

In numerical relativity [AGL+99, Bru00, ABB+01], α, β and γ are the primary computational quantities. Hereby γ is given as a set of initial data at some time $t = \text{const.}$ (computing these initial data is a huge task by itself) and evolved via some clever choice of α and β. Both functions α and β are free to choose, they are not determined by physical means but may severely impact the stability and precision of a numerical evolution. A good choice is one of the recipes for a numerically stable evolution. The spatial metric γ needs to be provided for some initial time and is then evolved with Einstein's equations of the gravitational field.

A crucial problem is the difficulty of *grid stretching*, when the metric distance as measured with the numerical metric (2.47) between neighboring points on a given grid becomes too large or even infinite. E.g. the spatial distance to the event horizon is infinite in Schwarzschild coordinates – it is thus not a good idea to use Schwarzschild coordinate slicing for numerical relativity.

An important issue when developing new evolution schemes and coordinate slicings is thus the detection of grid stretching. Where does it occur and in what direction? A direction visualization of the metric tensor field, as it is developed in this thesis, is thus of significant importance. If the shift vector field is given in its covariant form $\beta_i dx^i$, computing $|\beta|$ for visualization purposes requires calculating the contravariant form via the inverse of the spatial metric, i.e. $\beta^i = \beta_j\gamma^{ij}$. In practice it is preferable to avoid matrix inversions and use a Gaussian elimination algorithm for solving linear equation systems of the form $Ax = b \Rightarrow x$, which leads to faster and numerically stabler results when solving $\gamma_{ij}\beta^i = \beta_j \ \Rightarrow \beta^j$. Nevertheless the distinction between $\beta_i dx^i$ and $\beta^i\partial_i$ is crucial and must be supported by a suitable data model.

Four-dimensional formulation of Maxwell equations

The 3+1 decomposition of Einstein's equation is similar to the Maxwell equations for electrodynamics. In its classical form, these equations are

$$\begin{aligned} \mathrm{div}\vec{B} &= 0 \qquad \mathrm{rot}\vec{E} - \tfrac{\partial\vec{B}}{\partial t} = 0 \\ \mathrm{div}\vec{E} &= \epsilon_0\varrho \qquad \mathrm{rot}\vec{B} + \tfrac{\partial\vec{E}}{\partial t} = -\mu_0\vec{j} \end{aligned}$$

It was the great achievement of Maxwell to find a unified theory for electricity (denoted by the electrical vector field $\vec{E}$) and magnetism (expressed by the magnetic vector field $\vec{B}$). Each of these equations has a certain physical meaning. E.g. the first equation claims that there are no magnetic monopoles and the second one tells that a time-dependent magnetic field causes an electric field.

However, while usually $\vec{E}$ and $\vec{B}$ are treated as vector fields, they are not really vector fields. Actually the Maxwell equations are a 3+1 decomposition of a four-dimensional tensor equation, similar to the Einstein equations. This can be seen when we inspect the behavior of a charged particle, e.g. an electron, under the influence of an electromagnetic field. The ("Lorentz") force $\vec{F}$ which an electron experiences is

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \quad .$$

Let us consider a case where an electron moves in a homogeneous magnetic field orthogonal to its initial velocity $\vec{v}$. Here, $\vec{E} = 0$, and, as it is well known, the solution of the equation of motion is a circle around an axis parallel to the $\vec{B}$ field.

But now let us imagine a physicist sitting in a train, which is traveling through the $\vec{B}$ field with the same velocity as the initial velocity of the electron (see fig. 2.8) i.e. the electron resides at rest on the

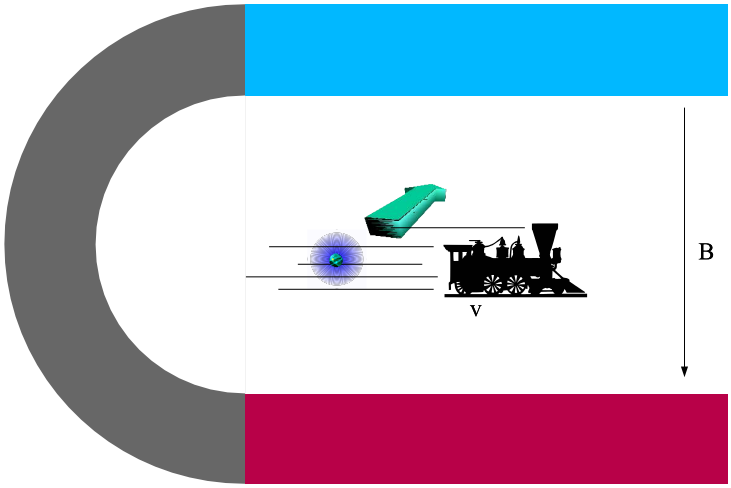


Figure 2.8: If an electrical charge e is moved in a magnetic field B, it will experience a sideway force $\vec{F} = e\ \vec{v} \times \vec{B}$. However, if ether theory is abandoned and velocity treated as a relative quantity - as seen from a train with the same speed, the velocity of the charge is zero, so why does it move sideways? Considering this situation actually guided Einstein to his theory of relativity [Rin93].

finger tip of the physician in a train's cabin. The velocity of this electron, as seen by the physician, is zero. So whatever kind of magnetic field the physicist would measure, it cannot influence the electron. On the other hand, we know that the electron will start to move and follow a circular path as seen from outside, so it cannot remain at rest in the hand of the traveling physicist. So what ghostly force can demand the movement of the electron as seen by the physicist, if it is not the magnetic field? The only source left for such a force is the electric field, meaning that the moving physicist will experience such an electrical field when passing through the homogeneous magnetic field. However, a vector (field) which is zero in one coordinate system, will be zero in any coordinate system. Thus, this electrical field ghostly appearing "out of nothing" just due to the motion of the observer can impossibly be a "usual" vector field. Instead, it must be related to the magnetic field, giving rise to the idea that in reality these are not independent vector fields, but components of some more complex object.

Thorough mathematical analysis using the framework of differential geometry has shown that they can be seen as components of "an antisymmetric tensor field of rank two" in a four-dimensional spacetime. This antisymmetric tensor field is called the Faraday field F. Such a field has six independent components in four dimensions, which is exactly the number of components which are available in the two three-dimensional (pseudo-)vector field $\vec{E}$ and $\vec{B}$. These components, written in matrix notation, are identified

usually as:

$$F = (F_{\mu\nu}) = \begin{pmatrix} 0 & E_1 & E_2 & E_3 \\ -E_1 & 0 & B_3 & -B_2 \\ -E_2 & -B_3 & 0 & B_1 \\ -E_3 & B_2 & -B_1 & 0 \end{pmatrix} \tag{2.49}$$

Where $''\vec{B}'' = (B_1, B_2, B_3)$ and $''\vec{E}'' = (E_1, E_2, E_3)$. So actually the electrical and magnetic field E, B are just components of a four-dimensional tensor written "as if they were vectors". But this decomposition depends on the choice of coordinates: while in one coordinate system the E-components are zero, in another one they are not. The Maxwell equations in their original form, treating of the electrical and magnetic field as two "vector fields", are thus just valid in exactly one certain coordinate system, that one of the "ether". As we nowadays know by experiments that such an ether does not exist, we need a superior formulations of the equations in a coordinate-free manner, i.e. independent of the choice of coordinates.

Written in this four-dimensional formalism, the "calculus of differential forms", the Maxwell equations may be formulated elegantly using operations like the exterior differential ("d") and the Hodge star operator ("$*$"):

$$dF = 0 \tag{2.50}$$

$$d * F = -\mu_0 J \tag{2.51}$$

Hereby, 2.50 contains the first two Maxwell equations, which actually are one tensor equation instead of two vector equations, and 2.51 contains the third and fourth Maxwell equations, which relates the Faraday field to charged sources. $J = (\varrho, \vec{j})$ is the four-dimensional matter current. When the Faraday field is examined in a certain coordinate system, its components may be measured as the electrical and magnetic field, as quantities $\vec{E}$ and $\vec{B}$ relative to this coordinate system. By transforming the Faraday field into another coordinate system, e.g. of a moving observer utilizing the Lorentz transformations as known from special relativity, its components change such that part of an $\vec{E}$ field transits into the $\vec{B}$ field and vice versa.

Coordinate-Free Formulation

The method of decomposing the four-dimensional Einstein equations into a set of three-dimensional equations is similar to treating the four-dimensional equations for the Faraday field as the usual Maxwell equations by introducing some observer. Before special relativity, the $3+1$ formulation of the Maxwell equations was the only one. It was considered to be the "right" physical formulation, because it was thought that there is a special and unique spacelike slicing of the spacetime, meaning that there is a global time which is valid for everyone everywhere in the universe, independent from some movement or other influence. This concept of the ether was proved to be false (at least in the form as it was used originally), and todays physical insight is that there is no special coordinate system which is "more physical than others", and all observers are equivalent. Although the three-dimensional Maxwell equations are known to be valid only in one special coordinate system, they are still widely used and the four-dimensional formulation of the Maxwell equations is hardly known outside of the community of theoretical physicists.

Two examples demonstrate that the four-dimensional formalism may actually be easier than their corresponding $3+1$ notation:

1. From $F = 0$ the lemma of Poincaré tells that F is an exact two-form, i.e. F can be written as

   $$F = dA \quad ,$$

   meaning that F "has a potential". A is the four-dimensional vector potential of the electromagnetic field. In the $3+1$ decomposition, this equation corresponds to the two equations

   $$\vec{E} = \mathrm{div} A_0 \qquad \text{and} \qquad \vec{B} = \mathrm{rot}\vec{A}$$

   whereby A_0 is the electrostatic potential and $\vec{A}$ is the magnetic vector potential.

2. Combining $F = dA$ with $d * F = -\mu_0 J$, this leads in vacuum ($J = 0$) to the equation

   $$d * dA = 0$$

   whereby $d * d \equiv \Box$ is the d'Alambert or wave operator. The solutions of $\Box A = 0$ are waves of the form $A_\mu = A_\mu|_{\vec{x}=0,t=0} \sin(\vec{k}\vec{x} \pm \omega t + \varphi)$, indicating the existence of electromagnetic waves in vacuum.

The idea of the data model approach as part of this thesis is to cope the same level of expressiveness by hiding component-wise expressions behind an abstract, coordinate-free layer.

2.3 Physical Significance of Tensor Fields

2.3.1 Inspected Spacetimes

For developing numerical and visualization algorithms it is clearly useful to test these upon known solutions which can be studied analytically as well.

Schwarzschild Metric

A few months after Einstein had published his final equations of the gravitational field in his publication of the theory of General Relativity, Karl Schwarzschild found a solution for the spherically symmetric vacuum case. Later Birkhoff has proven that this solution was the only one. This solution is known as the Schwarzschild metric describing the gravitational field within a spherically symmetric spacetime:

$$ds^2 = \left(1 - \frac{2m}{r}\right) dt^2 - \frac{1}{1 - \frac{2m}{r}} dr^2 - r^2 d\Omega^2 \tag{2.52}$$

It contains a true singularity at $r = 0$ and a coordinate singularity at $r = 2m$, which is known as the Schwarzschild radius. This coordinate singularity does not appear in other coordinate systems such as the Eddington-Finkelstein coordinates or the Kruskal-Skeresz coordinates. The Schwarzschild spacetime contains a spherical region out of which nothing can escape, because the drag of the gravitational field is stronger than the escape velocity of light. This region is known as a black hole; its boundary is known as an event horizon. The term "black hole" was coined by the US relativity expert John Archibald Wheeler in the year 1967. The alternative naming, especially in the eastern hemisphere, was "frozen stars" (see pp.290 in [Tho93]), that emphasizes the infinite apparent time delay of infalling light and particles on the event horizon.

The Schwarzschild spacetime can be described in many coordinates, which then determine the properties of the spatial 3-metric that is the subject of 3-dimensional tensor field visualization, as investigated here (primarily, but not exclusively). The original coordinates that had been used by Karl Schwarzschild for deriving his solutions describe the spacetime as seen from a static observer who is at rest relative to the mass contained in this spacetime.

Kerr metric

The Kerr solution was found as an exact solution of Einstein's equation in the year 1967, and it was discovered that it describes a rotating black hole. Up to present, no way of deriving the Kerr solution in a physically reasonable way has been found (Straumann [Str96], Chandrasekhar [Cha83]), but it can be proven that it is the only solution describing a rotating (i.e. stationary and axially symmetric) black hole – at least in the outside region, it is still unclear if the solution is also unique in the interior part beyond the event horizon. It describes a rotating black hole with mass m and angular momentum a. In Boyer-Lindquist coordinates $\{t, r, \vartheta, \varphi\}$ the Kerr-Newmann solution [Str91] (which includes electrical charge) is given by

$$\begin{aligned} ds^2 =& (1 - \frac{2mr - Q^2}{\varrho^2}) dt^2 + \frac{(2mr - Q^2) a \sin^2 \vartheta}{\varrho^2} dt d\varphi \\ & - \frac{\varrho^2}{\Delta} dr^2 - \varrho^2 d\vartheta^2 - \frac{(r^2 + a^2) - \Delta a^2 \sin^2 \vartheta}{\varrho^2} \sin^2 \vartheta d\varphi^2 \quad , \end{aligned} \tag{2.53}$$

or, alternatively written in a shorter form involving differences of basis vectors

$$ds^2 \quad = \quad \frac{\Delta}{\varrho^2} \left[dt - a \sin^2 \vartheta d\varphi\right]^2 \quad - \quad \frac{\sin^2 \vartheta}{\varrho^2} \left[(r^2 + a^2) d\varphi - a dt\right]^2 \quad - \quad \frac{\varrho^2}{\Delta} dr^2 \quad - \quad \varrho^2 d\vartheta^2 \quad , \tag{2.54}$$

whereby the abbreviations $\Delta = r^2 - 2mr + a^2 + Q^2$ and $\varrho = r^2 + a^2 \cos^2 \vartheta$ are used. The constant m is interpreted as the mass of the black hole and a as its angular momentum. The condition $a < m$ must hold for a physically reasonable black hole. The case $a = m$ is known as a maximally spinning black hole. For $a = 0$ the Kerr metric is identical to the Schwarzschild metric. In matrix notation, the metric is

$$g = (g_{\mu\nu}) = \begin{pmatrix} 1 - \frac{2mr - Q^2}{\varrho^2} & 0 & 0 & \frac{1}{2} \frac{(2mr - Q^2) a \sin^2 \vartheta}{\varrho^2} \\ 0 & -\frac{\varrho^2}{\Delta} & 0 & 0 \\ 0 & 0 & -\varrho^2 & 0 \\ g_{t\varphi} & 0 & 0 & -\frac{(r^2 + a^2) - \Delta a^2 \sin^a \vartheta}{\varrho^2} \sin^2 \vartheta \end{pmatrix} \quad .$$

As special cases we get

Q=a=m=0:	Minkowski flat space, see (2.27)
Q=a=0:	Schwarzschild solution
a = 0	Reissner-Nordström solution of electrically charged static black holes
Q = 0	Kerr solution

The Kerr solution contains two event horizons $r_\pm = m \pm \sqrt{m^2 - a^2 - Q^2}$. The outer one at $r = r_+$ corresponds to the event horizon of the Schwarzschild solution, while the inner one has the property of a Cauchy horizon, which is not healthy to cross [Str96, Cha83]. It will not be discussed further here, but it is just mentioned that the entire outside part of the universe becomes visible there as an intense, infinitely blue-shifted flash for an observer crossing the horizon. Beyond the inner event horizon, the future of an observer is no longer determined by his past and the central singularity becomes visible and avoidable, i.e. the observer may chose a path leading to a parallel universe without falling into the central singularity. Moreover, when inspecting the $g_{\varphi\varphi}$ component of the Kerr metric, one finds that there exist certain regions in the $r < r_-$ domain where ∂_φ becomes timelike (2.1.1) i.e. $g_{\varphi\varphi} > 0$. Since φ is still a periodic coordinate with $\varphi + 2\pi = \varphi$, this indicates the existence of closed timelike lines in the inner region of the Kerr metric (like mentioned in 2.1.2 before).

Warp metric

Science and science fiction always formed a mutually inspiriting pair. Superluminal space travel, rendered to be impossible by Einstein's theory of special relativity, is a substantial essence of interstellar travel scenarios, and one of the earliest workaround solutions is the idea of a warp drive. In 1994 Miguel Alcubierre [Alc94] showed that the properties of a warp drive can be mimiced by a simple construction of spacetime; he demonstrated on a concrete example that apparently global superluminal travel is possible without violating the local speed limit of special relativity. The idea is to build a spacetime where the shift vector field β in (2.47) smoothly increases. Thus the velocity of neighboring points, expressed by the difference of the shift vector field, is always less than the speed of light, but the numerical value of the shift vector field may be arbitrary large, indicating an arbitrary speed in the chosen coordinate system. If we chose an asymptotically flat metric, we thus achieve arbitrary speed as compared to Minkowski space. The shift vector field β can easily be modeled by an arbitrarily large number v^i and a function $f(r)$ as $\beta^i = v^i f(r)$ whereby $\lim_{r\to\infty} f(r) = 0$, $f(0) = 1$ and r is the coordinate distance $r = \sqrt{(x - vt)^2 + y^2 + z^2}$ from some (moving) point of interest, e.g. the location of a spaceship (assuming movement in x-direction for simplicity hereby). The Warp metric can then be written as:

$$ds^2 = dt^2 - (dx - vf(r)dt)^2 - dy^2 - dz^2 \tag{2.55}$$

or, in matrix notation,

$$(g_{\mu\nu}) = \begin{pmatrix} 1 - v^2 f^2(r) & vf(r) & 0 & 0 \\ vf(r) & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \quad . \tag{2.56}$$

For $f(r)$ we chose some plateau function, such that the metric hardly changes around $r = 0$ – then the metric is just the Minkowski metric in a coordinate system moving at – arbitrary – speed v there, i.e. a space ship would experience no acceleration there as the Christoffel symbols (2.34) $\nabla^\sigma_{\mu\nu} = 0$ vanish around $r = 0$. In [Alc94] the function f was chosen as

$$f(r) = \frac{\tanh\Big(\sigma\ (r + R)\Big) - \tanh\Big(\sigma\ (r - R)\Big)}{2\ \tanh(\sigma R)} \tag{2.57}$$

but any other function fulfilling the desired asymptotic conditions like $f(r) = \frac{1}{1+\sigma r}$ would serve as well. In (2.57), R is the "size" of the "warp bubble" and σ is the falloff parameter, which denotes the "sharpness" of the transition between flat, unmoving and flat, moving space. In the intermediate region, the spacetime is not flat (although the spatial metric γ is flat everywhere). A visualization is given in fig. 4.35. By inserting the Warp metric into the Einstein equations, we may compute the energy-momentum tensor of the matter distribution that is required to produce the desired Warp spacetime. One finds that this spacetime requires a region of negative energy density. Up to our current knowledge, it is not clear whether matter of such kind may exist in our universe. However, under certain circumstances negative

energy densities occur, e.g. due to the Casimir effect or when accelerating a mirror (as pointed out by W.Israel [Isr97]). Nevertheless, the energy required to move a space ship of reasonable size has been estimated to be of the order of a thousand solar masses, which will be quite unrealistic to achieve for human technology in the next couple of years.

Brill Waves

In 1959, D.Brill [Bri59] has shown that black holes may also be constructed from pure vacuum with no matter[3] involved. This is due to the fact that gravitational energy has a mass equivalent like other forms of energy. Small fluctuations of pure gravitational energy in asymptotically flat space, i.e. weak gravitational waves, will just dissipate, like the waves created by a drop of water. However, if the concentration of gravitational energy is strong enough, the spacetime contains an event horizon – a black hole. The metric considered by Brill, with spatial components

$$ds^2 = \psi^4 \left[e^{2q}(d\varrho^2 + dz^2) + \varrho^2 d\varphi^2 \right] \quad ,$$

describes an axisymmetric gravitational wave with a toroidal shape. Brill himself could only consider initial data to find if such a spacetime does or does not contain a black hole right from the beginning Numerical simulations performed at AEI [AGL$^+$99, Bru00] could evolve these initial data in time and found that spacetimes without initial event horizon formed a black hole after some time once a critical value of the amplitude was reached. The study of such solutions is of certain interest because they allow to inspect the creation of black holes without assumptions of state equations for matter.

2.3.2 Mathematical Properties of Rank-2 Tensors

The metric tensor field is described by a $n \times n$ tensor on a n-dimensional manifold. Such a $n \times n$ tensor $G \in T_p^*(M) \times T_p^*(M)$ is represented by a quadratic matrix $G_{\mu\nu}$ in a coordinate system. To get an intuition what the numerous values of the tensor's representation mean, we can analyze this matrix by decomposing it into scalar- or vector-components which have a straightforward interpretation. Also, we may inspect invariants, that can be computed from the tensor in a coordinate-free way.

Symmetric/Antisymmetric Decomposition

The matrix transpose of tensor components yields a tensor again (the transposed tensor components obey the same coordinate transition rule). This allows for the decomposition of an arbitrary tensor into a symmetric and an antisymmetric component:

$$G = \underbrace{\frac{G + G^T}{2}}_{S} + \underbrace{\frac{G - G^T}{2}}_{A}$$

i.e. $S_{ij} = (G_{ij} + G_{ji})/2$ and $A_{ij} = (G_{ij} - G_{ji})/2$.

For a 3-dimensional space, the antisymmetric part is constructed by three components, which can be interpreted as a vector via cyclic identification $v_1 = A_{23}$, $v_2 = A_{31}$, $v_3 = A_{12}$:

$$A = \begin{pmatrix} 0 & A_{12} & A_{13} \\ -A_{21} & 0 & A_{23} \\ -A_{31} & -A_{23} & 0 \end{pmatrix} = \begin{pmatrix} 0 & v_3 & -v_2 \\ -v_3 & 0 & v_1 \\ v_2 & -v_1 & 0 \end{pmatrix} \tag{2.58}$$

Thus any 3×3 tensor can be imagined as a symmetric tensor (visualized as a quadric surface, 2.3.2, which is independent of the antisymmetric part), and a vector, or better as the rotation around an axis given by this vector, because (2.58) has a structure similar to the cross-product; the choice of righthanded/lefthanded rotation corresponds to the choice of sign in $v_1 = \pm A_{23}$ etc. This kind of decomposition is also known as *polar decomposition*.

[3] It is interesting to remark that such solutions are actually in contradiction to Mach's principle, which states that local inertial systems are related to matter around. Gravitational waves like Brill waves demonstrates that in General Relativity local inertial systems may also be influenced by "pure, matter-free" space. So although Einstein's theory was greatly influenced by Mach's principle, it actually contain's "Anti-Mach-solutions" [Rin93].

Quadric Surface

A straightforward way to look at a tensor in a coordinate-independent way is to make use of its definition as a map $T_p(M) \times T_p(M) \to \mathbb{R}$, that maps two directions to a number. We then consider the set of tangential vectors $\vec{v} \in T_p(M)$ which are mapped to the same number

$$G(\vec{v}, \vec{v}) = \mathcal{C} \quad . \tag{2.59}$$

The coordinate expression in a chart $\{x^\mu\}$ with $\vec{v} = \{x, y, z\}$ is given by

$$G(\vec{v}, \vec{v}) = G_{xx}x^2 + (G_{xy} + G_{yx})xy + G_{yy}y^2 + (G_{yz} + G_{zy})yz + G_{zz}z^2 + (G_{zx} + G_{xz})xz \tag{2.60}$$

and corresponds to the equation of a quadric surface, i.e. an ellipsoid or an hyperboloid, depending on the signs. This quadric surface can be computed in any coordinate system by mapping unit vectors $\vec{u} \in T_p(M)$ with $|\vec{u}| = 1$ to

$$\vec{v} = \frac{\vec{u}}{\sqrt{G(\vec{u}, \vec{u})/\mathcal{C}}} \quad , \tag{2.61}$$

which obviously fulfills $G(\vec{v}, \vec{v}) = \mathcal{C}$ (compare with the definition of the four-velocity def. 33). If the tensor is not positive definite, then $\bar{u}$ becomes complex for $\mathcal{C} > 0$ in certain directions, i.e. the quadric surface is a hyperbolic surface then. However, a quadric surface does not display the full information content of a general tensor but only its symmetric part. The antisymmetric part does not contribute to the quadric surface, but can be displayed as an additional vector field.

Trace and Anisotropy

The trace of an $n \times n$ tensor G is an invariant and computed by index contraction

$$tr(G) := G_{\mu\nu}\delta^\nu_\mu = \sum_\mu G_{\mu\mu} \quad .$$

With η the unit matrix, the tensor

$$D := G - \frac{1}{n}\, tr(G)\, \eta \tag{2.62}$$

is tracefree ($tr(D) = 0$, $tr(\eta) = n$ is the dimension of the underlying manifold). It is called the *deviator* of the tensor G, while $\frac{1}{n}\, tr(G)\, \eta$ is the *isotropic part* of the tensor G. The isotropic part does not contain any directional information; using the deviator can thus be used to emphasize the anisotropy of a tensor by "removing isotropic background".

Eigendecomposition

The most important decomposition of a matrix is given by searching for the directions which are unchanged under the action of the matrix, i.e. we look for vectors $v \in T_p(M)$ and scalar values $\lambda \in \mathbb{C}$ for which

$$Gv = \lambda v \quad . \tag{2.63}$$

These solutions are known as the eigenvector and the eigenvalue, respectively. With η the unity matrix, a solution exists if

$$\det(G - \lambda\eta) = 0 \tag{2.64}$$

(the characteristic equation) which yields a polynomial of order n in λ with n roots, i.e. there are n (not necessarily independent) eigenvectors $v_{(i)}$ and eigenvalues $\lambda_{(i)}$ (we use bracketed indices here to distinguish among this numbering scheme from covariant and contravariant indices). The n equations (2.63) can be written as one matrix equation

$$G_{ij} \underbrace{v^j_{(k)}}_{=:T} = v^i_{(k)}\lambda_{(k)} \equiv v^i_{(k)} \underbrace{\delta^{kl}\lambda_{(l)}}_{=:\Lambda} \quad . \tag{2.65}$$

If G is symmetric, then the eigenvectors are orthogonal and the eigenvalues are real. If further G has full rank n, then the matrix T of the eigenvectors can be inverted and (2.65) can be written as

$$T^{-1}GT = \Lambda \tag{2.66}$$

or

$$G = T\Lambda T^{-1} \equiv \begin{pmatrix} v_{(1)} & v_{(2)} & v_{(3)} \end{pmatrix} \begin{pmatrix} \lambda_{(1)} & & \\ & \lambda_{(2)} & \\ & & \lambda_{(3)} \end{pmatrix} \begin{pmatrix} v_{(1)} \\ v_{(2)} \\ v_{(3)} \end{pmatrix} \quad , \tag{2.67}$$

i.e. the tensor representation matrix G can be diagonalized to Λ. The matrix T can be seen as a transformation matrix into another (local) coordinate system, where the matrix representation of the tensor object G is diagonal.

Since the trace is invariant under coordinate transformation, it can be read off easily in this coordinate system that the trace is the sum of all eigenvalues, i.e.

$$tr(G) \equiv \sum_i \lambda_{(i)} \quad .$$

For a symmetric 3×3 tensor we may order the three real eigenvalues according to their value and name them the maximum, medium and minor eigenvalues $\lambda_{max}, \lambda_{med}, \lambda_{min}$ with $\lambda_{max} \geq \lambda_{med} \geq \lambda_{min}$. Eigenvalues are invariant under coordinate transformations.

In a coordinate system using the eigenvectors as basis vectors $\{\vec{v}_{max}, \vec{v}_{med}, \vec{v}_{min}\}$ the equation of the quadric surface (2.60) becomes:

$$G(v, v) = \lambda_{max}\ x^2 + \lambda_{med}\ y^2 + \lambda_{min}\ z^2 \tag{2.68}$$

with $v = x\ \vec{v}_{max} + y\ \vec{v}_{med} + z\ \vec{v}_{min}$. Thus the quadric surface is an ellipsoid oriented along the orthonormal eigenvectors, whereby the length of the i-th axis is given by $1/\sqrt{\lambda_i}$, i.e. the largest extent of the ellipsoid is determined by the smallest eigenvector and vice versa (this is clear also from eq. (2.61)). It is an hyperboloid for $\lambda_i < 0$.

Once we know the eigenvalues and eigenvectors, we also know the quadric surface of the inverse tensor $g = G^{-1}$ or $g^{mn}G_{nl} = \delta^m_l$, which is given by

$$g(V, V) = \frac{1}{\lambda_{max}}\ x^2 + \frac{1}{\lambda_{med}}\ y^2 + \frac{1}{\lambda_{min}}\ z^2 \tag{2.69}$$

with $V = x\ \flat\vec{v}_{max} + y\ \flat\vec{v}_{med} + z\ \flat\vec{v}_{min}$ a covector in the same coordinate system.

Isotropy Artifacts It is common use to visualize tensor fields via their eigenvalues and eigenvectors at each point. Using vector field visualization methods on fields of eigenvectors is a straightforward method, but when computing integral curves in an eigenvector "field" one needs to take care that the eigenvector equation determines its orientation, but not its sign, i.e. $-\vec{v}$ is the same solution as $\vec{v}$. Thus an eigenvector may flip from one point to the next by 180^o without indication of a vector field pathology. Existing vector field visualization methods can thus not be applied directly.

Actually, using eigenvectors turns out to be problematic at all, because they are not uniquely defined for an isotropic tensor, where all directions are equivalent. An eigenvector field's integral curve in the most trivial case of an isotropic tensor field would thus display unreal features, in the worst case it would result in a chaotic agglomeration of line segments. Weinstein et.al. [WKL99] therefore developed the idea of *tensorlines* (which will be addressed in more detail in 4.3.1) to address this problem by weighting the integration with the tensor field's isotropy, but this method involves some arbitrary parameters that allow some user to adjust the "stiffness" of the lines. For a metric tensor field, eigenvector integral lines don't have a sound physical meaning – more physical insight is depicted by geodesics. They are intrinsically free of isotropy artifacts, but impose the problem of appropriate initial conditions, because they are the solutions of a second-order differential equation and require a starting direction as well, whereas vector field integral lines just require seed points. This and the fact that computing geodesics is more effort (from the computational and from the implementation aspect) has presumably prevented their application in tensor field visualization beyond general relativity and differential geometry.

Principle Invariants

The coefficients of the characteristic equation (2.64)

$$\lambda^3 - I_1\lambda^2 + I_2\lambda - I_3 = 0$$

can be expressed by the eigenvalues

$$I_1 = \lambda_{max} + \lambda_{med} + \lambda_{min} \tag{2.70}$$

$$I_2 = \lambda_{max} \cdot \lambda_{med} + \lambda_{med} \cdot \lambda_{min} + \lambda_{min} \cdot \lambda_{max} \tag{2.71}$$

$$I_3 = \lambda_{max} \cdot \lambda_{med} \cdot \lambda_{min} \tag{2.72}$$

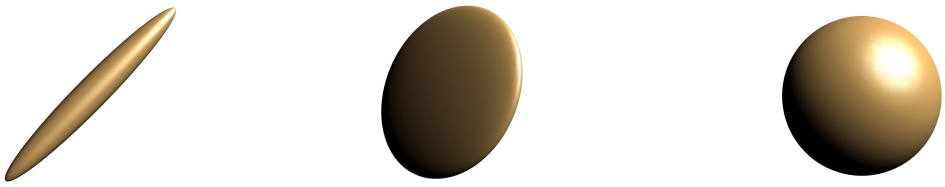
and are called the *principle invariants* of the tensor (see e.g. the review in Boring's thesis [Bor98]). They can be used to express tensor properties in a coordinate-independent way as well. As Zhukov et.al. [ZMB+03] pointed out, they can be computed without solving the cubic eigenvalue equation directly from the tensor components in an arbitrary coordinate system. E.g., I_1 is just the trace of the tensor.

Shape Classification

An intuitively useful classification of the *shape* of tensor ellipsoids was invented by Westin [WPG+97]. If we are primarily interested in the directions that are preferred in some tensor field, then we may disregard the trace and just look at the relationship of the tensor's eigenvalues. We may write a tensor G in matrix notation as a (same as (2.67)) linear combination of its eigenvalues/eigenvectors:

$$G = v_{max}\lambda_{max}v_{max}^T + v_{med}\lambda_{med}v_{med}^T + v_{min}\lambda_{min}v_{min}^T \quad .$$

Westin [WPG+97] based his tensor shape classification on the relationships of the eigenvalues of a



(a) Linear - one small eigenvalue

(b) Planar - two small eigenvalues

(c) Spherical - all eigenvalues similar

Figure 2.9: Classification of a tensor ellipsoid by its shape, according to the relationships of the eigenvalues.

diffusion tensor, stating that the linear case corresponds to one large, dominant eigenvalue here. For a metric tensor, the opposite interpretation is correct, since the length of the tensor ellipsoids (representing a light sphere) is proportional to the inverse square root of the eigenvalue: the tensor ellipsoid is a needle if one eigenvalue is very small as compared to the others - which is clear from (2.61). However, the classification scheme was adopted by the visualization community with regard to diffusion tensors and is now widely spread. Therefore we employ the inverse tensor's eigenvalues here for consistency with Westin's definitions:

- **Linear case**: $\lambda_{max} \gg \lambda_{med} \approx \lambda_{min}$, one eigenvalue of the inverse tensor is dominant. The tensor ellipsoid is mostly like a needle along the eigenvector with the dominant eigenvalue and can be approximated by

  $$G \approx v_{max}\lambda_{max}v_{max}^T =: \lambda_{max}G_l \quad .$$

  An example is a diagonal matrix with diagonal elements $(1, 0, 0)$. The inverse tensor reveals a disc (Fig. 2.9(b)).

- **Planar case**: $\lambda_{max} \approx \lambda_{med} \gg \lambda_{min}$, two eigenvalues of the inverse tensor are dominant. The tensor ellipsoid basically is a flat disc (Fig. 2.9(b)) that is spanned by the two eigenvectors with the dominant eigenvalues and can be approximated by

  $$G \approx \lambda_{max}\left(v_{max}v_{max}^T + v_{med}v_{med}^T\right) =: \lambda_{max}G_p \quad .$$

  An example is a diagonal matrix with diagonal elements $(1, 1, 0)$. The inverse tensor is a needle (Fig. 2.9(a)).

- **Spherical case**: $\lambda_{max} \approx \lambda_{med} \approx \lambda_{min}$, all eigenvalues are of approximately the same size. No direction is preferred (isotropic case), the tensor and inverse tensor ellipsoids are both spheres. The shape of the tensor ellipsoid is basically

  $$G \approx \lambda_{max}\left(v_{max}v_{max}^T + v_{med}v_{med}^T + v_{min}v_{min}^T\right) =: \lambda_{max}G_s \quad .$$

  An example is a diagonal matrix with diagonal elements $(1, 1, 1)$.

Generally, a tensor G will be a blend of these three components and can be written as a linear combination

$$G = (\lambda_{max} - \lambda_{med})G_l + (\lambda_{med} - \lambda_{min})G_p + \lambda_{min}G_s \quad .$$

Normalizing the blend factors by the trace $tr(G) = \lambda_{max} + \lambda_{med} + \lambda_{min}$ yields invariants that only depend on the shape independent from its size:

$$\begin{aligned} c_l &= \frac{\lambda_{max} - \lambda_{med}}{\lambda_{max} + \lambda_{med} + \lambda_{min}} \\ c_p &= \frac{2(\lambda_{med} - \lambda_{min})}{\lambda_{max} + \lambda_{med} + \lambda_{min}} \\ c_s &= \frac{3\lambda_{min}}{\lambda_{max} + \lambda_{med} + \lambda_{min}} \quad . \end{aligned}$$

The scaling numbers 2 and 3 are used such that each shape factor is in the interval $[0, 1]$. Other normalization choices are possible as well. The three shape factors obey the relationship

$$c_l + c_p + c_s = 1$$

and can thus be interpreted as barycentric coordinates within a triangle, as illustrated in Fig. 2.10. The spherical factor c_s is a direct measurement of the anisotropy.

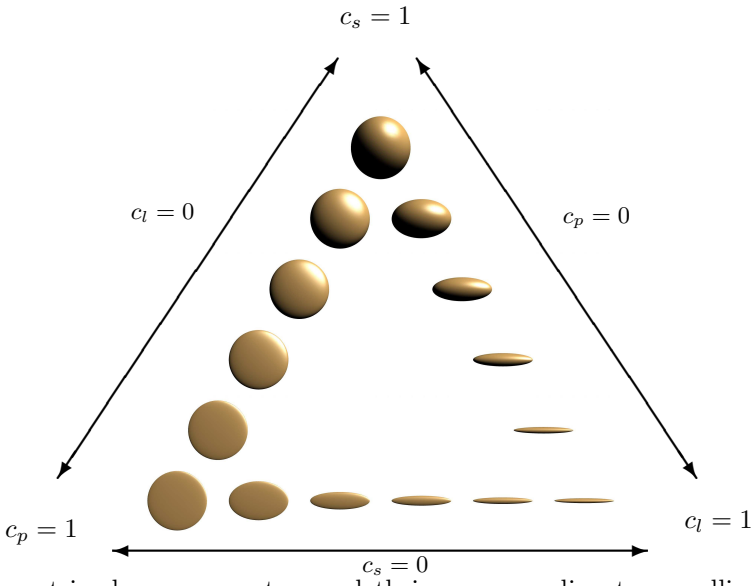


Figure 2.10: Barycentric shape parameters and their corresponding tensor ellipsoids. The left edge corresponds to zero linearity, bottom edge to zero sphericity and right edge to zero planarity. Note that shown tensor surfaces correspond to constant minimum eigenvalue (such that the maximum half-axis of the ellipsoid is constant), not constant trace.

Given the shape factors (two of them are sufficient) and the trace, the eigenvalues can be reconstructed:

$$\begin{aligned} \lambda_{min} &= c_s/3 \; tr(G) \\ \lambda_{med} &= (c_s/3 + c_p/2) \; tr(G) \\ \lambda_{max} &= (c_s/3 + c_p/2 + c_l) \; tr(G) \end{aligned} \tag{2.73}$$

The shape factors provide an alternative view of the eigenvalues, two of them plus the trace contain the same information. They may be analyzed on a tensor field as derived scalar fields; we call them the "Linearity","Planarity" and "Sphericity" (fields) of the tensor field. The shape factors of the inverse tensor are computed easily by exchanging $c_l \leftrightarrow c_p$ while keeping c_s.

Anisotropy Measurements

The eigenvalues are the basic quantities describing the shape of the quadric surface. If we are only interested in the directional information, it is useful to normalize the tensor shapes to the same maximum extent, thus forming a normalized eigenvalue triple:

$$\left(1, \frac{\lambda_{med}}{\lambda_{max}}, \frac{\lambda_{min}}{\lambda_{max}}\right)$$

As we have thrown away the trace information, the tensor shape has been reduced to two numbers (similar to the shape factors, that actually are just two numbers as well). To enhance the anisotropic properties, we might be interested inspecting just the deviator of the tensor field; the eigenvalues of an arbitrary tensor's deviator are given by:

$$(\lambda_{max} - \lambda_{min}, \lambda_{med} - \lambda_{min}, 0)$$

Both reductions can be combined to the eigenvalue triple of a tensor's normalized deviator:

$$\left(1, \underbrace{\frac{\lambda_{med} - \lambda_{min}}{\lambda_{max} - \lambda_{min}}}_{=:\epsilon}, 0 \right)$$

whereby $\epsilon \in [0, 1]$; ϵ is undefined in isotropic regions where all eigenvalues coincide. It is a reduction of the tensor shape to the single quantity ϵ by dropping trace and isotropy information. This single quantity can straightforwardly be used for indexing a colormap. Its known numerical range is useful as well. Other shape and anisotropy classifications are possible as well, as for instance discussed in Alexander et.al. [AHK+00].

Tensorfield Interpolation Aspects

When a tensor field is given on a discrete manifold, we sometimes wish to know the tensor field value between vertices. Thus the question arises how to perform correct interpolation, given two tensor matrices G_1 and G_2 on different points of the discretized manifold. The naive way is interpolating each tensor field component separately like a scalar field. However, this interpolation scheme would not sustain certain tensor field properties, like the linearity. Consider two linear tensors G_1, G_2 with different orientation and their arithmetic median $\frac{1}{2}(G_1 + G_2)$, :

$$\frac{1}{2}\begin{pmatrix} 1 & & \\ & 0 & \\ & & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & & \\ & 1 & \\ & & 0 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & & \\ & 1 & \\ & & 0 \end{pmatrix}$$

The resulting interpolated tensor of two linear tensors thus yields a planar tensor, which is not necessarily the expected result. An alternative interpolation scheme is thus to interpolate eigenvectors and eigenvalues instead of tensor components. This method is not equivalent to interpolating tensor components: if (v_1, λ_1) is an eigenvector/eigenvalue pair of G_1 and is (v_2, λ_2) is an eigenvector/eigenvalue pair of G_2, then $(v_1 + v2, \lambda_1 + \lambda_2)$ is not necessarily an eigenvector/eigenvalue of $G_1 + G_2$. We can see this algebraically by inserting the eigenvalue equations

$$\begin{aligned} G_1 v_1 &= \lambda_1 v_1 \eta \\ G_2 v_2 &= \lambda_2 v_2 \eta \end{aligned}$$

into the expected eigenvalue equation

$$(G_1 + G_2)(v_1 + v_2) = (\lambda_1 + \lambda_2)(v_1 + v_2)\eta$$

which is only true if

$$v_1(G_2 - \lambda_2\eta) + v_2(G_1 - \lambda_1)\eta = 0 \quad ,$$

but this is not necessarily the case. Thus for interpolating values among grid points, it makes a difference if the eigenvectors/eigenvalues are interpolated or the tensor field. Which interpolation scheme is preferable depends on the application and on the user expectations – usually the naive component-wise interpolation is the most appropriate one, even though it does not sustain certain tensor invariants, but the reduction of a tensor field to its invariants is information loss anyway.

Projections and Embeddings

Given two linearly independent tangential vectors $u, v \in T_p(M)$, the projection[4] g of a tensor $G \in T_p^*(M) \times T_p^*(M)$ in an higher dimensional space $dim(M) \geq 2$ onto the two dimensional plane that is span by these to vectors is given by

$$(g) = \begin{pmatrix} G(u,u) & G(u,v) \\ G(v,u) & G(v,v) \end{pmatrix} \quad . \tag{2.74}$$

[4] Note that the projection is different from the "image" or "shadow" of a metric ellipsoid casted onto a plane. This case will be discussed in 4.3.2.

If G is symmetric, g is symmetric as well. If G is a metric tensor field in the higher dimensional space, then g is the metric on an embedded surface. The embedding space of an arbitrary surface with a given metric is not necessarily flat or three-dimensional; nevertheless we can ask: Is there a surface with the same metric coefficients as a given surface which can be embedded within flat, three-dimensional space? Such a surface would give a good intuitive insight of a metric given on a surface. The general answer is no: An embedding of arbitrary two-dimensional metrices in $\mathbb{R}^3$ is not possible. Embeddings exist only locally and global embeddings can only be found in rare cases.

Various theorems describe the existence and properties of embeddings: If we are just interested in topologically equivalent embeddings (def. 14), then we can make use of *Whitney's smooth embedding theorem* [Whi32], which states that every n-dimensional manifold, can be embedded (topologically equivalent, but not yet isometric) in an $2n+1$-dimensional Euclidean space. Here, we are interested in the rather strict case of isometric embeddings (def. 12). The *Nash embedding theorem* [Nas65] states that every Riemannian n-dimensional manifold can be isometrically embedded in a Euclidean space $\mathbb{R}^{2n+1}$. In the case of a spacetime, we require *Clarke's embedding theorem* [Cla70] which states that any pseudo-Riemannian manifold can be embedded isometrically in some $\mathbb{R}^n$ with constant metric tensor, where n is at most 90 and the number of timelike dimensions is at most 3. Thus a spacetime may need up to 87 spacelike and 3 timelike dimensions. *Campbell's embedding theorem* states that any n-dimensional Riemannian manifold can be *locally* embedded into an $(n+1)$-dimensional manifold with Ricci curvature $R_{ab} = 0$. A similar version of the theorem for a pseudo-Riemannian manifold states that any n-dimensional pseudo-Riemannian manifold can be locally and isometrically embedded in an $n(n+1)/2$-dimensional pseudo-Euclidean space.

Given a two-dimensional manifold and a metric g on it with coordinates u, v, i.e.

$$ds^2 = g_{uu}\ du^2 + 2g_{uv}\ dudv + g_{vv}\ dv^2 \quad ,$$

we seek a surface $\Sigma(u,v) : \mathbb{R}^2 \to M$ such that

$$g(\partial_u, \partial_u) = g_{uu} = G(\Sigma_{,u}\,, \Sigma_{,u}\,) \tag{2.75}$$
$$g(\partial_u, \partial_v) = g_{uv} = G(\Sigma_{,u}\,, \Sigma_{,v}\,) \tag{2.76}$$
$$g(\partial_v, \partial_v) = g_{vv} = G(\Sigma_{,v}\,, \Sigma_{,v}\,) \tag{2.77}$$

For an embedding into flat, Euclidean space $M = \mathbb{R}^3$ with metric $G = \eta$ and the abbreviations $x := \Sigma^x,\ y := \Sigma^y,\ z := \Sigma^z$, i.e. $\Sigma(u,v) = \big(x(u,v), y(u,v), z(u,v)\big)$, we get

$$g_{uu} = {x_{,u}}^2 + {y_{,u}}^2 + {z_{,u}}^2 \tag{2.78}$$
$$g_{uv} = x_{,u}x_{,v} + y_{,u}y_{,v} + z_{,u}z_{,v} \tag{2.79}$$
$$g_{vv} = {x_{,v}}^2 + {y_{,v}}^2 + {z_{,v}}^2 \tag{2.80}$$

Various approaches exist to solve this system of partial differential equations. Romano and Price[RP95] used the Darboux equation to compute a local embedding in the misner spacetime, which describes the initial data of two black holes colliding head-on. Nollert and Herold [NH96] were starting from a triangular surface and adjusted the distance relationships among each edge to match those of the given metric; their algorithm works well in principle, but suffers from ambiguities since only squared distances are available from the metric, such that single triangle patches "flip sign", revealing a surface with the same metric, but unpleasant appearance. Ingrid Hotz [Hot02] solves this problem by also taking into account the gaussian curvature of the surface and constructs surfaces from a given initial point by building rings of triangles by "outspiralling" from a seed point. Another approach aiming towards a global approximative embedding instead of a locally exact one was carried out by Bondarescu, Alcubierre and Seidel [BAS02]: They describe a surface by a series expansion of basis functions and use a minimization algorithm for the coefficients until the surface approximates the same metric coefficients in flat space as the given input metric. In the published application apparent horizons from numerical simulations of colliding black holes were investigated, where the spherical topology is already known, so spherical harmonics were used as basis functions. However, other sets of basis functions like toroidal harmonics, were also possible in principle to cover alternative topologies. While the numerical computations for this approach have been performed in CACTUS [MPIfGP03], the according visualizations have been based on the fiber bundle visualization. The formulation of the underlying data structures for visualization purposes are described in A.1.3.

Example: Schwarzschild Metric Let us carry out the work flow path of finding an embedding surface in the case of the Schwarzschild metric (2.52) . We want to investigate the appearance of the spacetime for a static observer, i.e. one that hovers at constant distance to the black hole. As this spacetime is spherically symmetric, we limit our inspection to the equatorial plane $\vartheta = \pi/2$. Then the spatial metric becomes

$$ds^2 = \frac{1}{1-2m/r}dr^2 + r^2 d\varphi^2 \quad .$$

For the ease of computation we use cylindrical coordinates and model the embedding surface as $\Sigma(r,\varphi) = (r,\varphi,z(r,\varphi))$. We get $\Sigma_{,r} = \partial_r + z_{,r}\,\partial_z$ and $\Sigma_{,\varphi} = \partial_\varphi + z_{,\varphi}\,\partial_z$. The Euclidean metric is of the form

$$ds^2 = dr^2 + r^2 d\varphi^2 + dz^2 \qquad ,$$

in these coordinates, i.e. $G_{rr} = 1$, $G_{\varphi\varphi} = r^2$ and $G_{zz} = 1$. The embedding equations (2.75), (2.76) and (2.77) then become

$$\frac{1}{1-2m/r} = g_{rr} = G(\Sigma_{,r}\,,\Sigma_{,r}\,) = G(\partial_r + z_{,r}\,\partial_z, \partial_r + z_{,r}\,\partial_z) = 1 + {z_{,r}}^2 \tag{2.81}$$

$$0 = g_{r\varphi} = G(\Sigma_{,r}\,,\Sigma_{,\varphi}\,) = G(\partial_r + z_{,r}\,\partial_z, \partial_\varphi + z_{,\varphi}\,\partial_z) = z_{,r}\,z_{,\varphi} \tag{2.82}$$

$$r^2 = g_{\varphi\varphi} = G(\Sigma_{,\varphi}\,,\Sigma_{,\varphi}\,) = G(\partial_\varphi + z_{,\varphi}\,\partial_z, \partial_\varphi + z_{,\varphi}\,\partial_z) = r^2 + {z_{,\varphi}}^2 \tag{2.83}$$

We see that equations (2.82) and (2.83) are fulfilled if z does not depend on φ, leaving a single differential equation for $z(r)$:

$$\frac{1}{1-2m/r} = 1 + {z_{,r}}^2$$

We can now solve this equation for z:

$$z = \int \sqrt{\frac{1}{1-2m/r} - 1}\; dr = \int \sqrt{\frac{1}{1-2m/r} - \frac{1-2m/r}{1-2m/r}}\; dr =$$

$$= \int \sqrt{\frac{2m/r}{1-2m/r}}\; dr = \int \sqrt{\frac{1}{r/2m-1}}\; dr \tag{2.84}$$

This integral can be solved by substituting $\varrho := r/2m - 1$, $dr := 2m\; d\varrho$:

$$z = 2m \int \sqrt{\frac{1}{\varrho}}\; d\varrho = 2m \int \varrho^{-1/2}\; d\varrho = 2m\frac{1}{2}\varrho^{1/2} = m\sqrt{\varrho} = m\sqrt{r/2m-1}$$

Thus, the embedding surface of a Schwarzschild black hole is a parabola rotated around the z-axis and opened along the equatorial plane. It is defined only for $r \geq 2m$. At $r = 2m$ its slope is infinite, indicating the infinite spatial distance to the event horizon as experienced by a static observer (g_{rr} becomes infinite). (in other words: static observers cannot exist at the event horizon itself) whereas the circumference of a black hole is simply $2\pi r$.

A common visualization of this behavior is a throat as in fig. 2.11: traveling around its circumference is easy, but radial traversal is hard and even impossible once the slope gets vertical. Depicting this behavior in full space rather than on a two-dimensional slice by an embedding would require a six-dimensional space (proof not given here), which is beyond current hardware graphics and mental capabilities. However, the tensor field visualization methods that are explored in chapter 4 are able to depict this behavior even in flat rendering space (see fig. 4.26 in particular).

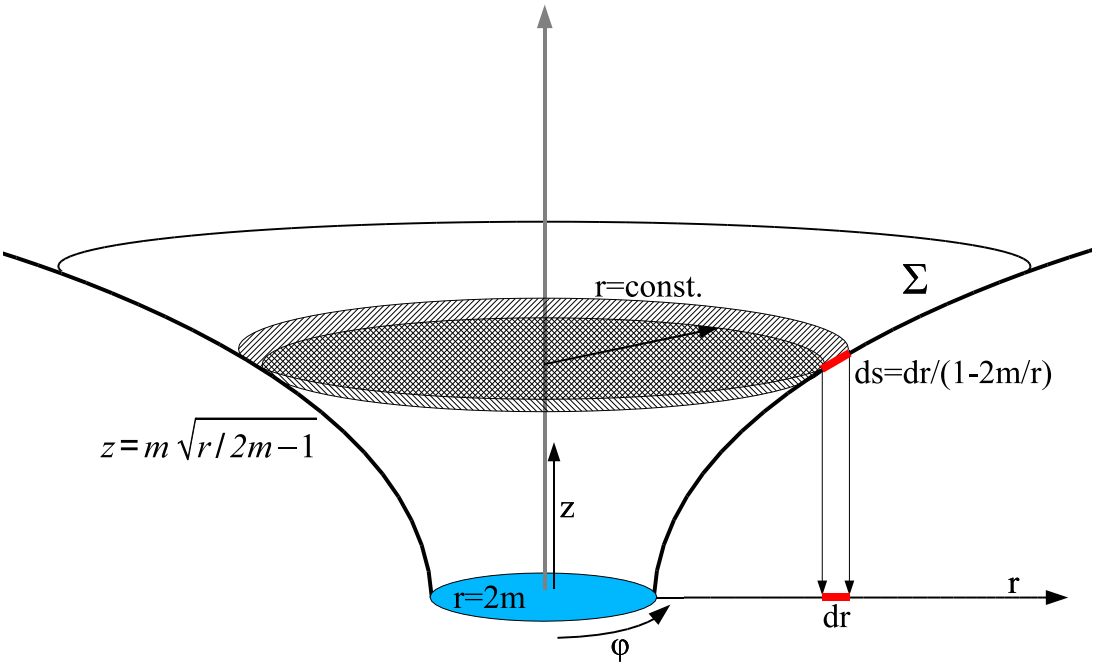


Figure 2.11: Embedding surface of the equatorial plane of the Schwarzschild metric. The metric ds is directly visible as the distances on the surface Σ. The relation ds/dr of the metric distance differences ds to radial distance differences dr becomes infinite at $r = 2m$.

Chapter 3

Data Model Design and Architecture

When performing numerical computations on some spacetime, one needs to discretize the underlying physical manifold into a set of points. Many such discretization schemes exist, each of them providing advantages and disadvantages. No "best" discretization scheme exists to work optimally under all circumstances and environments. Nevertheless, when a certain problem is to be solved, the user should ideally only be bothered with the problem description on an abstract level an abstract level, and the effective discretization scheme is chosen by some implementation. It might even change during some computation, e.g. starting computations on a uniform grid, then continuing on a tetrahedral grid, and go back to some adaptive mesh refinement grid on regular cells, as required by the numerical algorithms automatically.

This behavior is not commonly in use today because these different discretization schemes have different properties and thus also different ways to describe and handle their data (a problem equivalent to data exchange among different applications), although they all have the same roots. Following the pioneering ideas of Butler [BB92], to tackle this problem it is therefore required to go back to these roots, the abstract mathematical formulations of data in space and time. It must thus be our goal to derive a data model which is able to

- abstract the geometrical description of spatial objects from their numerical representation in a specific coordinate system
- abstract the physical computation domain from its underlying discretization scheme.

If both abstractions can be achieved, then there exists an abstract layer on top of the data but independent from their actual representation. It allows to formulate tasks in a problem-oriented way and leaves some implementation the freedom to chose the "best" method to solve them. Some development into this direction was done in the Sophus library [MKH95], but not in the context of computer graphics or as basis of a data model.

The data model presented here provides such abstraction mechanisms, and allows the formulation of grid-independent algorithms in the same spirit as mathematical expressions can be formulated in a coordinate-free manner using the form calculus. This does not necessarily mean that *any* existing algorithm can be formulated grid-independent. Like coordinate expressions that are only valid in a certain chart, also grid-specific algorithms can only be formulated for a certain grid with a-priori known properties. It depends on the respective case if some algorithm can be formulated grid- and coordinate-independent; however, the presented fiber bundle data model provides a "language" to describe the requirements of an algorithm on the final application level. In this view, the various hierarchy levels and components of this data model play the role of "semantic words".

In particular when aiming at the visualization of tensor fields, one clearly would not like to rewrite any algorithm and implementation for any data type. In contrast to scalar and also vector field visualization, operating on tensor fields requires much more work just within the tangential space. Typical examples are the computation of eigenvectors, or derived scalar fields like the "linearity" as mentioned before when computing shape factors. All these operations as mentioned in 2.3.2 happen only in the tangential space $T_p(M)$, so it is just natural if not required to seek for a common interface of such operations independent of the underlying manifold M and its discretization schemes.

The presentation also includes descriptions of the prototype implementation, but one should keep in mind that this mapping of the presented concepts to the shown C++ API is not the only one, e.g. some C++ object might be just an integer identifier in some HDF5 implementation.

As mentioned by John Shalf [Sha03], an important issue for the practical acceptance of a data model is also that it can be mapped bijectively to simpler models and their correspondingly simpler file format. This is can be achieved by defining certain states of the more general model as default values, such that these values are assumed implicit when interpreting data from a simple model. In particular they don't need to be specified explicitly when writing some file format. This ensures that the minimal set of required information is contained in some file, with successively adding more information content only if this information is actually required. Nevertheless, this mapping to "simpler appearance" is mainly an implementation effort and does not touch the design of the full data model by itself.

3.1 Fiber Bundle Data Hierarchy

If data have the mathematical structure of a fiber bundle – it was stated by Butler [BP89] that practically all scientific data are of this kind –, it can be organized as a cartesian product $B \times F$ of a base space B with a fiber F (see def. 50). The fiber might optionally carry vector space structure. The base space is described via its topological properties and algebraic invariants independently from its fiber data. Sections of the fiber bundle (def. 54) are called "fields"[1], and operations on the fields can be performed independently from the base space. Although not all algorithms can be separated into pure base space and fiber operations, many of them can be classified this way. This explains the strength of the fiber bundle data model. Its properties are especially of worth for the purpose of tensor field visualizations, but its capabilities go far beyond this particular application.

3.1.1 Hierarchy Levels

The data structure is formed by a tree-like structure with cross-references. It is constructed from associative arrays, which allow to map certain semantic objects to the next hierarchy level that ultimatively provide the actual data sets on the final leaves. This data structure together with a set of related functions that are parameterized by these semantic objects form the *data model*. All data which are involved in such operations, are accessible from some "trunk" object, which is called the *Bundle* object.

Structure of Bundle Objects

The base space might refer to data within a "physical space" by definition of def. 39. If so, it has a time orientation and the dimension of the base space is at least one (e.g. a time series of a data value). The most natural choice is thus to model the base space as a cartesian product $(\mathbb{R} \times \Sigma, \uparrow)$. Another possible choice were e.g. $(\mathbb{S}^1 \times \Sigma, \uparrow)$ to handle periodic events. Hereby Σ is the space describing the base space at some instant of time. Via discretization of $\mathbb{R}$ we get a foliation of the physical space into an *ordered set of slices* (the term "slice" is hereby borrowed from the $3+1$ decomposition in numerical relativity 2.2.3). The ordering implements the time-orientation (def. 37) of the manifold. This is the uppermost level structure of this fiber bundle data organization model. Alternative slicings of a given spacetime are

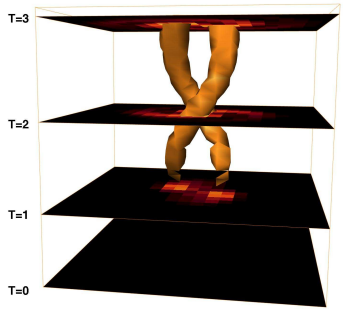


Figure 3.1: The evolution of two orbiting bodies emitting some waves, time is shown as 'up' coordinate. The entire spacetime is sliced into spatial subspaces. This slicing determines the outermost hierarchy level of the fiber bundle data structure described here.

possible. Usually, this is required only in the context of general relativity where there is no unique time

[1] Note that this definition is not identical to the algebraic definition of the term "field".

coordinate and various time coordinates may co-exist. A certain `Bundle` object needs to support only one time slicing, which is sufficient for non-relativistic cases. Another time slicing can be computed into another `Bundle` object, that provides the same functionality as the original `Bundle` and may share data for efficiency. Alternative time slicings may also be used to impose a different ordering of data, e.g. to

Figure 3.2: A different time slicing of some evolution as in fig.3.1. It is handled from another Bundle Object that may share the same data.

order some data according to some other quantity instead of time ("at which time does the pressure of some gas reach a certain threshold" instead of "what is the pressure of some gas at a certain time").

Limitations and possible Extensions

- Closed timelike lines (def. 2.1.2) or any spacetimes with causality violations are problematic to implement in this framework. Such cases cannot occur in numerical evolutions of Cauchy surfaces, so handling them is not a burning issue for practical purposes.
- Data without time-orientation can not be handled in this concept; however, they can be put into a certain time step and handled as completely spatial data space.
- In an extended version (which was not implemented here), we would think of a fiber bundle over an parameter space $P \times \Sigma$. Then, the timelike case described above were just the special case of a one-dimensional parameter space $\mathbb{R} \times \Sigma$, but a multidimensional discretized sparsely covered parameter space $\mathbb{R}^n \times \Sigma$ is reasonable as well. It were useful for parameter studies, and the case of time-independent data sets is formulated as a fiber bundle over a zero-dimensional parameter space $\mathbb{R}^0 \times \Sigma$.

C++ API In describing the C++ implementation, we defer the definitions of the respective next level. For a Bundle object, the next level is the "Slice" level. The structure of the outermost level can be formulated using an associative array mapping elements from the parameter space to Slice objects. For a one-dimensional parameter space, this can be done using the "map" template from the Standard Template Library (STL), as the parameter space will usually not be covered uniformly:

```
struct Bundle : map<double, Slice>
{};
```

A Slice object is accessed from a Bundle Object via some floating-point parameter that specifies the physical time:

```
double time = 7.85;
Bundle myBundle;
Slice &mySlice = myBundle[time];
```

Usually, only one Bundle Object is required per application, but multiple ones may coexist and share data. Here, we opt to the concept of "indexing by physical time" instead of "indexing by the notion of timesteps". At first glimpse, this may sound troublesome, because timesteps as output by some simulation process are given as exact integers, while floating point values always might lead to rounding errors. However, the motivation is that one bundle object should not only cover a data from a single simulation, but possibly from many simulations. These will not necessarily provide the same mappings between timesteps and physical time. Moreover, the physical time difference among subsequent timesteps

might not be constant. However, a correct visualization of the physical behaviour demands using physical time and not exposing the artifacts of time discretization to the user. Although a "quick and easy" implementation might still be based on timesteps, and advanced one needs to be able to handle indexing by physical time correctly, including performing possible interpolation of intermediate timelike information.

Structure of Slice Objects

The physical space covered by a Bundle may contain various geometrical entities, like surfaces, some bounded data volume, a set of particles and so on. By the timelike slicing of the base space, these geometrical entities are split into multiple instances. Each of these is related to a subset of the spatial base space Σ at various time slices (or parameter slices, in the generalized version). We call such a (spatial) subset describing a geometrical entity a *Grid Object*.

To identify evolving Grids over various time steps, they must be identifiable through different time slices. This is done by some associative map on each time slice, which maps some "*Grid Identifier*" to the respective Grid Objects. The easiest implementation of a "`Grid` Identifier" is a textual description. An advanced implementation might use a structure which allows to formulate semantic relationships among grid objects as well, like "this surface is an isosurface of the volumetric data set xyz". The only requirement from the fiber bundle data model is an equality relationship among `Grid` Identifiers which allows to trace a certain `Grid` within many slices.

C++ API As with the Bundle structure, we may employ an STL map for associating Grid Identifiers to Grid objects within a Slice object:

```
typedef string    GridIdentifier ;

struct   Slice  :  map<GridIdentifier,  Grid>
{};
```

As mentioned, the GridIdentifier could be a data structure carrying semantical information beyond the fiber bundle data model and is typedef'd to a string for demonstration purposes here. A `Grid` objectcan be can be assigned to and retrieved from a Slice Object via some Grid Identifier:

```
Grid  myGrid;
        mySlice["geometricobject"] = myGrid;
Grid&GeometryGrid = mySlice["geometricobject"];
```

An actual implementation will use reference-counter pointers to `Grid` objectinstead of `Grid` objects.

Structure of Grid Objects

A `Grid` Object is a collection of decompositions (topological spaces with CW-complexes, def. 60) and houses many datasets, which are called *Grid components* or shortly components (but it should not be confused with the different notion of the "components of a tensor field"). The components of a `Grid` are categorized by common properties, especially their relation to the k-cells of the underlying decomposition (i.e. k-skeletons of the triangulation according to def. 60). This categorization leads to the subsequent grouping structure. Examples of such categories are the points (vertices) of the Grids, its cells or its cell complexes. Examples of Grid components are the data sets describing e.g. the coordinate values of the vertices in a certain coordinate system, relationships among points and cells. The `Grid`object also is the place to store an *atlas*. This "`Grid` Atlas" is a set of chart objects with transformation rules among each other; it will be explained in 3.2.3.

The entity of all Grid components determines the topological properties (def. 2) of a Grid object. I.e. there is no single enumeration value determining the "grid type", but the full information about the properties of a Grid object is not known until all Grid components have been inspected and interpreted. This model allows a smooth transition among similar grid types, as two grid types might just differ by a single grid component whereas all other components are identical (and could even be shared). It also allows to construct many different of grid types by combinations of relatively simple grid components as "building blocks". The determination of the full properties of a grid type is rarely required, as for a certain application only certain properties of the grid are required. Property-based operations on a Grid object thus automatically lead to synergy effects among similar grid types. They allow to find and make use common properties that would have hardly be found when using the enumeration-based method, which is simpler to implement in a short-term approach and which is the method usually found in data handling and visualization environments.

All grid components related to a certain type of k-cells (e.g. equal dimensionality and same neighborhood information) are grouped together into "*Topology Objects*", the fourth level in the fiber bundle hierarchy after the Bundle, Slice and Grid levels. Topology objects have an Index Space associated with it and its data sets are organized in compatible arrays, a concept defined as follows.

Concept of Arrays and Index Spaces A Topology object imposes certain restrictions on the data set it contains, in that all of these data sets have to share the same index space (definition following below). They are discrete sets and will be described by "arrays", a term which we need to define yet:

Definition 64 *An **array** A is a "discrete map", i.e. a map from some subspace $I \subset \mathbb{N}$ to some space D. It associates some data element $A[i] \in D$ to an integer value $i \in I$ (the "index"):*

$$\begin{array}{ccc} A : I & \longrightarrow & D \\ i & \longmapsto & A[i] \end{array} \tag{3.1}$$

The notation $A[i]$ is used here to indicate indexing with integer arguments in accordance with the array indexing syntax of native C++ arrays.

Definition 65 *The set of all valid indices I of an array A is called the Index Space I_A of the array A.*

In this definition, no relationships among the elements in the Index Space is imposed, in particular no neighborhood information is implied, i.e. the Index Space does not necessarily need to be a connected [see def. 4] subspace of $\mathbb{N}$). This definition is in contrast to e.g. the array definition in POOMA[Cod], where an array object also implicitly contains neighborhood information about its indices. An array in POOMA is thus considered a fiber bundle over a topological space by itself, but in the context of this model it is just a discrete map. It can be seen as fibers of the Index space, which does not need to be connected.

Definition 66 *The **size** of an array is the number of elements of its Index Space.*

This definition is in accordance with the `size()` function within the STL[S+99]. It is *not* directly related the number of bytes occupied in computer memory, because an array with non-zero "size" might even use zero bytes in computer memory, when it is given implicitly or implemented procedurally and even compile-time evaluated (see 3.1.2 for examples).

Definition 67 *Two arrays A, B are **compatible** if their index space is identical, each index which can be used for array A can be used as index for array B as well:*

$$A, B \quad \textit{compatible} \quad \Longleftrightarrow \qquad I_A = I_B$$

Obviously, all compatible arrays have the same size (number of elements). Having the same size is a necessary, but not sufficient condition for two arrays to be compatible. Further constraints determining compatibility are given in the description of the next level. All arrays beyond the fourth fiber bundle hierarchy level, the Topology level, have to be compatible.

C++ API for Arrays and Index Spaces The concept of compatibility between native C++ arrays not supported by this programming language. Here, array elements are accessed using some integer data type, but there is no check whether the specified integer is valid for this type of array, neither numerically ("range check") nor conceptually (using an index for cells for an array providing per-vertex information is an error). C++ allows the definition of user-defined objects which appear like native C++ arrays via operator overloading. This functionality is frequently used to define objects that implement the missing range check functionality and still behave mostly like native C++ arrays. To implement the concept of index spaces, we additionally intend to verify the correct index depth on each element access. In contrast to the numerical range check, this information is available at compile-time and therefore should not impact runtime behavior. It is thus required to attach additional information to mere numbers, thus allowing them to carry information about their respective index space. This is done via *integer proxy types*, which are C++ class objects behaving like integers (especially regarding performance and memory requirements), but actually are classes. The definition of an array then includes the specification of the allowed index space and only appropriate integer proxy types may be used to access the elements of the array. This "check for correct index space" is based on type information and is thus performed at compile-time only and does not impact execution speed. It nevertheless improves programming security.

Using C++ template notation, an array is written as the type `Array<Index, type>`, whereby `Index` stands for the array's index space I_A. The actual implementation is untouched by this notation, in particular the storage method of the array elements: they can reside in a contiguous memory block in local or shared memory, as sequence of partially loaded data, which is read from disk on demand, or as list of elements, etc.

The concept of an array is reflected by a class hierarchy of three levels:

1. an abstract base class which provides mere runtime `size()` and compile time index space information

2. a derived abstract class which provides procedural read-only element access

3. a concrete class which handles array element generation, either by storing elements in compute memory or creating element values procedurally

An exemplary C++ class hierarchy implementation is like this:

```
template <class Index> class IndexedArrayBase
{
public: virtual Index size() const = 0;
};

template <class Value, class Index>
        class ArrayBase : public IndexedArrayBase<Index>
{
public: virtual const Index&operator[](const Index&i) const = 0;
};

template <class Value, class Index>
        class Array : public ArrayBase<Value,Index>
{
        Value*values;
        Index n;
public:
        Index size() const { return n; }
        const Index&operator[](const Index&i) const
        {       return values[i];       }
};
```

In this scheme, the purpose of the `IndexedArrayBase` class is to allow storage of untyped array pointers that allow runtime-type conversion via C++ runtime type information (`dynamic_cast`). The purpose of the `ArrayBase` class is to allow procedural access to array elements. The concrete memory layout or value creation algorithm is not specified for this interface. The `Array` class is a possible implementation of a simple class that actually stores data values in memory. In contrast to a procedural class, it also allows writable data access (member functions not shown).

In the fiber bundle data structure, only pointers to `IndexedArrayBase` objects are stored. All such array pointers within one `Topology` object are of the same type, thus ensuring index space consistency and array compatibility. If some algorithm requires read-only data access, it will `dynamic_cast` such pointers to an `ArrayBase` object. If it requires write-able data access or wants to circumvent the (slow!) virtual element access functions by direct data access, it would `dynamic_cast` the pointer to an `Array` object. For performance reasons, it is recommendable to instantiate array algorithms over a list of known `Array` types. The associated `ArrayBase` objects should only be used as "worst case" for `Array` types which are introduced just at runtime (when re-compilation of the array-specific algorithm is not possible or not desirable). By this approach, algorithms gain speed by the cost of increased compile times and increased binary execution code. For n arrays and p array types involved in some algorithm, this results in an n^p compile-time overhead. Clearly, using this technology can easily reach the limits of available compiler technology and the code developer's patience. On the other side, each instantiated algorithm is capable of using the various features provided by an array implementation to its full extend (e.g. the computation of values on-demand) and thus optimal performance is achieved for numerous special cases and combinations of them. This is particularly important when describing certain properties of a `Grid` object via procedural arrays, like the coordinates of a uniform grid or the cell type of a regular grid.

LIMITATIONS

- The array interface as shown does not return a value, but a reference to some variable containing the value. This results in better performance for data types which are more expensive to copy in memory than a pointer. This is especially true for tensor types which consist of many float values. However, for procedurally computed data values the result has to be stored in a persistent memory location. If the same memory location is used for each call to the element access function, then the array element access will not be thread-safe and not reentrant.
- Arrays with varying return types (e.g. returning sets with varying cardinality) impose implementation problems:
  - the required memory resources can no longer be computed from the array size and array type alone,
  - algorithms can no longer execute identical code per element but must explicitly inspect the element type,
  - type information must be returned in addition to the raw data.

  It is hard to find an efficient solution for this problem (e.g., returning an `array` is possible, but inefficient). This is a major hurdle in implementing sheaves with stalks (see def. 55) instead of a fiber bundle with fiber objects.
- The mentioned n^p compile-time overhead impacts code development productivity. It requires to limit the number of possible array implementations to the most important cases and thus practically disables a fully generic approach, including user-defined array implementations.

C++ API for `Grid` objects The most frequently used `Topology` objects of a `Grid`are mandatory, but others (e.g. for usage within Adaptive Mesh Refinement) may be added dynamically.

```
struct  Grid
{
        typedef string    TopologyIdentifier ;

        list <Chart>     Atlas;

        Topology<IndexSpace<0> >              Points;
        Topology<IndexSpace<1> >              Connectivity , Edges, Faces;
        map<TopologyIdentifier, TopologyBase> Topologies;
};
```

`Topology` objects are modeled as template instantiations over an index space. Their internal structure will be explained in the next section. In addition to `Topology` objects, a `Grid` object houses a set of Chart objects, the "Atlas" (see 3.2.3), which specifies the "potential" of a `Grid` objectrather than its actual state: It tells which coordinate representations are possible for the data stored on the `Grid` object, but these coordinate representations do not necessarily exist yet.

Structure of Topology Objects

Each `Topology` Object contains an array describing the neighborhood information for each element, i.e. the "`Neighborhood`" array is a mandatory `Grid` component stored at the `Topology` level. Each element of the neighborhood array specifies an open set around a topological point. Here the term "point" does not only refer to the vertices of a `Grid` object, but also to e.g. the k-cells of a `Grid` object, as the set of all k-cells of a `Grid` object form a (discrete) topological space by itself (but not within the physical space, where this set is just a k-skeleton). Consequently, the *dimensionality* k is useful for classifiying `Topology` objects.

The cell dimensionality is not a sufficient description, and beyond this purely mathematical property we may additionally categorize `Topology` objects with respect to their relationship to the physical space. Subsets of the physical space are denoted as "primary". Sets which describe properties of primary sets are "secondary" and further on. This semantic information (it gives information about the purpose of a `Topology` object) is called the *index depth*:

Definition 68 *The **index depth** of an index space is the number of dereferencing operations that is required to reach points within the physical space from some Topology Object on this index space.*

Primary sets (i.e. information on vertices) get assigned an index depth of 0, secondary sets (information on k-cells)) have depth 1 etc. A set of index depth $n+1$ is built by groupings of elements taken from a set of depth n. The index depth is a property of an index similar to the type of a pointer types in C++ . This analogy is demonstrated by this code fragment:

```
double x;          // primary data, index depth 0
double *x;         // secondary data, index depth 1
double **x;        // index depth 2
```

An index can be interpreted as a "relative pointer" - it does not point to an absolute location, but to a location within arrays. A certain index is valid for all arrays within a `Topology` object (in contrast, a pointer is only valid within a certain array). The index depth corresponds to the type information of a pointer. An example of various `Topology` objects with different various index depth and dimensionality is given in table 3.1.

	index depth	dimensionality
Vertices	0	0
Edges	1	1
Faces	1	2
Cells	1	3
Collection of vertices	1	0
Path of connected edges	2	1
Surface built from faces	2	2
Cell complex	2	3
Set of cell complexes	3	3

Table 3.1: Various `Topology` objects that might reside on a three-dimensional grid.

A third essential categorization number is the *refinement level*. It allows to specify two or more `Topology` objects for topological elements of the same dimensionality and same index depth, e.g. the vertices of a grid, with different sizes of the corresponding index space. In special cases, the refinement level might be multidimensional by itself to formulate e.g. refinement in x-direction only vs. refinement in y-direction only. However, this case is problematic as it does no longer provide an ordering relationship among refinements.

These three categorization criteria "index depth", "dimensionality" and "refinement" do not yet provide a complete description of `Topology` object. For instance the operation of "subsampling", which selects each nth element from within a `Topology` object, increases the refinement factor, but leaves index depth and dimensionality unchanged. In contrast, the operation of "downsampling", which averages n elements respectively from within `Topology` object, sustains dimensionality but increases the index depth. However, both operations are not commutative: first subsampling, then downsampling leads to another data set than first downsampling, then subsampling. The "*Topology Identifier*" per `Grid` object mentioned in the former section needs to reflect the categorization scheme of the associated `Topology` object. In particular it must allow freedom to distinguish among cases like the subsampled/downsampled and downsampled/subsampled ones. It is required to carry the semantic information about the history of the `Topology` object and to encode a rule on how to reproduce it. It is not yet clear how large the parameter space needs to be for a sufficient description, that might be a topic of further research beyond this current data model.

`Topology` objects are unique per `Grid` Object, but different `Grid` Objects may carry comparable `Topology` Objects. Identity relationships among `Topology` Objects on `Grid` Objects allow to classify grid types and operations on them. If two `Topology` Objects on different `Grid` Objects are comparable or even identical, then certain algorithms (or better: components of algorithms, as will be described in 3.2.2) can be performed on both `Grid`objects without changes.

A `Topology` object carries various data fields on it. These data should ideally be described in a coordinate-free manner. However, numerical computations can only be performed in a certain chart. The same data field may be represented in various ways, e.g. in different coordinate systems, like the components of a vector field in cartesian and polar coordinates. A collection of representations allows to select the "best" suitable one depending on the respective requirements. Data fields within a `Topology` object are grouped into "`Representation` objects", with the exception of the topological neighborhood information, which is an "intrinsic" (i.e. representation-independent) property of the `Topology` object itself. `Representation` objects are selected from a `Topology` object via `Representation` Identifiers,

which is a reference to a chart (a coordinate system) or another `Topology` object. This object is called the "*Representer*" of a `Topology` object within a `Grid`.

C++ API The current implementation uses a textual description for the Topology identifier to encode the three numerical categorization parameters and additional more information. The `Topology` object is implemented as instantiation of a template class over an index space. This index spaces defines the possible arrays that may be used for storing data fields within the `Topology` object. `Topology` objects have a similar purpose as charts, they can both be used to select `Representation` objects. While `Topology` objects provide neighbourhood information and multiple data representations related to % or `Topology` objects, % objects provide transformation rules (which will discussed later in 3.2.3). They are thus derived from the same base class:

```
struct TopologyBase
{};

template <class Chart>
struct TransformationBase
{};

// See 3.2.3 for definition.
template <class SourceChart, class DestChart>
struct Transformation : TransformationBase<SourceChart>;

template <int dims>
struct Chart : TopologyBase
{
        map<ChartBase*, TransformationBase<Chart>*> Transformations;
};

template <class IndexSpace>
struct Topology : TopologyBase
{
        ArrayBase< IndexedArrayBase<IndexSpace>, IndexSpace> Neighbourhood;

        map<TopologyBase, RepresentationBase<IndexSpace> > Representations;
};
```

Abstract operations are possible on `ChartBase` objects, but operations involving the detailed layout of a `Chart` or `Topology` require specific code, usually formulated via instantiations of generic template programs. `RepresentationBase` is an abstract base class for all `Representation` objects for the same index space.

Structure of Representation Objects

`Representation` objects contain a mandatory `Grid` component called "Positions". It stores the generalized "coordinate" locations of the points of a `Topology` object. In the case of a vertex related `Topology` object (i.e. index depth 0), the respective representations are representations in different coordinate systems. The "representer" objects thus usually are charts, but the concept of using multiple representations will also be applied to `Topology` objects of higher index depth. For `Topology` objects describing k-cells of a grid the "Positions" component stores the incidence relationships (def. 2.2.1) to other `Topology` objects describing cells of different dimensionality or refinement. The adjacency relationship is stored in the `Topology` object's neighborhood array. In this generalized sense multiple representations are no longer related via coordinate transformations (which are only defined point-wise on the base space and its tangential bundle). Multiple representations of data on index spaces with depth larger than 0 can be used to store data referring to other `Topology` Objects, thereby using `Topology` objects as `Representation` *Identifiers*, i.e. "representers", like charts (i.e. "chart objects") select a certain coordinate system. For instance, a data field which shall be computed on cells may stem from a data field given on the vertices, faces or edges (and similar). These different data fields can be interpreted as "multiple representations" (even though there is no natural transformation rule, as with tensorial data on index spaces of depth 0).

EXAMPLES:

- A color value shall be computed for each cell. Color data are given for each vertex and, independently, for each edge (i.e. pair of neighboring vertices). The color value of each cell may be computed by averaging all the colors of the vertices of this cell or from the colors of all edges of this cell. The results may be different, so the first data field is to be stored in the `Representation` object of "vertex-related cell data", the latter one in that one of "edge-related cell data".

- Given the vertices per cells, we want to compute and store the inverse information, i.e. the cells per vertices. This data field, a map from indices of depth 0 to an n-tuple of indices of depth 1 (with different $n \in \mathbb{N}$ for each point), is to be placed at the representation of "the cells in the vertices".

The purpose of multiple representations and representer objects is two-fold: once to describe the history of a data set (how it was created), and secondly to provide parameters for a query of a data set (how to create it).

The availability of multiple representations allows coordinate-free formulations. E.g. the operation $v(f)$ with $v \in \mathcal{T}(M)$ and $f \in C^\infty(M)$ (see def. 27) can be performed in an arbitrary chart $\{x^\mu\}$, where we know the partial derivatives $f_{,\mu}$ and the components of v^μ. However, both $f_{,\mu}$ and v^μ might possibly be known only on a subdomain of the entire manifold, since the chart $\{x^\mu\}$ is possibly not defined on the entire manifold. Thus an algorithm may freely switch (locally) to another chart to compute the required scalar field $v(f)$. The only input of the algorithm are the field names of the vector field v and the scalar field f.

C++ API A `Representation` object is the instantiation of a template class over some "domain". A domain is an index space together with some "cell type" that specifies how to "locate" the elements of the `Representation` objectwithin the topological space described by its representer. An example of a domain is a cartesian chart, where the "cell type" is a triple of floating point values named "x","y" and "z". We get the specific `Representation` object by providing a % object (which is actually an class object derived from an index space of depth 0 `IndexSpace<0>`, see previous paragraph):

```
CartesianChart3D myChart;
Representation<CartesianChart3D>
        myRepresentation = myGrid.Points->find( myChart );
```

Another example of a "domain object" is used for the description of triangle vertex indices; here the "cell type" is a triple of three vertex indices (defined in a `TriangleDomain` class, which is derived from the `IndexSpace<1>` class). The role of a "chart" object is taken by the `Topology` object describing the vertices, e.g. the "Points" component of a `Grid` is used as the representer:

```
Representation<TriangleDomain>
        myTriangleVertices = myGrid.Connectivity->find( myGrid.Points );
```

We may say that "the connectivity of the grid is represented by the points of the grid". The "positional information" of this representation consists of the "points per grid cell". It is now straightforward how to formulate other relationships among `Topology` objects, e.g. the inverse relation of "cells per point" (for instance, the triangles at each vertex). Moreover, `Topology` objects are not required to reside at the same Grid. The formulation of "relative representations" between arbitrary `Topology` objects thus also covers temporal and spatial relationships among different `Grid` objects.

The Representationo provides positional information ("coordinates" of whatever kind) on this index space and an arbitrary number of fields[2]:

```
typedef string  FieldIdentifier ;

template <class IndexSpace, class coord_t>
struct  Representation : RepresentationBase<IndexSpace>
{
        ArrayBase<coord_t, IndexSpace> Positions;

        map< FieldIdentifier , IndexedArrayBase<IndexSpace> >  Fields ;
};
```

The `FieldIdentifier` can be something else than a string, this type definition is given here just for easier understanding. The type of the field identifier is beyond the semantics of the fiber bundle data

[2]Note that in the actual implementation pointers to arrays are used instead of arrays themselfes.

model; a more advanced implementation might employ a more descriptive data structure here to encode the physical meaning of the associated object. For instance, it might specify that field A is the gradient of another field B.

Operating with Field Objects `Representation` objects finally carry arrays (def. 64). All arrays within the `Topology` object housing this `Representation` object are *compatible*(def. 67), thus a `Topology` object has a defined size (def. 66). Arrays within a `Representation` object, the "Field Objects" are identified via "field identifiers". The set of all Field objects with the identical field identifiers within a `Topology` object provides an abstraction of the individual field object and allows to perform *coordinate-free operations*: field operations within a `Topology` object may be parameterized by specifying the just field identifier, thereby deferring the decision which instance of the field object to select among the available representations to the implementation. The set of all Field objects of a `Bundle` object forms a *section* (def. 54) of the fiber bundle. If some algorithm wants to operate on sections only, it only needs to know the field identifier. It is then left to the implementation to decide in which time range to perform the operation. E.g. in some interactive environment it is preferable to execute some algorithm only for the "currently selected" time slice, while in some non-interactive batch job all available time slices would be handled. Also the specification of *which grid* to use is an independent parameter. This capability of specifying field operations independently of a certain `Grid` achieves the desired abstraction from the discretization scheme.
EXAMPLE:

- A data field shall be retrieved for visualization from a remote simulation. Only subsets of the remote dataset, e.g. various arbitrarily oriented 2D slices from within a 3D volume are retrieved upon user request. A visualization may want to display all instances of the same field on the various grid fragments.
- Numerical simulations of a metric tensor field shall be inspected, where data are available on uniform grids, hierarchical meshes, and a tetrahedral triangulation of the spacetime. Once the visualization has been set up for the given field, browsing through the different discretization schemes is possible by exchanging the grid identifiers while keeping the field identifiers constant.

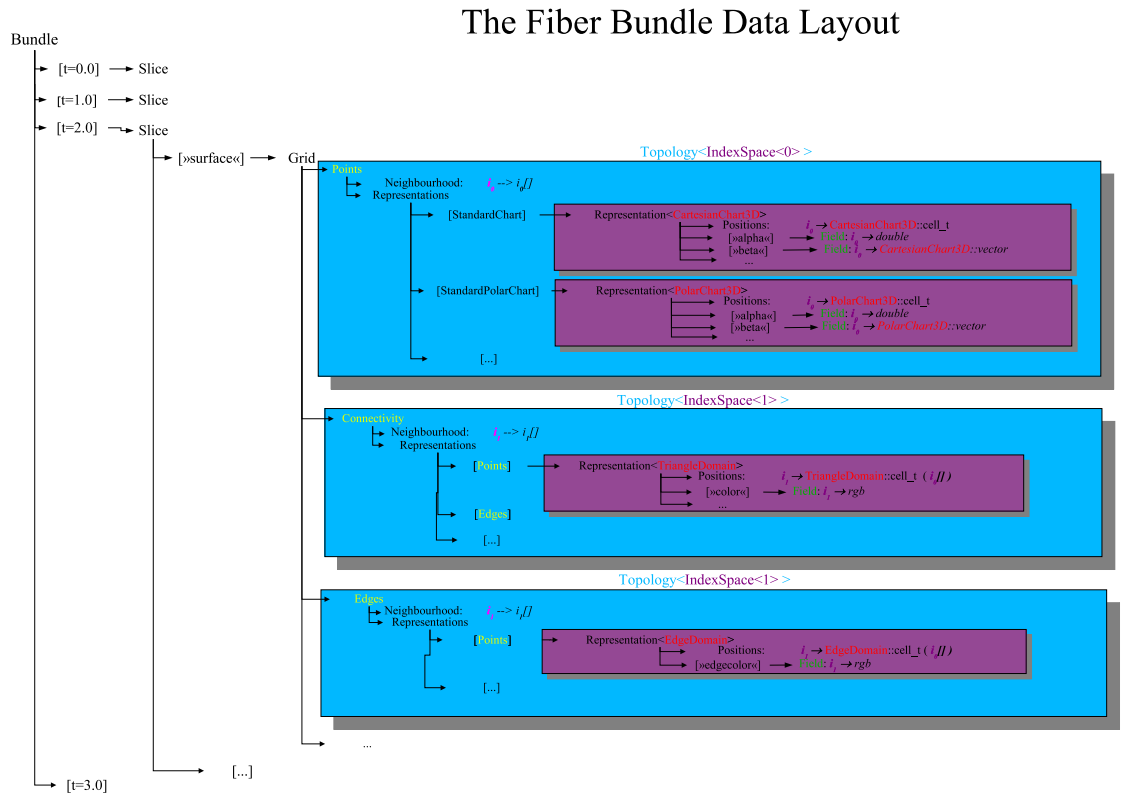


Figure 3.3: Overview of the complete Fiber Bundle hierarchy.

3.1.2 A Semantic Language to Describe Data

The Fiber Bundle data model is built from a tree-like structure of five levels that are called the "Bundle", "Slice", "Grid", "Topology" and "Representation" group. The `Topology` group contains cross-references to other `Topology` groups. A possible branch at this level refers to charts and coordinate systems. We can interpret these hierarchy levels as the "syntax" of a language describing a certain data set, and the datasets residing on each levels as the "words" of this data model language. In this view, a certain data set is a specific sentence in this model. In the following we will explain how to construct such "sentences".

To access the elements of a group, appropriate identifier objects have to be used. These "subgroup identifiers" are of different data type and have a certain meaning:

Group name	identifier type	identifier semantics	result
Bundle	float value	time	Slice Object
Slice	Grid ID (text)	geometric entity	Grid Object
Grid	Topology ID (text)	grid property	Topology Object
Topology	reference	algorithmic context	Representation Object
Representation	Field ID (text)	physical data field	data array

The "Topology Identifier" is not exposed to the end-user. It is required only internally to determine and compare properties of `Grid` objects. The "grid" and "field identifier" are visible to the user. They serve to store semantic information beyond the pure mathematical description. An advanced implementation might use data structures here to allow the formulation of semantic information on an higher level. E.g., it might tell that `Grid` object B is an isosurface of `Grid` object A, or that some field D is the derivative of field C. It might also store physical information like "field ϱ is a density" and "field ε is an energy per volume area", or that "tensor field g is a metric", "K is an extrinsic curvature" (as an example of two fields with the same physical units but different meanings – physical units could be casted into the field's type information as well, similar to Brown's SI unit library [Bro98] where the unit information is formulated via C++ template types). Such extensions are left for future work.

Let us denote the full space of a manifold with any data available on it as the fiber bundle $\mathcal{F}(M)$ (def. 50). The first three hierarchy levels "Bundle", "Slice" and "Grid" just grab convenient subsets of the base space M. A "Bundle" group B corresponds to the complete fiber bundle $\mathcal{F}(M)$ as far as it could be of relevance for application purposes, i.e. $B \subseteq \mathcal{F}(M)$. It is formulated as a cartesian product of spatial subspaces with corresponding fibers and a time coordinate, i.e. $B = \mathbb{R} \times \Sigma_t$, where Σ_t denotes a "Slice" group. It is constructed by a set of spatial subspaces $\Sigma_t = \bigcup_i G_i$, where G_i is "Grid" group. This structure is demonstrated in the following diagram:

$$\begin{array}{lll} \boxed{\textit{Bundle}} & \longrightarrow & \mathbb{R} \times \Sigma_t \\ \quad \Big\downarrow t \ \textit{Time slicing} & & \\ \boxed{\textit{Slice}} & \longrightarrow & \Sigma_t = \bigcup_i G_i \\ \quad \Big\downarrow \textit{Spatial Subdomain} & & \\ \boxed{\textit{Grid}} & \longrightarrow & G_i \subset \mathcal{F}(M) \end{array} \tag{3.2}$$

Within each Slice group, a `Grid` object is uniquely determined by some (textual) identifier i.

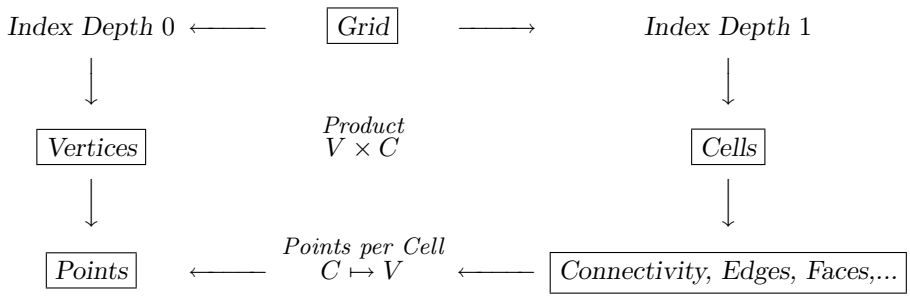
Definition 69 *The set of all Grid groups within all Slices of a Bundle B with the same Grid identifier i is called a* ***Grid Evolution*** *$\mathcal{G}_i(B)$:*

$$\mathcal{G}_i(B) := \left\{ G_i : G_i \subset B = \mathbb{R} \times \bigcup_k G_k \right\}$$

In an user interface only the grid identifier is visible when dealing with a `Grid` evolution. The specific `Grid` object are hidden, although they are the real objects used by visualization and numerical algorithms.

A `Grid` object is constructed by many `Topology` objects with possible relationships among each

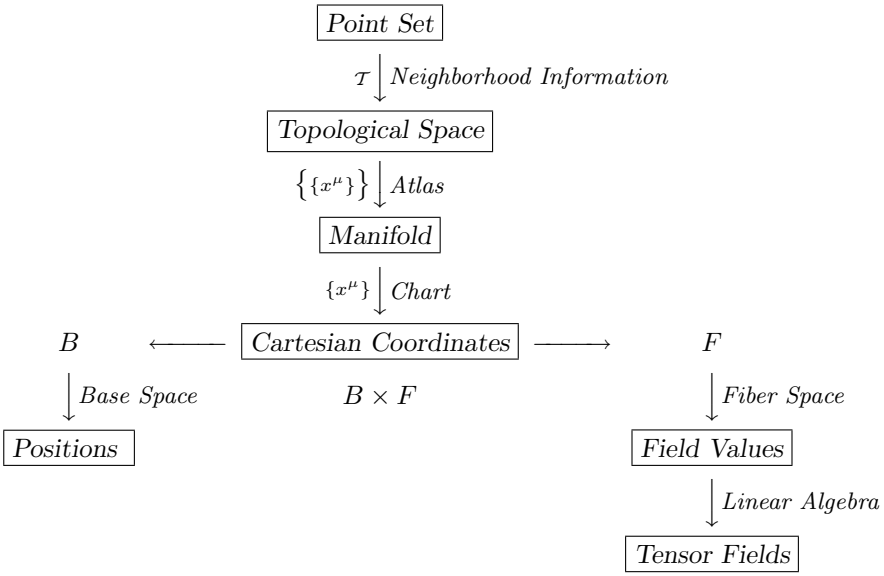
other, as demonstrated by the following diagram:



(3.3)

The "Points per Cell" information is an example of such a cross-reference. Here one topological space is treated as a chart object for another one: like $\mathbb{R}^n$ serves as a chart object for an n-dimensional manifold M (thus providing n real numbers per point via $\{x^\mu\} : M \to \mathbb{R}^n$), also the topological space V may serve as a "chart object" for another topological space C. Here V is the topological space of all "Vertices" and C of all "Cells" of a grid. The map $C \to V$ between the discrete spaces V and C will yield a subset of V for each element of C. It is a subset of $\mathbb{N}$, but in general its cardinality will not be constant for each element of C when the number of points per cell is be different for each cell (unstructured meshes). It will the constant number three for a triangular surface.

These topological spaces on a `Grid` object, formulated by the `Topology` objects, are similar to the usual concepts of "meshes". However, usually "meshes" only refer to primary vertex-related information and intermix cell-related information on them, whereas in this model the vertex and cell-related information is treated distinct in separate spaces. When inspecting the topological space for vertices, then the corresponding Topology object is similar to the original proposal for the formulation of vector bundles[BP89]. We may visualize it in the following diagram, see diagram 2.46 for comparison with Butler's model:

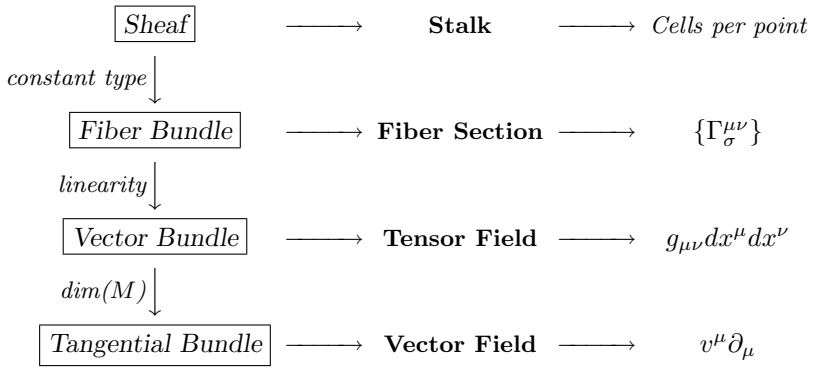


(3.4)

There is a distinct difference to Butler's concept 2.46 in that this model does *not* treat the fiber space with the same semantic structures as the base space. E.g. no neighborhood information is implied for the fibers. Although it might certainly exist, this is left to tensor calculus on the fibers but not casted into the specification of a neighborhood. Instead, the entire fiber bundle is divided into many topological subspaces (structured as shown in (3.4)) with respective fibers. The division into fiber space and base space does no take place before a certain chart object is specified. On the abstract level of an programmer or user interface, the fiber space and base space is only accessed via coordinate-independent `Grid`and `Field`identifiers.

The data in the fiber space can be categorized by the amount of information that is available on it. E.g. the fiber space of a fiber bundle has a certain dimension and thus an element has the same dimensionality at each point. If the dimensionality is unknown or may vary at each point, then we are dealing with the generalization of a fiber bundle to a sheaf with stalks (imposing various implementation difficulties, as mentioned at p.63). If we know more about an object with a fixed dimension, e.g. some linearity relationships, we get a vector bundle, and in the special case of the same dimensionality of the fiber space

and the base space, we end up with an tangential bundle. The relationships of the possible fibers and their specialization by adding more information is shown in (3.5), which can be straightforwardly casted into an class hierarchy:



(3.5)

Classification scheme

Similar to Butler's original approach in 2.2.2 we may derive a data classification scheme from the data model. It can be built from the following questions, which are more or less easy to answer for a given scientific data set:

1. How many `Grid`Evolutions exist in a certain data set?
2. How many time instances exist per `Grid`Evolution ("timesteps")? In general, what is the dimension and density of the parameter space?
3. Which index depths are required per `Grid`?
4. How many `Topology` objects are needed per `Grid` per index depth, dimensionality and refinement? Are there additional properties of a `Topology` object that go beyond these three categorization parameters?
5. How many coordinate systems (chart objects) are desirable for each `Topology`?
6. What is the range of the cardinality for the coordinate representation of the topological points? (How many numbers are required for the coordinates, and if this number varies from point to point, what is the range of possible coordinate numbers?)
7. How many fields are contained in the fiber space of each Topology?
8. What is the tensor rank of a fiber field? If it is not a tensor field, what is its coordinate transition rule?

The answers to these questions uniquely define the placements of pieces of a given arbitrary data set in this data model.

Examples:

- Astrophysical simulation data from the evolution of a star cluster:
  1. One evolving data set.
  2. 1000 timesteps.
  3. Only vertex data are given.
  4. Only points are of relevance.
  5. Data are given in cartesian coordinates.
  6. Base space is three-dimensional.
  7. We have three fields called "mass", "metalicity" and "velocity".
  8. "mass" and "metalicity" are scalar fields, i.e. rank 0×0, "velocity" is a vector field, i.e. a tensor field of rank 1×0.

- A static triangular surface:
  1. One evolving data set.
  2. 1 timestep.
  3. Vertex and Cell data are given, i.e. index depths 0 and 1 are used.
  4. Points and Connectivity are given (one topology of depth 0, one of depth 1).
  5. Vertex locations are given in cartesian coordinates, Connectivity refers to the Points.
  6. Base space is three-dimensional, each connectivity cells consists of three points.
  7. We have no fields on the fiber.
  8. No tensor field information.
- An hierarchical regular AMR mesh (vertex-centered data):
  1. One evolving data set.
  2. 200 timesteps.
  3. Only Vertex-related data are available, i.e. and only index depth 0 is used.
  4. Points information given in 32 levels of refinement, each refinement level is described as its own `Topology`. They contain the same index depth 0 and dimensionality 0, but their refinement level increases. Cell information is available via procedural cell `Topology` objects.
  5. Vertex locations are given in cartesian coordinates and also relative to other refinement levels (but not all of them).
  6. The base space is three-dimensional, each vertex can have 1,2,4 or 8 parent vertices and one or no child vertices.
  7. We have one fields on the fiber called "energy".
  8. It is a scalar field.

Regular Topologies

Grids with regular n-dimensional topologies are usually traversed using n separate indices. Such an iteration scheme allows easy selection of subspaces and subsampling, also known as "hyperslab" operation. Cell and connectivity information is constant for all elements of a grid and usually treated procedurally. The only information that needs to be stored is the extent in each dimension, explicit storage of this cell information were possible, but inefficient. Here we will demonstrate how one can efficiently make use of the abstraction concepts of the fiber bundle environment while still sustaining its generality. They key will be the usage of techniques from C++ template metaprogramming, like those demonstrated by Veldhuizen [Vel95].

Multidimensional Indices An n-dimensional iteration scheme can be implemented independent from the actual dimension by using "multidimensional indices". These are indices providing n incremental operators, i.e. they may be incremented in each of n directions independently. The idea is to follow the mathematical concept of proof by induction: Assume we have some operation defined for n dimensions. Then expand its definition to one more dimension $n + 1$. By providing a one-dimensional operation, we then end up with a natural recursive scheme covering dimensions. This scheme can directly be implemented using recursive template definitions (for the $n \rightarrow n+1$ step) and template specializations (for the $n = 1$ case). As the dimensionality is resolved at compile-time, we may expect optimal performance from the compiled code. The template prototype of a multidimensional index looks as this:

```
template <int dims, class Index>
        class MultiIndex : public MultiIndex<dims-1, Index>
{
        Index    i;
};

template <class Index>
        class MultiIndex<1, Index>
{
        Index    i;
};
```

A `MultiIndex` type of dimension n is derived from the `MultiIndex` type of dimension $n-1$. To stop the recursive definition, the `MultiIndex` of dimension 1 has to be defined separately via template specialization; it is no longer derived from a base class. Each `MultiIndex` contains and "Index" member, which is basically an integer value. With each dimension added, one more Index member is added to the class such that a `MultiIndex` of dimension n contains n index values. The additional `Index` template parameter is used to specify some Index Space, thus allowing a type distinction among semantically different multidimensional indices (e.g. multidimensional indices of cells or vertices).

Multidimensional indices provide incremental and decremental operations. Some native C++ code iterating over an three-dimensional hyperslab,

```
for(int k=k_start; k<k_finish; k+=k_increment)
for(int j=j_start; j<j_finish; j+=j_increment)
for(int i=i_start; i<i_finish; i+=i_increment)
{
        // compute the linear index into a one-dimensional data array
int     linear_index = i + i_TopologySize*(j+k*j_TopologySize);
        ...
}
```

can be written independent of the dimension using multidimensional indices and its associated class, the "`Dimensionator`" class (which is basically a multidimensional index as well, but serves to define sizes, not iteration states):

```
typedef Dimensionator<Dims, Index> Dimensionator_t;
typedef MultiIndex<Dims, Index> midx_t;

Dimensionator_t TopologySize, Start, Finish, Increment;

Dimensionator_t Size = End - Start;
    midx_t mi;
    do
    {
        midx_t midx = mi + Start;
                ...
        index_type i = midx.linear( TopologySize );
                ...
    }
    while( mi.inc(Size, Inc) );
```

The recursive definitions of the involved member functions is not shown here, but they are straightforward to implement. The compiled code is reduced to the same efficient code like the above explicitly version because the dimension is a compile-time parameter. The recursive template programming *forces* loop unrolling for all multidimensional index expressions.

Zero-Memory regular arrays The neighborhood of a regular grid of dimension n can be defined recursively by the tensor product of one-dimensional neighborhoods according to

$$\begin{aligned} R_{n+1} &:= (R_n \otimes \{0\}) \cup (\{0\}^n \otimes \{-1,+1\}) \\ R_1 &:= \{-1,+1\} \end{aligned}$$

and the connectivity (which vertices belong to a cell) is given by the nth tensor product of the set $\{0,1\}^n$. To cast these operations into a recursive template metaprogam to get a compile-time evaluated procedural array, we need objects which

- implement the set $\{0\}$, $\{0,1\}$ and $\{-1,+1\}$ ("constant arrays"),
- implement the conjunction operation $A \cup B$ for two arrays A and B ("consecutive arrays"), and
- implement the product operation $A \times B$ of two arrays A and B ("product arrays").

By a suitable combination of these constant (i.e. procedural, compile-time evaluated) arrays both the regular neighborhood and the regular connectivity information can be casted into suitable template metaprograms providing the desired abstraction level.

Constant Arrays Implementing an array that yields a constant set of values (like the sets $\{0\}$, $\{0,1\}$ and $\{-1,+1\}$) with random access is straightforward. The relevant C++ template parameters are the interval boundaries and the numerical increment. The instantiation of a such a constant array does not use any memory, because the complete information is already available as pure type information. An example implementation as follows:

```
  template <int start, int steps, int increment, class Index>
          class RegularIndexArray
{
public:
        Index size() const
        {       return Index(steps);            }

        Index operator[](const Index&i) const
        {
                return Index( start ) + i*Index(increment);
        }
};
```

Iterators like in the Standard Template Library [S+99] can be added as well.

Consecutive Arrays The conjunction $A \cup B$ of two sets A, B is formulated as a template class with two array types as template arguments. The resulting class shall behaves like one consecutive array. As in STL [S+99], we expect the template arguments to export a `value_type` type member specifying the numerical storage type of the result. The `value_type` from both array need to be identical or at least convertible. An advanced implementation could also compute a type which is sufficient to hold the values of both array arguments; such template algorithms are common practice in template metaprogramming. A `ConsecutiveArray` can be used to, e.g. merge a `vector<>` and a `dlist<>` from the STL, but iterating over the `ConsecutiveArray` does not expose this construction.

```
template <class FirstArray, class SecondArray> class ConsecutiveArray
{
public:
        FirstArray      first ;
        SecondArray     second;

        typedef typename FirstArray::value_type         value_type;
        typedef typename FirstArray:: difference_type   difference_type ;
        typedef ConsecutiveArrayIterator <FirstArray, SecondArray, value_type> iterator ;

        typedef typename FirstArray:: return_value_type     return_value_type ;

        difference_type  size() const
        {
                return  first . size() + second.size();
        }

        return_value_type  operator[](const difference_type &it) const
        {
                return it < first . size() ? first [ it ] : second[ it - first . size ()];
        }
};
```

STL [S+99]-compatible iterators can be added, too.

Product Arrays The product $A \times B$ of two arrays $A : i \mapsto a, B : j \mapsto b$ is an array which maps a pair of indices (i, j) to a pair of values (a, b). The pair of indices can further be reduced to a single index by taking splitting it into an "higher" and "lower" part using a constant number, e.g. the size of one of the two underlying arrays (which is a natural choice, although other choices were possible as well). A possible implementation, which makes use of the STL [S+99] `pair` class, is shown in the following code fragment; note that the `get()` function takes an index pair while the array random access operator `[]` takes a linear index:

```
template <class FirstArray, class SecondArray>
        class   ProductArray : public pair<FirstArray, SecondArray>
```

```
{
public:
        typedef pair<typename FirstArray::value_type,
                                typename SecondArray::value_type> value_type;
        typedef typename FirstArray:: difference_type   difference_type ;

         difference_type    size () const { return  first . size () * second. size (); }

        value_type  get(const pair<typename FirstArray:: difference_type ,
                                        typename SecondArray::difference_type>&it)
        {
                return  value_type(  first [ it . first ],  second[ it .second]  );
        }

        value_type  operator[](const  difference_type &idx) const
        {
        typename FirstArray:: difference_type    it_first    = idx %  first . size ();
        typename SecondArray::difference_type  it_second  = idx /  first . size ();

                return  value_type(  first [  it_first  ],  second[ it_second ]  );
        }
};
```

Each argument of the `ProductArray` template can be an `ProductArray` itself, thus arbitrary powers of product arrays and indices can be achieved.

Regular Neighborhood A regular neighborhood R of dimension $n \in \mathbb{N}$ can be constructed from the cartesian product of the base sets $Z := \{0\}$ and $I := \{-1, +1\}$:

$$\begin{aligned} R_{n+1} &= R_n \times Z + Z^n \times I \\ R_1 &= I \end{aligned}$$

At first we define the multidimensional recursion scheme using the array definitions mentioned above. The `class RegularPointNeighbours` provides an array of n-dimensional indices for the points the n-dimensional point $\{0\}^n$. This class does not require any memory, i.e. it is a type-only class.

```
template <int Dims, class Index> class RegularPointNeighbours
{
public:
        // reference to an eventually existing lower−dimensional neighborhood R_{n−1}
        typedef RegularPointNeighbours<Dims−1, Index> Rn_1;

        typedef typename RegularPointNeighbours<1, Index>::Z Z;
        typedef typename RegularPointNeighbours<1, Index>::I I;

        // The product set R_{n−1} × Z
         typedef ProductArray<Rn_1, Z> Rn1_Z;

        // The set Z^{n−1}, taken from lower dimension definition
        typedef typename Rn_1::Zn Zn_1;

        // The set Z^n (required for higher dimensions)
        typedef ProductArray<Zn_1, Z> Zn;

        // The product set Z^{n−1} × I
        typedef ProductArray<Zn_1, I> Zn1_I;

        // The concatenation of the two subsets R_{n+1} = R_n × Z + Z^n × I
        typedef ConsecutiveArray<Rn1_Z, Zn1_I> Rn;

        typedef typename Rn::value_type              value_type      ;
        typedef typename Rn::difference_type      difference_type ;

        // A formal data member
        Rn      PointNs;
```

```
public: // retrieve the number of elements information
        difference_type size() const
        {       return PointNs.size(); }
        // retrieve the nth neighbor of the point with index 0
        const value_type&operator[]( const difference_type &it) const
        {       return PointNs[ it ]; }
};
```

The formal data member is introduced here only to allow the overloading of a virtual class like in a class hierarchy as shown on p.62. Without this constraint, the formal data member could be omitted and all member functions may be static, because they do not .

What is left is the definition of the one-dimensional case. This is trivial as it is just the set $\{-1, +1\}$:

```
template <class Index> class RegularPointNeighbours<1, Index>
{
public:
        // I is the set {−1, +1}
        typedef RegularIndexArray</*start=*/−1, /*steps=*/2, /*increment=*/2, Index> I;

        // Z is the set {0}
        typedef RegularIndexArray</*start=*/ 0, /*steps=*/1, /*increment=*/1, Index> Z;

        typedef Z Zn;
        typedef I Rn;

        Rn      PointNs;
public:
        typedef typename Rn::value_type          value_type ;
        typedef typename Rn::difference_type     difference_type ;

        // number of elements
        difference_type size() const
        {       return PointNs.size(); }

        value_type operator[]( const difference_type &it) const
        {       return PointNs[ it ]; }
};
```

These classes provide neighborhood information for a point with index 0. It can be extended easily to the neighborhood of an arbitrary point shifting the retrieved indices by a constant offset, i.e. the multidimensional indices of the point of interest (not shown here).

Regular Connectivity Similar to the regular neighborhood information, the regular connectivity information can be constructed using the described technique. The connectivity is based on product arrays of the set $\{0, +1\}^n$, leading to the following recursive definition scheme:

```
template <int Dims, class Index>
class   RegularConnectivity : public RegularConnectivity<Dims−1, Index>
{
public:
        // C is the set {0, +1}^n
        typedef ProductArray< typename RegularConnectivity<Dims−1, Index>::C,
                              typename RegularConnectivity<1,     Index>::C     > C;
};

template <class Index> class RegularConnectivity<1, Index>
{
public:
        // C is the set {0, +1}
        typedef RegularIndexArray</*start=*/0, /*steps=*/2, /*increment=*/1, Index> C;
};
```

The bracketing operator `[]` and the extension to arbitrary points is not shown here, but similar to the case of a regular neighborhood.

Hierarchical and Dynamical Grids

In general, the instantaneous structure of a `Grid` object may change from one time step to the next one, e.g. due to adaptive mesh refinement (AMR), where one grid point may be represented by a couple of grid points in the next time slice, or a particle system simulation where some particles break up into smaller pieces with own trajectories. This behavior can be handled by a representation of a subsequent

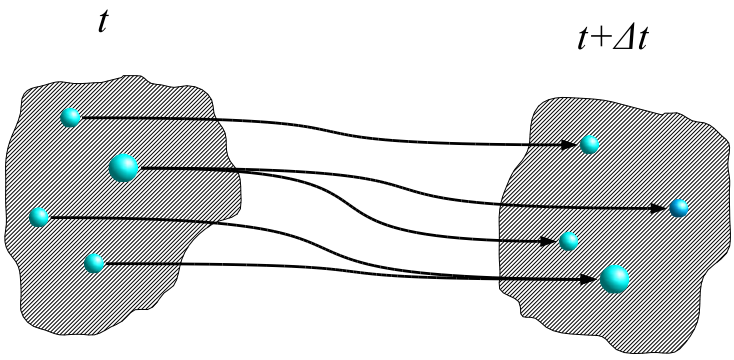


Figure 3.4: Particles within a `Grid` object may split up or merge during a time evolution. The same can happen with the vertices of a `Grid` object or its cells, edges, etc., i.e. with all topological elements that constitute `Grid` object.

`Topology` object by a preceding one. The "cell type" of such a representation are "*future cones*", which map vertices from a time $T = t_1$ to the vertices at time $T = t_2$. They define a child $\leftrightarrow$ parent relationship. For vertices, the cell type is an array f of index depth 0 specifying a number of point indices for the next time step:

$$\begin{aligned} f : I|_t &\longrightarrow \mathcal{P}(I|_{t+1}) \\ i &\longmapsto f_i \subset I|_{t+1} \end{aligned}$$

where f_i is a set of point indices from the index space I_{t+1} at time $t + 1$ which is associated with each element of the index space I_t at time t. The cardinality $\#f_i$ of this set may vary from point to point; so in this case we are leaving the concept of fiber bundles and arrive at the concept of sheaves (see def. 55), which increases implementation difficulties, because the return type of an array is no longer of constant size. However, using appropriate abstraction schemes, this is not a real conceptual problem. The worst case, but also most straightforward implementation would employ a dynamic array like an STL[S+99] `vector<>` for each element in a `C++` implementation. In case of a non-varying grid (i.e. a constant number of vertices) f is an identical mapping, which can be omitted completely in computer memory.

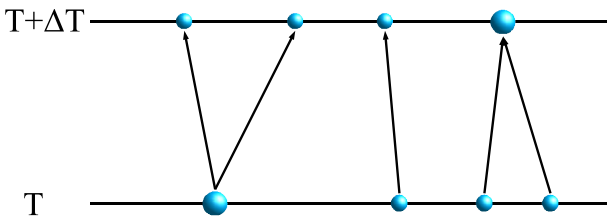


Figure 3.5: "Future Cones" allow the mapping of points from one topological space to the points of its successor in a timelike evolution or among a spatial hierarchy of discrete topological spaces.

The same structure also works as "*past cone*", hereby specifying gives all points which contributed to the current grid point from the previous timeslice. If the number of grid points is constantly increasing in time, then the "past cone" array would contain no more than one past index per grid point (or even none, if some grid point does not have a predecessor). If some grid simplification occurs during evolution and some points dissolve, then the past array will contain some non-trivial information.

The structure of future and past cone arrays can not only be used to indicate temporal changes of the grid, but also spatial ones, therefore referring one grid to another one in a hierarchy. Such *hierarchy cone* arrays indicate correspondence between various grid levels. A maximum of two hierarchy cone levels is sufficient to traverse through arbitrary grid levels (similar to a double linked lists), but there could be more if some requirement arises.

These *cone arrays* (future cones, past cones, hierarchy cones) are used to map information from one topological space (the "source" topological space) into another (the "target" topological space). They play the same roles as coordinate functions map a manifold to $\mathbb{R}^n$. The idea is thus to treat the target topological space like a chart object (fig.3.6). Associated with a chart object (the "target" topological

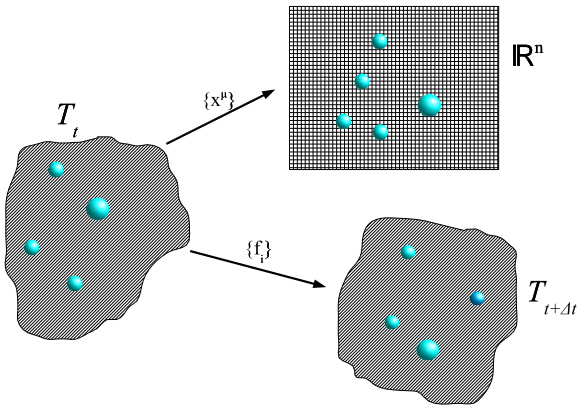


Figure 3.6: Relationships among topological spaces are treated like a coordinate system on a manifold. The coordinate functions are the "cone arrays" in this case.

space in this case) is a `Representation` object in the "source" topological space. It's "Positional" component is the respective cone array. Additionally we may store fiber fields in this representations. This representation of the fiber fields is dedicated for information which is of relevance for the relation of the two topological spaces, for instance the interpolation weights of one grid (e.g. a triangular surface) within another grid (e.g. a uniform three-dimensional grid). These interpolation weights, which specify the contribution of each point's field value from the source grid for each point on the target grid, are identical for all fields that might need to be computed from a source grid to the target grid. They are preferably stored in the relative representation from one grid into another grid.

3.1.3 Combinatorial Structures

The positional information of the "Points" and "Connectivity" Topology objects provide vertex and face lists, which are sufficient to describe polygons. However, for certain algorithms using these data sets as the primary information is not efficient. The "winged edge data structure" has been invented to support efficient operations (see Baumgart [Bau75]) on orientable manifolds. It is usually implemented as a structure describing an edge by pointers to:

1. vertices of this edge,
2. its left and right faces,
3. the predecessor and successor of this edge when traversing its left face, and
4. the predecessor and successor of this edge when traversing its right face.

The use of pointers to relate information among points of topological spaces is incompatible with the concept of (contiguous) index spaces and random access to elements. With an index space, the total number of elements is known, it has "global" knowledge about its element, with some additional information defined on each element, so adding new information to all elements (a new fiber) is trivial, and it can be added once for all elements. Pointer-based structures for all elements with cross-references (e.g. linked lists) do not know about all elements instantaneously , they have just "local" knowledge about one element and its relationship to others; additional information structures (new fibers on each point) have to be added locally to each element.

Nevertheless the information content as required by such local structures can be formulated in the context of the fiber bundle model in a manner similar to replacing pointers by indices. Additionally, any information given per edge needs to be formulated as fields defined on the set of edges (i.e. as fibers on the topological space of edges). The only task left is then to find the "best" place for these fields, where we need to choose an appropriate `Representation` object. Here, we may chose among the space of vertices, edges or faces and need formulate the four fields of the winged-edge data structure as

1. positions of the edges in the representation relative to the vertices, i.e. a pair of vertex indices for each edge

2. a field of a pair of face indices in the representation of the edges relative to the faces, or two fields called "left" and "right" containing face indices

3. a field in the face-relative representation?

4. another field in the face-relative representation?

In all cases we have the option to formulate a field of n elements also by n fields of one element each (operations like Butler's component projection and component direct product, p.40, can be used to switch between both variants). Since 3 and 4 provide information about the neighbors of an edge, also a suitable place could be to cast these fields into the neighborhood information of the "Edges" `Topology` object, thus providing four edge indices per edge.

quad edge data structure The more recent quad-edge data structure was considered by Guibas and Stolfi [GS85]. It is similar to the winged-edge data structure and also makes use of information on vertices, edges, and faces. Edges play the leading role again. The quad-edge data structure groups together four edges: an edge, the "reverse edge" (an edge with the two vertices exchanged), the "dual edge" (referencing to faces instead of vertices, e.g. to the face right of the edge) and the reverse dual edge (referencing to the face left of the edge). The recipe for casting this data structure into the fiber bundle data model is quite straightforward by replacing structure members by fields on the topological spaces of vertices, edges, faces and cells. Algorithms then need to use indices referring to the respective topological spaces instead of pointers to the elements itself.

data structure conversion recipe The general recipe of casting some data structure into the fiber bundle model is to replace pointers by indices and structure members by fields on the index space. For algorithms that require frequent insertion and deletion of points, this re-formulation of the data structures and operations may introduce some performance impact because any change of points in the base space needs to be reflected in the fiber space as well. Additionally, topological cross-relations (future/past cone arrays, hierarchical relationships) need to be checked as well. It might be more efficient to employ some "free index" management instead of continuously adjusting the entire data structure at each change. Another alternative is to use the fiber bundle data structures just for storage of static data and map them to local data structures for highly dynamical algorithms. As such a mapping is bijective, it can be done without much overhead, and still the advantages of both approaches can be used:

	fiber bundle	**pointer-based**
I/O	global (parallel)	local (sequential)
change of base space	costly	efficient
static traversal	easy	easy
adding fibers	easy and efficient	costly
fiber operations	independent of base space	specific to data structure

Explanation of the terms in the comparison table:

- **I/O:** the ability to read/write all data elements at once is a huge performance gain for large datasets; high-performance I/O libraries like HDF5 [NCS03] require such abilities and become very inefficient for sequential element-wise I/O.

- **change of base space:** adding or removing vertices, edges, faces etc.

- **"static traversal":** operations like finding all the edges of one vertex, where the complete information already is available.

- **adding fibers:** adding (or removing) data on vertices, edges, faces etc.

- **fiber operations:** numerical operations on data fields, like vector space operations on vector fields (e.g. computing the magnitude of a vector on each vertex and storing this value with each vertex).

3.2 Interface Specification and Design

The fiber bundle data model is based on hierarchically nested associative maps of identifiers to the next hierarchy level. Instead of having the user of a programming or graphical interface directly deal with actual data (the right-hand side of associative maps), any interactions and requests are to be formulated via the respective *identifiers* (the left-hand side of associative maps). The full complexity of the data model should not be exposed to the end-user, but only those identifiers which provide semantic information above the mathematical context. These are the grid and field identifiers. Additional parameters are possible of course, e.g. the specification of a time range for a certain operation, or the preference to select a certain chart for coordinate-specific operations. Nevertheless, an API or GUI which allows the specification of operations via grid and field identifiers achieves the desired abstraction level.

Example requests:

- *visualize field "density" on grid "accretion disk"*. Such a request would trigger an animation of the evolution of some density field. It might perform coordinate transformations if the data are provided in a coordinate system which is not appropriate for the visualization algorithm, e.g. polar coordinates instead of cartesian coordinates.
- *evaluate field "lapse" from grid "simulation volume" on a surface "apparent horizon"*
- *save field "beta" on all grids to disk*
- *display a list of all fields from a grid called "neutron star simulation"*

3.2.1 Implementation Issues

Templates vs. Inheritance

A crucial decision is what part of the code in a C++ implementation shall be formulated by generic programming and which part by object-oriented programming. C++ offers both possibilities, whereby support for object-oriented programming is superior to generic programming. However, generic programming offers a huge performance gain for runtime code, especially when modern techniques from *template metaprogramming* [Vel95, SS01, Ber00] are employed.

Template metaprogramming is based on the discovery that C++ templates form a programming language on their own on top of C++ . C++ templates provide

- Loop iterations through recursive template definitions
- Conditional operations through template specialization

As elaborated by Veldhuizen [Vel], C++ templates thus form a turing-complete language, i.e. *any* algorithm can be implemented upon templates (at least in theory; practical restrictions arise due to limited recursion depths, compiler bugs and last not least due to an awkward syntax which tends to get out of the programmer's control). These template metaprograms are *interpreted* by the C++ compiler (which can be forced to compute e.g. the number π and print it as error message – that was actually the historic discovery of this compiler feature) and transformed into constant numbers or execution code. In contrast to a preprocessor, C++ templates are *not* independent from C++ : they "know" about C++ and e.g. allow computations on *types*, not only on numbers.

With the recent advances of modern compiler technology, techniques from template metaprogramming are increasingly used, e.g. POOMA [Cod] is entirely based on generic programming. However, such codes have been mainly developed to perform numerical simulations where the desired operations are known prior to program execution. This is not the case for an interactive visualization environment like Amira [Con03]. Here the requested operations depend on user interaction and thus cannot be known at compile-time. Therefore it is not possible to completely cast an interactive visualization program into a template metaprogram. An appropriate compromise is required to still profit from these advanced programming techniques as far as possible.

In the case of the Fiber Bundle concept, which will describe very low operations like element-wise data access in a very abstract way, the use of operations that can be inlined by an optimizing compiler is strongly desirable. When performing operations on each vertex of a huge data set, each performance gain counts. On the other hand, we need to collect algorithms on a higher level such that they can be called at runtime through abstract interfaces using virtual function calls. The fiber bundle data model

Object oriented programming	Generic programming
uses inheritance	uses templates
decision at run-time	decision at compile-time
works for objects derived from the common base	works for objects having the right members
one function created for the base class – **saves space**	a new function created for each class used – **wastes space**
virtual function must be compiled in separate code (function call overhead) and cannot be optimized beyond the function body – **slow**	code can be inlined and optimized beyond the function body – **fast**
extension possible using only definition of base class	extension needs definitions and implementations of all functions
useful for application frameworks, user interfaces, high level structures	useful for small, low level constructs and generic algorithms

Table 3.2: Generic vs. object-oriented programming

then provides means on how to parameterize such high-level operations, see p.86 and p.84, through grid and field identifiers.

The basic idea is to invoke pre-compiled template metaprograms through virtual functions from abstract base classes:

```
class   AlgorithmBase
{
public: virtual void compute(int parameter) = 0;
};

template <class Type> class Algorithm : public AlgorithmBase
{
        void compute(int parameter)
        {
                // invoke type-specific template metaprogram here
        }
};
```

Then some code may operate on `AlgorithmBase` classes and invoke element-wise operations without knowing the detailed implementation of a template instantiation. For instance, the `AlgorithmBase` class could stand for an object describing the representation of vertices in a coordinate system, whereas the derived object is the actual representation of the vertices in some 3-dimensional coordinate system using some application-specific storage for the coordinate numbers. Coordinate transformations can be requested in an abstract way which only involves base objects (at runtime), whereas the actual code is instantiated once a new coordinate system is introduced in the application (at compile-time). As a drawback, this code design does not allow the introduction of new coordinate systems at runtime.

Beneath the possible runtime performance gain achieved through template metaprogramming, templates can be used as compile-time check for conditions which otherwise can only be checked at runtime. By using such mechanisms, it might be sometimes harder to get code to compile, but once it is compiling, it is less prone to runtime errors. An example is to extend native C++ types with additional information that can be checked for code consistency, like physical units [Bro98] or "index depth" according to def. 68. In other words, the cost of struggling with template constructs might be overcompensated by reduced runtime debugging costs. Certain kinds of errors, like using an index within a wrong index space, simply can no longer occur.

Memory Management

> C programmers think memory management is too important to be left to the computer. Lisp programmers think memory management is too important to be left to the user [ES].

In this implementation, we adhere to the latter opinion. Practical experience shows that memory management in plain C++ is a real burden. Dangling pointers (unsoundness) and memory leaks (incompleteness) are known pitfalls and direct consequences of the C++ native pointer concept. Dealing with these insufficiencies of the C++ programming language, also known as "bugfixing", takes a significant amount of time, which should better be spent on the real problems. It is thus better to solve this memory management problem once in a general manner than to permanently suffer under it.

Survey of Existing Approaches Methods for automated memory management are subsided under the keyword of *garbage collection*. In the domain of C and C++ , garbage collection has a bad odor because it is considered to be complex with unpredictable runtime behavior and application pausing. The current opinion in literature is that this was true in the early days of garbage collection research and development, but this is no longer valid and algorithms have matured since. These days most non-trivial C++ libraries and applications employ some kind of garbage collection, including such high-performance libraries as OpenInventor. Within the terminology of garbage collection schemes, OpenInventor's algorithm is called "direct identification of garbage with immediate identification and reclamation (object destruction)", more commonly known as *reference counting*. This method is applicable for data structures which are organized in a polytree (a singly connected graph). The Scene Graph of OpenInventor fits well into this concept. In this scheme the operation of "immediate reclamation", i.e. the reclamation of resources when objects are deleted, is unbounded (it may take infinitely long) and may be an expensive operation because a possibly large subtree must be completely traversed and destructed by a single event. For deep trees with many leaves deferred reclamation/object destruction (e.g. by keeping a list or stack of to-be-deleted objects which is decreased at the next allocation routine) is preferable within reactive applications (those that need to respond fast on some event). "Deferred identification" is an optimization method to reduce temporary changes in reference counts and is said to half the overall memory management time, but it may cause minor pausing. Further optimizations for small objects (where problems arise when the reference count itself occupies a significant portion of the overall memory) are possible by distinguishing among single and multiple references.

Direct identification (reference counting) operates locally. As a consequence, non-local structures are beyond its scope and may cause troubles. These arise if there are *cyclic references*. In the simplest case, two objects refer to each other and thereby keep themselves alive, even if there is no reference from outside (e.g. from variables within the main program, the "*roots*" of the graph). Such unaccessible but resource occupying objects are "*garbage*", leading to memory leaks. A possible solution to this problem is to distinguish among weak and strong references: Strong references form an acyclic graph, which is closed by weak references. Algorithms exist to trace cycles and to determine garbage this way.

A common approach to handle cyclic reference is to trace the entire graph of referencing objects from the roots and to remove non-connected objects. This is called "indirect identification" of garbage. It is done e.g. via marking the subgraphs of roots recursively and removing unmarked objects after completion (*mark and scan*). Optimizations aim at reducing the (memory, disc,...) storage defragmentation which happens after many allocation/deallocation operations with varying storage cell sizes. Compaction of non-contiguous free memory into larger contiguous segments is done by *sweep* algorithms (e.g. [JM88]), which move all objects and update all references to them. Such mark-scan algorithms have significant application pause time, but it can be parallelized, such that a single collector thread runs concurrently with one or many allocating threads. Reducing the need to move objects is subject of further optimizations, some of which speed up on the cost of memory usage, others trace generations of objects (generational garbage collection) based on the observation that objects are mostly either very short-living or very long-living.

Hybrid methods of direct and indirect identification do local marks and scans, but may suffer under the complexity of required operations at each object destruction.

Requirements in the Fiber Bundle Data Model In the case of the fiber bundle data structure we do not have a large subtree nor do we expect a huge number of objects, so this structure is appropriate for reference counting. However, due to cross-references we do not have a single connected graph, thus requiring a more advanced scheme. Most of the memory requirements will be due to relatively few but contiguous large data sets, so sweeping objects is not acceptable. Deferred reclamation is not desirable, because a certain object might be tied to external resources like a graphical user interface component, open files or network connections, texture memory in graphics hardware and similar. If such external

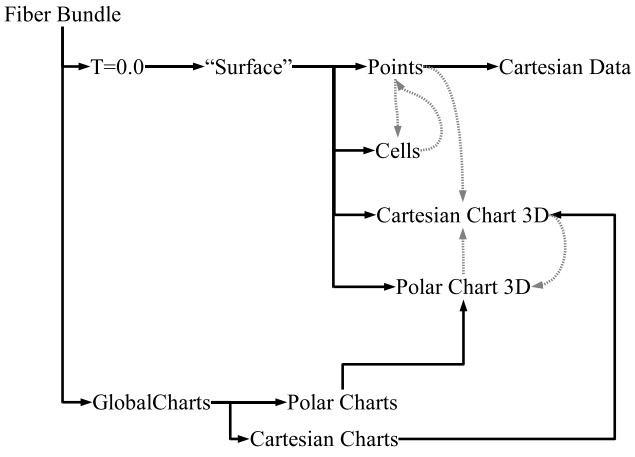


Figure 3.7: An example graph of referencing objects demonstrated on the description of a surface in polar and cartesian coordinates

resources are no longer required, they should be released as soon as possible. Usually the effort of (possibly recursively) freeing resources is minimal.Last not least, because the entire data model implementation has been constructed from scratch with no a-priori constraints (e.g. without the requirement to use native pointers or to interface some existing reference counting scheme like that of OpenInventor), it could be built as a cooperative environment, e.g. by explicit specification of strong and weak references.

The resulting constraints for the implementation of a superior "pointer" C++ classes or template therefore are

1. reference counting: keep objects alive exactly as long as at least one pointer refers to it
2. weak and strong pointers: allow back links and circular references [3]
3. support of the upcasting operation, conversion from child class to base class (going "up" in the class hierarchy), like it is always possible with native C++ pointers
4. support of the downcasting operation: conversion from base class to child class (going "down" in the class hierarchy), possibly revealing a NULL pointer, similar to the `dynamic_cast<>` pointer conversion in C++
5. constant objects
6. multiple inheritance
7. thread-safety

The condition "exactly as long as" instead of "at least as long as" rules out garbage collection schemes, because we here want explicit control of resources.

Implemented Solution Simple reference counting pointers are dangerous because

- conversion from reference pointers to native C++ pointers is problematic, because the native C++ pointer does not ensure the existence of the pointed object - if the reference pointers is destructed, then the object might be deleted but the C++ pointer is still existent ("dangling pointer"). It is thus advisable to access reference counted objects through reference pointers only, not using C++ native pointers at all.
- circular references among reference counted objects leave to memory leaks. Such objects will never be removed from memory, although they can no longer be accessed from the application any more. A typical case is a double linked list, where each object refers to its successor and predecessor. An object containing a reference pointer to itself will keep "itself alive" forever.

[3] In Qt, weak pointers are called "guarded pointers".

The implementation achieves 1 and 2 via indirect referencing using an intermediate reference counting object. The reference count is not stored at the objects themselves, but in an special "referrer" object. This referrer objects stores a pointer to the object plus two reference counters, a "weak" and a "strong" reference count. The actual object contains a pointer to such a referrer object, exactly one such referrer exists per object. Any pointer to the object increments the weak reference count. Strong pointers also increment the strong reference count. Destruction of strong pointers decrements the strong reference count, the last one also deletes the object, but leaves the referrer object intact. This referrer object is kept alive as long as at least one weak pointer to the object exists. A weak pointer appears as invalid ("NULL") pointer if the strong reference count is zero. Any strong pointer is also a weak pointer. The object itself contains a weak pointer to itself. Explicit destruction of the object resets the strong reference count of its referrer object to zero, thus invalidating all strong and weak pointers to it. This mechanism provides significant features:

1. back links are possible, thus completely eliminating the need to use C++ native pointers with their danger of dangling pointers and unfreed memory,
2. explicit destruction of objects is allowed, thus pointers to automatically allocated objects on the stack (no dynamic memory allocation overhead, except for the intermediate referrer) are possible,
3. to avoid access to reference counted objects via C++ native pointers, there should be no way to get the address of an object via its reference pointer, except for debugging purposes. However, all object members shall be accessed via the reference pointer through, appropriate operator overloading
4. no need for internal `static_cast<>`'s (like in the concept of "engines" that are sometimes used in template metaprograms), because the referrer object contains an pointer of exact type to the object.

Reference Pointers and Runtime Type Information A useful extension of reference pointer templates is their combination with runtime type information by implicit usage of C++ `dynamic_cast<>`s. Standard C++ code as

```
Base * B;
Derived * D = dynamic_cast<Derived*>(B);
        if (D)
        {
                ...
        }
```

which can also be written in the shorter and more efficient (because the lifetime of object D is reduced) code variant using a declaration within the `if`-statement:

```
Base * B;
        if (Derived * D = dynamic_cast<Derived*>(B) )
        {
                ...
        }
```

can then be reduced to the shorter

```
RefPtr<Base> B;
        if (RefPtr<Derived> D = B)
        {
                ...
        }
```

This shorter notation is especially useful when "`Derived`" is a complicated class name, e.g. some template class with multiple parameters. However, arbitrary type conversions when using reference pointers is not desirable, as this "feature" would annihilate C++ 's type safety. It is thus preferable to allow dynamic type conversion only within certain groups of classes, e.g. all classes which are derived from a common base class (implementation details are not given here).

Reference Pointer Arithmetic Pointer arithmetic as it is known from native C++ pointers does not make sense for reference counting pointers, because reference pointers are designed to point to single objects, not to multiple objects residing consecutively in memory. Therefore incrementing or decrementing reference pointers is not a valid operation. Nevertheless it is still of interest to have an ordering

relationship among them to allow storing reference pointers in a ordered map or a hash table. Thus each reference pointer needs to be assigned a number that is uniquely associate with the managed object. At first glimpse one might think that the object's memory address would be appropriate here. However, it is not, because such a table could also be constructed by weak pointers; such weak pointers may change their values (because their managed object is being destroyed) after insertion in the table, thus destroying the table's ordering. Retrieving the address of an destroyed object instead of NULL is not a good idea either, because this address might be re-used due to a new allocation on the same numerical address value. Instead it is possible to use the memory address of the internal intermediate reference counting object. It is safe to use this address, because it remains valid and unique as long as at least one pointer exists. This is surely the case as long as a table of pointers exists.

3.2.2 Abstract Interfaces for Grid Manipulations

Beside the data layout of the fiber bundle data model, its concept also provide a natural way to categorize, parameterize and implement operations on this data structure. We not only get an abstraction scheme on data organization, but also on procedures.

Grid Algorithms

Definition 70 ***Grid Algorithms*** *are functions of a single Grid yielding one or more values:*

$$\begin{aligned} f : \mathbb{R} \times \Sigma_t &\longrightarrow \mathbb{R}^n \\ G &\longmapsto f(G) \end{aligned}$$

Examples of grid algorithms are visualization methods. They take a grid object and compute some result (e.g. an image), which is not intended to be stored at the source grid.

The properties of a `Grid` are determined by a set of (independent, but related) `Topology` object. Some grid algorithms can be split up to operate on the various `Topology` object independently, e.g. some component of a grid algorithm might require just neighborhood information on the cells, another portion just requires data given on the points of the grid (for instance, some averaging algorithm, which projects data given on points to cells). Such an algorithm can be split up into two components:

1. a component which requires topological information (neighborhood relationships)
2. a component which performs actions based on data fields

The first component iterates over a set of elements: we call this a "*topological iterator*". The second component executes certain operations per element: we call this a "*topological operator*".

This split of a grid algorithm is similar to the interpretation of a grid G as subset of a data space as subset of a fiber bundle $B \times F$, but now this split is performed within the function space: The component of some grid algorithm $f(G)$ which operates on the base space is the topological iterator $f(B)$, the component operating on the fiber is the topological operator $f(F)$. In pseudo-mathematical notation:

$$\begin{array}{lccc} \text{data space} & G & \xrightarrow{\text{"fiberize"}} & B \times F \\ & \downarrow & & \downarrow \\ \text{function space} & f(G) & \xrightarrow{\text{"fiberize"}} & f(B) \times f(F) \end{array} \tag{3.6}$$

Such a split of an grid algorithm into base and fiber component (topological iterator and operator) obviously requires some linearity properties of the algorithm, which is not the case for arbitrary algorithms. Nevertheless a large class of algorithms fulfills this condition and can thus be "fiberized". Such algorithms are parallelizable and scale linearly.

Examples for Topological Iterators:

- An often required Topological Iterator is an iterator which takes a Topology object and a Representation identifier to traverse over all the positional elements of the respective Representation; it calls an Topological Operator for each element. This is an "Positional Iterator".
  - A Positional Iterator for index space 0 is a "Point Iterator", which calls a Topological Operator for each vertex.

  - A Positional Iterator for index space 1 is a "Cell Iterator", which calls a Topological Operator for each cell. A Cell Iterator may call a Point Iterator for each cell to traverse over the vertices of a cell.

- A more complex Topological Iterator takes a sequence of grids (e.g. a Grid Evolution) to trace points throughout this sequence. It is useful e.g. to draw the trajectories of particles.

Examples for Topological Operators:

- Visualization methods which take point coordinates and draw icons at each point. The icon may be parameterized by the values of a field at the same point, eg. a point colored by a scalar field, a vector arrow, a covector plane, a tensor glyph. Most of the tensor field rendering algorithms described in the next chapter are implemented as topological operators.
- A visualization method which takes a line segment and draws it with special illuminations techniques (e.g. as presented by Zoeckler,Hege,Stalling [ZSH96]). In conjunction with a Cell Iterator, this viz operator can display the edges of the cells of a Grid or integral lines of a vector field; in conjunction with a Trajectory Iterator it draws the paths of moving particles. Still, there is one implementation of the graphical rendering for multiple purposes.

Implementing Topological Iterators and Operators Usually both components interact closely, so an efficient implementation is required to allow an efficient splitting of an algorithm into various "topology-related fragments". Operators only know about atomic data objects, but have no knowledge about their relations in a larger context. Iterators only know how to find those atomic data objects and how to traverse paths within them, but have no knowledge about the actual operation which is executed on them. This approach allows maximal reusability of algorithms.

A Topological Operator that is to be called for each vertex element of a `Grid` object with coordinates specified in a cartesian three-dimensional chart looks is demonstrated by this prototype (it could, for instance, issue some OpenGL calls):

```
class Operator
{
public:
        int begin();
        int apply(IndexSpace<0>::index_t i, CartesianCoordinates3D&Point);
        int end(int);
};
```

Note that none of the Operator's function are virtual. A compatible Topological Iterator is a C++ template taking the Operator as argument. Its purpose is to execute the Operator's member function at each element. The Topological Iterator for traversing a regular three-dimensional grid could be implemented as follows:

```
template <class Operator> class RegularIterator3D
{
public:
        int apply(Grid&G, Operator&what)
        {
        index   max_i, max_k, max_j;
        Array<index,CartesianCoordinates3D> Positions;
                ... // initialize max_... and Positions from the Grid
        int     e = what.begin();
                if (e) return what.end(e);

                for(int k=0; k<max_k; k++)
                for(int j=0; j<max_j; j++)
                for(int i=0; i<max_i; i++)
                {
                int I = (k*max_j+j)*max_j + i;

                        e = what.apply(I, Positions[i]);
                        if (e) return what.end(e);
                }
                return what.end(e);
        }
};
```

In practice it were preferable to employ multidimensional indices, which has been omitted here for demonstration purposes. By instantiation of the Iterator class on the Operator class, we get a concrete class that achieves full performance since in the inner loop there is no function call overhead and all operator functions may be inlined by an optimizing compiler. Nevertheless the Iterator class may be derived from an abstract base class that leaves the `apply()` member function virtual, in accordance with the concept of abstract algorithms as described on p.80. The topological Iterator may also depend on any runtime or compile-time parameters, e.g. a time interval that allows the iterator to select a subset of points from a `Grid` evolution.

Operator Caching `Grid` objects are a good place to store Operator objects such that an Operator object may remember its state when it is applied again at to the same `Grid` object. Thereby some computation can be abbreviated if the input parameters remain unchanged.

The mechanism works by querying a `Grid` object for an existing Operator object first, and only if no such Operator object exists yet on a `Grid` object it is created, otherwise the old one is re-used. Operator objects may co-exist in different flavors according to their input parameters and even for identical input parameters produce different results, depending on how the Iterator Object applies the Operator Object to each point. Thus the "Operator Cache" is accessed via two steps: first via an "iteration state", which is constructed from the Iterator Object only, and secondly from an "operator state", which reflects the relevant parameters of the Operator Object.

$$\text{Grid Algorithm:} \qquad f(G) \xrightarrow{\text{"fiberize"}} f(B) \times f(F) \xrightarrow{\text{Iterator State}} f(B)$$
$$\text{Operator State} \downarrow$$
$$f(F)$$

An application example are OpenGL animations of evolving Grids: once an OpenGL display list has been compiled for each `Grid` object within an evolution, the display lists may be recalled from the operator cache, thus achieving fast playback.

Grid Operations

Definition 71 *A **Grid Operation** is a function f which operates on components from one or more grids and stores the result in a grid. In pseudo-mathematical notation, with $G, G_0, G_1, ..., G_n$ grids, f can be written as:*

$$f : G_0 \times G_1 \times \ldots G_n \longrightarrow G$$

Data sets (except the topological neighborhood information) are only present in `Representation` objects. A frequent task is to compute data for storage in a `Representation` object (the "destination", R_{dst}) from data given in another `Representation` object (the "source", R_{src}). We may categorize this operation by the relationship of the source and destination `Representation` objects:

1. R_{src} and R_{dst} are contained within the same Topology (i.e. the data are different representations of the same topological space); then this operation is a **coordinate transformation** (also called a chart transition, def. 17)

2. R_{src} and R_{dst} reside in different `Topology` objects, but are related to the same chart; then this operation is an **evaluation**

3. one or more source `Representation` objects R_{src} and the destination `Representation` object R_{dst} refer to the same chart and reside in compatible `Topology` objects within `Grid` evolution; then this operation is an **interpolation**.

Examples

1. **Coordinate transformation:** A metric tensor field is given in polar coordinates; it needs to be computed in cartesian coordinates.

2. **Grid evaluation:** compute "cell-centered" data from "vertex"-centered data and vice versa or determine some data values on a 2D surface given some 3D field.

3. **Grid interpolation:** compute smooth intermediate values of data given in subsequent time steps.

All these operations of chart transition, evaluation and interpolation are hereby subsidized to the same interface. Moreover, all these operations can be nested transparently.

For instance the "evaluation" operation requires the source and destination representation to reside in the same chart. If there is no such representation in the `Topology` object t containing the source `Representation` object, it is necessary to compute the required data representation via chart transitions within the source `Topology` object. In case it is not possible to perform a chart transition to the destination chart on the source chart, then all charts which have a transformation chain to the destination chart need to be inspected. One or more chart transitions need to be performed after the interpolation has been done in a found compatible chart. The following diagram displays the situation when data are requested on a destination `Topology`T_{dst} in a chart $\{x, y, z\}$ from a `Topology`T_{src}. Hereby data exist only in a chart $\{\eta, \varrho, \zeta\}$. There is no direct transformation rule, but a common chart $\{r, \vartheta, \varphi\}$ exists with transformation rule $\{r, \vartheta, \varphi\} \rightarrow \{x, y, z\}$ in the destination chart and $\{\eta, \varrho, \zeta\} \rightarrow \{r, \vartheta, \varphi\}$ in the source chart:

$$\begin{array}{ccc} \mathbf{R_{dst}}\{\mathbf{x}, \mathbf{y}, \mathbf{z}\} & & \mathbf{R_{src}}\{\eta, \varrho, \zeta\} \\ {\scriptstyle (r,\vartheta,\varphi)\rightarrow(x,y,z)}\uparrow & & \downarrow{\scriptstyle (\eta,\varrho,\zeta)\rightarrow(r,\vartheta,\varphi)} \\ R_{dst}\{r, \vartheta, \varphi\} & \xleftarrow{eval} & R_{src}\{r, \vartheta, \varphi\} \end{array}$$

The request is formulated via specification of the destination `Grid`, G_{dst}, the desired destination chart $\{x, y, z\}$ and the source `Grid`G_{src}. The "point" `Topology`component is used by default, but may be overridden by some optional parameter.

EXAMPLE: A scalar field is given in some $\{\eta, \varrho, \zeta\}$ coordinates. A transformation rule exists into polar coordinates $\{r, \vartheta, \varphi\}$. The scalar field needs to be known on a uniform cartesian grid (e.g. for hardware-accelerated volume rendering). This mechanism has e.g. be used for the volume rendering of "Lazarus" data, as described in 4.3.3.

C++ API: Coordinate Transformation Given a certain chart, a `Grid` objectcan be queried for `Representation` object that contains a specific field:

```
template <class Chart_t>
 RefPtr<Representation<Chart_t> > QueryRepresentation(Grid&G,
                                                      const RefPtr<Chart_t>&WhichChart,
                                                      const string &fieldname);
```

E.g. some vector field is given in polar coordinates. It is required in cartesian coordinates. If a transformation rule from polar to cartesian coordinates has been defined for the Charts on the `Grid` object(within the "Grid Atlas"), it is employed internally to compute the result. Transformation rules among cartesian charts might be involved as well to shift the origin and the orientation of the polar coordinate system.

C++ API: Coordinate Transformation and Evaluation If some field is known to reside on some source grid (e.g. data in a grid which is uniform in polar coordinates, but not in cartesian coordinates) but is required on some other grid (e.g. on a uniform cartesian grid), then it can be requested via a function

```
RefPtr<VertexArray_t>  QueryTransformativeEval(Grid&Dest, Grid&Source, const GridObj&Fieldname,
                                               const RefPtr<ChartBase>&WhichChart,
                                               const  TopologyIdentifier &whichTopology=VERTEX_TOPOLOGY);
```

The field may hereby be given on the Vertices or the Cells of the source grid. In the latter case, no interpolation but straight copying will be performed. The implementation will then traverse a possible chain of transformation rules on both the source and destination grid until the evaluation operation can be performed in the "best" way.

C++ API: Hiding Evaluation as Coordinate Transformation It is possible to use the simpler interface `QueryRepresentation()` for doing a coordinate transformation instead of using the more complex `QueryTransformativeEval` interface by defining some known grid as a possible source when requesting a field on the destination grid. This approach has the advantage that the evaluation operation is then *deferred* until the data is actually required, like a transformation rule that exists, but is possibly not evaluated. For this purpose, we need to tell the fiber bundle data structures that some grid may be a possible source of data for another grid:

```
bool EstablishGridSource(Grid&Dest, const RefPtr<ChartBase>&myChart, Grid&Source);
```

C++ API: Evaluation, Coordinate Transformation and Interpolation Having set up all possible evaluation and transformation rules, it is then possible to query the Bundle Object to provide a certain field on a certain grid in a certain chart for a certain physical time. It is up to the implementation to internally find a coordinate system that allows timelike interpolation (if required) and some grid objects, that contains appropriate source data bracketing the requested physical time. This "`GainField()`" is one of the highest-level procedures:

```
RefPtr<VertexArray_t> GainField(Bundle&B,
                                const GridIdentifier &geometry_grid,
                                const GridIdentifier &data_grid,
                                const GridIdentifier &fieldname,
                                const RefPtr<ChartBase>&WhichChart,
                                double&t, double deltaT,
                                RefPtr<Grid>&DestGrid,
                                RefPtr<Slice>*GridSlice,
                                const InterpolationOptions_t &IOpts);
```

Here the "`data_grid`" parameter is used to establish a "grid source" for some yet unknown time steps. The "`geometry_grid`" parameter is used to find some destination grid at some unknown time slice, too. The result is a grid object with positional information interpolated from the "`geometry_grid`" evolution carrying a data field which is interpolated from the "`data_grid`" evolution and evaluated on the appropriate "`geometry_grid`".

Using the Grid Abstractions

When working with the fiber bundle model, its available data structures and possible operations, the following recipe can be used to formulate a given algorithm in this context. It is not claimed that it can be applied to an arbitrary type of problems or task, but it can be used for a large class and enables strong reusability of code and concepts. Some examples are given in more detail in section 4.3.3.

- Determine what is the input and output of some algorithm:
  - Can the input be formulated in terms of grid objects?
  - If so, is it a single grid or a group of grids?
  - What components of the grid objects are required for the algorithm?
  - How many parameters (e.g. chart objects) need to be specified to find these components of a grid?

  The answers to these questions determine the input parameters of an algorithm.

- Determine what is the output of some algorithm:
  - Can the output be formulated in terms of a grid object?
  - If yes, can the algorithm make use of available grid operations like coordinate transformations, grid evaluation, grid interpolation or coefficient evaluation? Can it be constructed by grid operations which require only a subset of the input parameters?
  - If no, can it be formulated via Grid Algorithms? Can these Grid Algorithms be separated into a Topological Iterator and Topological Operator?

  The systematic analysis of the answers to these questions open a path for constructing reusable algorithms. Components of such an algorithm can be used in another context as well with high probability.

For input and output parameters which can be formulated completely via `Grid` objects and their components, an algorithm can be parameterized via identifiers instead of actual objects. E.g. instead of specifying a set of `Grid` objects, also a time range, a grid name and a bundle object can be used. This is the parameter set that should be exposed to the end-user. A set of identifiers thus form the "words" of a semantic language in which the end-user formulates his requirements.

3.2.3 Local and Global Chart Objects

For performing coordinate transitions on a `Grid`, a chain of transformation rules is required. When a data representation in a certain chart is requested, this chain is traversed until the data source is found. Each `Grid` object is equipped with a set of charts, each of them (optionally) providing transformation rules to other charts. This set of chart objects is called the "*Grid Atlas*", whereby each transformation rule might be limited (e.g. via an index range or set of valid indices) to a certain subdomain of the entire `Grid`. Consecutive coordinate transformations correspond to a path through the `Grid` atlas. Given just the endpoints of such a transformation chain, the transformation path is not unique if there exist many bijective transformation rules. Strictly mathematically, this is not important, because all true coordinate transformations commute. However, if we treat general grid operations (def. 71), e.g. grid interpolation, through the same interface as coordinate transformation, this commutativity is lost. For instance, consider data are given in a source chart at time steps $T = 4.0$ and $T = 6.0$. Data are requested in a destination chart at an intermediate time step $T = 5.0$. This can be achieved by first performing a chart transition at the time slices when data exist, and then interpolating these to the requested time:

$$\begin{array}{ccccc} R_{src}(T=4.0) & & & & R_{src}(T=6.0) \\ \text{chart transition}\downarrow & & & & \downarrow\text{chart transition} \\ R_{dst}(T=4.0) & \xrightarrow{\text{interp.}} & R_{dst}(T=5.0) & \xleftarrow{\text{interp.}} & R_{dst}(T=6.0) \end{array}$$

Alternatively the temporal interpolation may be performed first in the source chart, and then the interpolated result is transformed into the destination chart:

$$\begin{array}{ccccc} R_{src}(T=4.0) & \xrightarrow{\text{interp.}} & R_{src}(T=5.0) & \xleftarrow{\text{interp.}} & R_{src}(T=6.0) \\ & & \text{chart transition}\downarrow & & \\ R_{dst}(T=4.0) & & R_{dst}(T=5.0) & & R_{dst}(T=6.0) \end{array}$$

These two variants do not commute in general (practical example given in A.1.3). It is thus necessary to label a certain path of data transformations. Therefore we introduce "local" and "global" chart objects: A certain chart object is part of a single `Grid` (but it may be shared among many `Grid`'s). The set of all such chart objects constitutes the "local charts". In order to trace a certain coordinate system through various grids and time slices, a common chart identifier is required to define compatibility of local charts. This chart identifier constitutes a "global chart" and is related to many grids. A list of global charts is stored in `Bundle` objects and called a "*Bundle Atlas*". It allows to detect which charts may be used for data representations. However, a certain `Grid` object might not be able to support a specific global chart. Also, a `Grid` atlas may contain charts which are not contained in the `Bundle` atlas, meaning that these charts are for internal usage withing this `Grid` object only and should not be exposed to the user.

The distinction among local and global charts allows to specify which of the above interpolation methods is to be used: temporal interpolation (in general: `Grid` interpolation) is permitted for two representation groups if their local chart object (their representer object) is identical. Otherwise, temporal interpolation may not be performed in this chart, and intermediate time steps must be computed via chart transitions from compatible representations.

Chart Objects and Transformation Rules

Local chart objects serve as an "anchor" for `Representation` objects, i.e. they define its meaning, the domain of a Representation object. Each of them provides a set of transformation rules, which are described by `Transformation` objects. For n charts there need to be n^2 `Transformation` objects in general, but some transformation rules might be omitted due to a limited domain of a chart or numerical instabilities of certain transitions (e.g. due to coordinate singularities).

Each `Transformation` object, which handles the transition from a source chart $\{x^\mu\}$ to a destination chart $\{x^{\bar{\mu}}\}$, must be able to provide

- a set of destination coordinate numbers $\{x^{\bar{\mu}}\}$ for a given set of coordinates $\{x^\mu\}$,
- a transformation matrix $\alpha^{\mu}_{\bar{\mu}}$ for lower indices and
- a transformation matrix $\alpha^{\bar{\mu}}_{\mu}$ for upper indices.

This information is sufficient to compute the chart transition of arbitrary tensor fields according to (2.22). Geometric objects with known non-tensorial transformation rules like the Christoffel symbols (2.36) can be transformed as well. The specification of one transformation matrix is sufficient, because the other one can be computed numerically via matrix inversion or by copying (or sharing) the numerical values from the inverse transformation rule. In practice it might be preferable to avoid such inversion rules because of numerical instabilities. Transformation rules can be specified in several ways:

1. Intrinsically known transformation rules, e.g. from polar to cartesian coordinates and vice versa; they need to be built into the implementation and are selected via keywords or enumeration types. Such `Transformation` objects don't require any memory storage (except the minimal keyword) and provide arbitrary resolution, but are limited to a-priori knowledge of explicit transformation rules. Typical examples are transitions from polar to cartesian coordinates and vice versa.

2. Analytical transformation rules, which are evaluated from expression trees, and need to be parsed from some input format.

3. Per-point specification of transformation matrices: coordinate values and transformation matrices are specified as explicit numbers for each point of the vertices of a grid.

4. Coordinate projections: as an extension of the mathematical requirement of bijective transformation rules, also the operation of coordinate projections can be casted as a non-invertible transformation rule, e.g. the projection on a sphere:

$$pr_{\text{spherical}} : \mathbb{R}^3 \longrightarrow \mathbb{S}^2 \tag{3.7}$$
$$\{r, \vartheta, \varphi\} \longmapsto \{\vartheta, \varphi\} \tag{3.8}$$

   The according transformation matrices are not invertible. Such transformation rules allow to inspect three-dimensional data e.g. on a two-dimensional sphere by just requesting cartesian data in $\{x, y, z\}$ coordinates in $\{\vartheta, \varphi\}$ spherical coordinates (the transition $\{x, y, z\} \to \{r, \vartheta, \varphi\}$ is performed internally as an intermediate step when requesting data in $\{\vartheta, \varphi\}$ coordinates).

5. Coordinate Fields: As the inverse operation of coordinate projections, a fiber field can be specified to be used as a new coordinate. E.g. a transformation object may specify to use a scalar field named "R" given on a spherical surface with coordinates $\{\vartheta, \varphi\}$ as a new radial coordinate. We thus get a representation of the surface in polar coordinates

$$\text{Coordinate Field "R"} : \mathbb{S}^2 \times \mathbb{R} \longrightarrow \mathbb{R}^3 \tag{3.9}$$
$$(\{\vartheta, \varphi\}, R) \longmapsto \{r, \vartheta, \varphi\} \tag{3.10}$$

   which may furthermore be transformed implicitly e.g. into cartesian coordinates. A typical application example are ray surfaces described by coefficients of spherical harmonics (see also A.2).

6. Grid sources: According to p.87, a transformation rule may also specify another `Grid`as source where some data may be retrieved. Since `Grid` interpolation may only be performed in the same local chart, the transformation matrix is the identity matrix here and may be omitted. However, the positional information of the relative representation of the two grids involved may be used to store interpolation weights, which have a similar (but not identical) purpose as transformation matrices.

Implementing the Atlas

Coordinates Coordinates are an n-tuple of (real-valued) numbers, see def. 17. The most direct way to cast this information into C++ is a structure definition which

1. exports the dimensional number of the coordinates as an `enum` constant called `dims`
2. exports the base type (`float, double`,...) as a typedef called `real`
3. provides storage place for the coordinate values under convenient freely definable names

An example is the definition of cartesian coordinates $\{x, y, z\}$ in $\mathbb{R}^3$:

```
struct CartesianCoordinates3D
{
        enum { dims = 3 };
        typedef float real;
        real x, y, z;
};
```

Other predefined coordinates are polar coordinates $\{r, \vartheta, \varphi\}$ (coordinates on $\mathbb{R}^n$):

```
struct PolarCoordinates3D
{
        enum { dims = 3 };
        typedef double real;
        real r, theta, phi;
};
```

For practical reasons, it is convenient to inherit coordinate types from a generic indicable type and to add some operators to the coordinate types, but these extensions are not relevant for the definition of charts (next section). An alternative formulation is also to employ type traits template classes for providing the required information on externally pre-defined types. This variant has the advantage that any existing coordinate type from an external library can be used, but this implementation detail will not be discussed here.

Tangential Objects Objects residing in the tangential space are described by a set of components in their coordinate representation. A certain component is accessed using an "upper" or "lower" index. To mime this functionality and the safety of mathematical index expressions as known from the Einstein sum convention, some C++ code should be able to distinguish among upper and lower indexing and detect wrong code at compile-time. Here, a pair of integer proxy classes are of use. They should allow a mapping from one index type to its dual type (index raising and index lowering):

```
template <class Chart> struct LowerIndex;
template <class Chart> struct UpperIndex;

 /// Class handling lower indices
template <class Chart>
        struct LowerIndex : public IntProxy<Chart>
{
        /// The UpperIndex is dual to this one.
        typedef UpperIndex<Chart> DualIndex;

        /// raise lower index using ~ Operator
        DualIndex operator~();
};

/// Class handling upper indices
template <class Chart>
        struct UpperIndex : public IntProxy<Chart>
{
        /// The LowerIndex is dual to this one.
        typedef LowerIndex<Chart> DualIndex;

        /// lower upper index using ! Operator
        DualIndex operator!();
};
```

An example on how to use these type-safe index classes will be shown on page 92. Additionally, we need a storage place for the actual data, i.e. an object that may be accessed with indices. This generically "indicable" object may also provide vector space operations like an addition operator and scalar operation:

```
template <class Index, class Element>
        struct Indicable
{
        /// Dimension of the underlying manifold of this Indicable
        enum { Dims = Index::dims };
        Element elements[Dims];

        /// Constant indexing operator
        const Element&operator[](const Index&I) const;
        /// Scalar multiplication
        Indicable operator*(double B) const;
        /// Vector addition
        Indicable operator+(const Indicable&B) const;
};
```

Note that hereby we assume that the `Index` class knows about the dimension of its underlying chart and exports it as an `enum` type. Tensorial objects may now be defined recursively by instantiating `Indicable` objects over other `Indicable` objects as "elements", starting from scalar objects (i.e. floating point types). Examples are given in the next section.

Chart objects A chart $\{x^\mu\}$(def. 17) is implemented via a template that takes the coordinate definition as its argument:

```
template <class Coordinates>
        class Chart
        {
                ...
        };
```

For a given chart$\{x^\mu\}$, e.g. cartesian coordinates,

```
typedef Chart<CartesianCoordinates3D> CartesianChart3D;
```

the instantiation of the chart template induces type definitions like a chart $\{x^\mu\}$ induces a basis in the tangential and co-tangential spaces $\{\partial_\mu\}$, $\{dx^\mu\}$ (def. 22):

```
CartesianChart3D::Vector
CartesianChart3D::Covector
CartesianChart3D:: Bilinearform
```

Tensors of higher rank can be defined recursively by combining elements from $T_p(M)$ and $T_p^*(M)$ using the `Indicable` template class described in the former paragraph together with the index types that are exported by the chart class as

```
CartesianChart3D::Upper
CartesianChart3D::Lower
```

For instance, most common basic tensorial types are constructed (within the chart) as:

```
typedef Indicable< Upper, real>      Vector;
typedef Indicable< Lower, real>      Covector;
typedef Indicable< Lower, Covector> Bilinearform;
```

These types allow vector space and contraction operations (def. 24), e.g.

```
CartesianChart3D::Vector     u,v,w;
CartesianChart3D::Covector   U,V,W;
CartesianChart3D:: Bilinearform  F;
float a,b,c;
        u = a*v + b*w;          // addition and scalar  multiplication  on vectors
        U = a*V + b*W;          // addition and scalar  multiplication  on covectors
        a = U*v;                // contraction  of vector and covector (scalar  product)
        W = F*u;                // contraction  of vector and bilinear  form
        c = v*F*u;              // applying a 2-form to two vectors
```

Note that vector space operations are not defined on the coordinates themselves, but just on these objects residing in the tangential space. The coordinate type might support such operations, but the chart definitions do not make use of this assumption that the underlying manifold has vector space structure. This distinction among point coordinates without vector structure and tangential vectors with vector space operations is unusual within computer graphics, where points in space are usually treated as vectors ("just three numbers", see the OpenGL manual for example). However, this distinction results in cleaner formulations of algorithms and operations, therefore leading to easier generalization of algorithms to e.g. curved surfaces where the assumption of vector space structure of the base manifold is no longer valid.

To access the components (def.2.1.1) of an indicable object, the appropriate index type from the chart has to be used, e.g. $v^\mu = 0.5$ for $\mu = 1$:

```
CartesianChart3D::Vector     v;
CartesianChart3D::Upper mu(1);
                v[ mu ] = 0.5;
```

The requirement to use appropriate index types for tensor operations ensures type safety at compile time and a direct mapping of mathematical expressions to the code, e.g. coding the expression $s = g(v, w)$ with g a bilinearform, $v, w \in T_p(M)$ and $s \in \mathbb{R}$, i.e. in a char$\{x^\mu\}$ $s = v^\mu g_{\mu\nu} w^\nu$, appears as:

```
double r = 0;
        for(Upper i; i.valid(); i++)
        for(Upper k; k.valid(); k++)
        {
                r += v[k] * g[!k][!i] * w[i];
        }
```

An advanced implementation might even omit the loop expressions and define an "indexed indicable" object as its own type (making use of techniques similar to expression templates).

Limitations

1. Symmetry properties of tensors cannot be specified in the current recursive scheme. To optimize memory usage, e.g. the metric is defined via an own template class adhering to the symmetry and storing only $\frac{n(n+1)}{2}$ elements instead of n^2, but otherwise behaves like a tensor object.

2. Here, the basic "real" type is propagated from the definition of the initial coordinate to all induced tensor types. It might be desirable to easily mix "cartesian float metrices" and "cartesian double metrices". This can be done via additional template parameters and member templates, but imposes an n^2 problem on all tensorial operations, because the numerical type is left free in such a case and all possible combinations of numerical types must be supported on all operations. However, this issue is basically limited by compiler technology.

Coordinate Transformations Coordinate transformations are formulated via Transformation objects, similar to the `Transition` object in Butler's concept (p.41). They implement the coordinate transition rules $\{x^{\bar{\mu}}\}(\{x^{\mu}\})$, the transition matrices $\alpha^{\bar{\mu}}_{\mu}$ and their inverse $\tau^{\mu}_{\bar{\mu}}$ (see also p.16):

```
template <class SourceChart, class DestChart>
struct Transformation : TransformationBase<SourceChart>
{
        typedef typename SourceChart::Point SourceChartPoint;
        typedef typename DestChart::Point   DestChartPoint;

        typedef typename DestChart::Upper DestUpperIndex;
        typedef typename DestChart::Lower DestLowerIndex;

        typedef typename SourceChart::Upper SourceUpperIndex;
        typedef typename SourceChart::Lower SourceLowerIndex;

        typedef typename SourceChart::real SourceType;
        typedef typename DestChart::real   DestType;

        //  {x^μ̄}({x^μ})
        DestChartPoint operator()(const coord_index_t &i, const SourceChartPoint&P);

        //  α^μ̄_μ ≡ ∂x^μ̄/∂x^μ ({x^μ})
        typedef Indicable< DestUpperIndex, Indicable< SourceLowerIndex, SourceType> > Alpha;
        //  τ^μ_μ̄ ≡ ∂x^μ/∂x^μ̄ ({x^μ})
        typedef Indicable< SourceUpperIndex, Indicable< DestLowerIndex, DestType> >   Tau;

        // Check for singularity during transformations. If one occurs, return false.
        bool computeAlpha(const coord_index_t&i, Alpha&alpha, const SourceChartPoint&P);
        // Check for singularity during transformations. If one occurs, return false.
        bool computeTau(const coord_index_t&i, Tau&tau, const SourceChartPoint&P);
};
```

Transformation rules may be given analytically and specialized for certain chart types, e.g. for implementing the known transformation rule among cartesian and polar charts. In this case, the `Transformation` object will be independent from a `Grid` object and the last parameter `SourceChartPoint P` is used to determine the coordinate location of the source point. In contrast, when a transformation rule is given explicitly at each point of a `Grid` object via numerical values, then the first parameter `coord_index_t i` will be used to retrieve the `Grid`-dependent in the discrete set of transformation matrices and point coordinates as provided by the `Grid` object.

The availability of the transformation matrix allows to compute the chart transition for every tensorial object. It may be defined recursively by providing a transformation rule for a single index like in the expression

$$T^{\bar{\mu}\cdots}_{\cdots} = \alpha^{\mu}_{\bar{\mu}} T^{\mu\cdots}_{\cdots}$$

In C++ this transformation rule is implemented via a transformation rule that operates on an arbitrary object that can be indiced with an upper index:

```
template <class Tensor, class SourceChart, class DestChart>
Indicable <typename DestChart::UpperIndex, Tensor>
 mul(const typename Transformation<SourceChart,DestChart>::Alpha&a,
     const Indicable <typename SourceChart::UpperIndex, Tensor>&T )
     {
     Indicable <typename DestChart::UpperIndex, Tensor> Tb;
        for(typename DestChart::UpperIndex mb; mb.valid(); mb++)
        {
                Tensor&Tmb = Tb[mb];
                Tmb = 0;
                for(typename SourceChart::UpperIndex m; m.valid(); m++)
                {
                        Tmb += a[mb][m] * T[m] ;
                }
        }
        return Tb;
     }
```

Technically, this code just performs a matrix multiplication. Semantically, this code is type-safe and ensures correct indexing. Further optimizations may include using expression templates, but this option has not been implemented in the current version. As a notable limitation, there is no automatic way to implement transformation rules for non-tensorial objects yet; these must be specified manually.

Global Chart Objects The concept of "local" vs. "global" charts allows to distinguish among two levels of compatibility among `Representation` object:

- "weak" compatibility: two `Representation` objects refer to the same coordinate system
- "strong" compatibility: data may be interpolated among two `Representation` objects

In an implementation, a local chart is an object carrying `Transformation` objects as described before. If two `Representation` objects refer to the same `Chart` object (i.e. their "Representer" is identical), they are regarded as "strongly" compatible and data interpolations may be performed. Each such `Chart` object is now equipped with a reference to a global chart object, which is basically just some textual description providing some information for the end-user. If two `Representation` objects refer to different `Chart` objects with the same global chart identifier, then they are treated as "weakly" compatible, i.e. data must not be interpolated among them. An interpolation request on two weakly compatible representations must traverse all possible coordinate transformation rules on both representations until a common `Chart` object is found. Here, data may be interpolated, and (possibly interpolated) transformation rules must be employed to yield the requested result on the interpolated grid. Since a data transformation might be grid interpolation again, a simple request might thus initiate a complex computational engine involving many operations.

Computing Coordinates

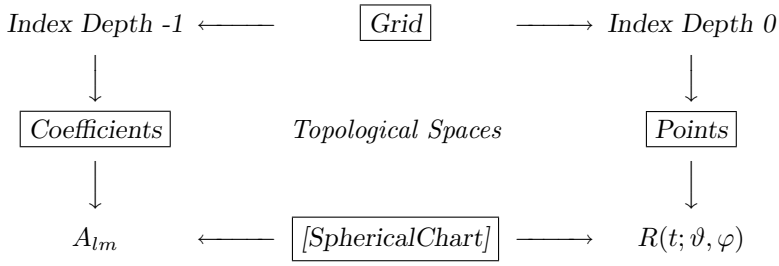
In many situations the spatial coordinates of some `Grid` object are given procedurally based on a set of coefficients describing the linear combination of basis functions, for instance fourier transformation data (with data given in fourier space), multispectral expansions (based on spherical harmonics), NURBS (non-uniform rational B-splines) and chebychev coefficients. Common to these data sets is that the actual spatial data can be computed with arbitrary geometrical resolution from a relatively small data set of basis function coefficients. An application example is given in A.1.3, where the temporal evolution of a surface is given as an expansion of spherical harmonic functions:

$$R(t;\vartheta,\varphi) = \sum_{l,m} A_{lm}(t)\, Y_{lm}(\vartheta,\varphi) \tag{3.11}$$

Hereby $A_{lm}(t)$ are the time-dependent coefficients as they are output by the numerical algorithm, $Y_{lm}(\vartheta,\varphi)$ are the usual spherical harmonics and R is the radial distance of a point of the surface as measured from some origin (which is also a free parameter and part of the surface description). Such a surface is also called a ray surface, since each point is associated uniquely with a ray trough the origin. Note that a straightforward implementation of (3.11) based on the recursive mathematical definitions of Y_{lm} via Legendre polynomials is relatively slow and, more serious, numerically instable. Section A.2 reviews and extends a numerically stable and fast algorithm presented originally by Deuflhard [Deu76].

Expansion coefficients are neither integer data related to cells or cell complexes, nor spatial data, so the question arises where to store them in the fiber bundle data model. Let us remember the concept of index spaces (see def. 64) and index depth, def. 68, for this purpose. Index depth 0 is related to discrete points in space, index depth 1 are aggregates of elements of index depth 0 ("cells", "edges", "faces", ...). So index depth 1 is only defined and makes sense if elements of index depth 0 exist. An index space of depth 0 can "live" without an index space of depth 1 (corresponding to a zero-dimensional set of points), but not vice versa. In this sense, index depth 0 "bears" index depth 1. Here, we have a similar situation physical coordinate locations are not available without the expansion coefficients: the expansion coefficients "bear" the physical coordinate locations. It is therefore quite canonical to extend the concept of index depth to negative numbers and let an "index depth -1" indicate that data of this kind are designed for computing coordinate locations.

A `Topology` object with index depth -1 is called a "Coefficient Topology". It describes the topological space of coefficients which serve to compute spatial coordinates.



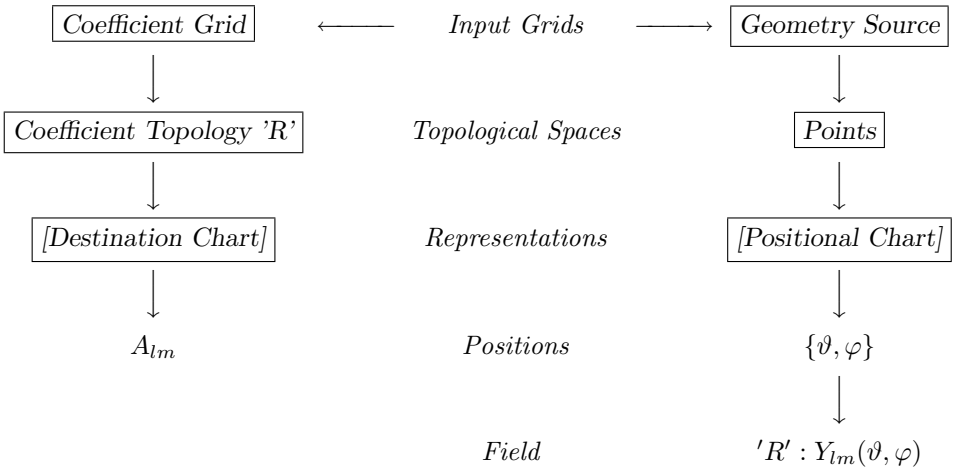
(3.12)

A `Coefficient Topology` may carry multiple representations for different coordinate systems as well. Each of these may provide different coefficients, for instance, think about multipole expansions around different locations in space.

Coefficient Evaluation and Geometry sources Beside the coefficients (e.g. the A_{lm} from (3.11)), we also need to know the appropriate basis functions (e.g. $Y_{lm}(\vartheta,\varphi)$ from (3.11)) to compute final coordinate values. These basis functions need to be known on spatial locations, i.e. on some pre-existing geometry. We can provide them as a *fiber field* on an existing `Grid` object. This "geometry source" grid can

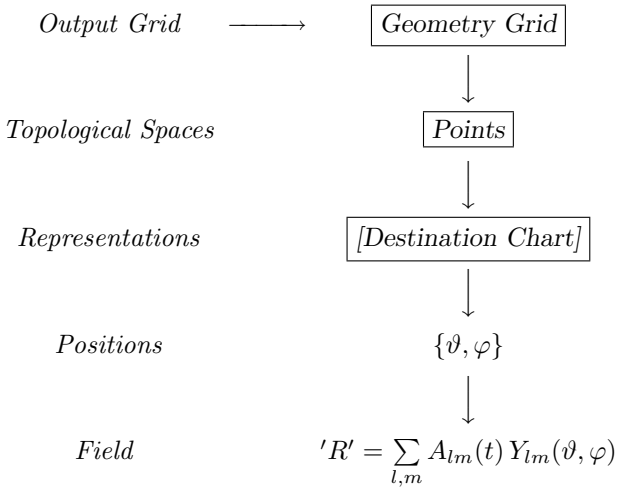
- be a constant object over multiple time slices, such that the basis functions need to be evaluated only once (a possibly computationally expensive operation), or
- be an object which is interactively modified by the user, such that the resolution of the resulting coefficient expansion is adjustable to the user's need, or
- one of many possible triangulations (e.g. a sphere with regular or triangular cells), or
- provide stored basis functions on the grid, if the application does not have means to evaluate them procedurally (which of course limits the spatial resolution then).

The computation of coordinates from some given coefficients can be formulated as a `Grid` Operation (def. 71), which maps two input grids, the "coefficient grid" and the "geometry source grid" to the "geometry grid":



(3.13)

We call this `Grid` operation a "**Coefficient Evaluation**". It takes two input grids and two charts as parameters: the "destination chart", which specifies on which coordinate systems the coefficients are valid, and the "positional chart", which specifies the coordinate system where the geometry is given. The "Coefficient Evaluation" will then need to find coincident coefficients topologies on the coefficient grid and a corresponding field on the geometry source grid (e.g. by using identical names, "R" in (3.13)) in the respective charts. The newly computed spatial information will be put onto the destination chart's representation of the new (or already existing) grid as a field with the same name ("R" in this example):



(3.14)

It is then left to a coordinate transformation object to interpret the resulting field as a coordinate field according to 5 on p.90. As all these operations can be implemented to happen *on demand*, the request for a `Grid`'s coordinate representation in some cartesian coordinate system (e.g. for visualization purposes) may therefore initiate a Coefficient Evaluation through the mechanism of chained coordinate transformation rules.

A more complex example where the surface coordinates $\{\vartheta, \varphi\}$ are computed from given coefficients as well is given in A.1.3. The availability of this functionality in the implementation enabled the researchers at the Max-Planck-Institute for Gravitational Physics to develop and test a new method of finding embedding diagrams of black hole apparent horizons, as presented by M.Bondarescu, M.Alcubierre and E.Seidel [BAS02]. On the visualization side, it involved many Coefficient Evaluations for a single `Grid`.

3.3 Summary

3.3.1 Overview of Supported Features

The ultimate goal of this fiber bundle data model is to achieve an abstraction layer to describe data objects and operations on them in a powerful manner with high re-usability. Especially for the purpose of tensor field visualization, where many non-trivial operations happen just in the tangential space $T_p(M)$ of a manifold M, the demand is high to unify such operations instead of re-implementing them for every different layout of the base manifold M. The two main abstractions on data objects to achieve are

- the abstraction of objects from their numerical representation in a specific coordinate system - the fiber bundle model achieves this through multiple representations on various topological spaces, and in particularly by employing "field identifiers" as abstract handles instead of actual data objects, and
- the abstraction from the underlying discretization scheme of the manifold - the fiber bundle data model achieves this through "grid identifiers", which allow to specify a certain layout without knowledge of its internal discretization structure.

On the procedural side, the fiber bundle data model defines the concepts of `Grid` algorithms (def. 70) and `Grid` operations (def. 71):

- For `Grid` algorithms, it provides a systematic way for developing and implementing algorithms on various manifold discretization schemes with high re-usability of algorithmic components. The key concepts here are the separation of `Grid` algorithms into "Topological Operators" and "Topological Iterators".
- For `Grid` operations, the model provides a unification of *coordinate transformations*, *grid interpolations*, *grid evaluations* and *coefficient evaluations*. All these operations are hidden behind a common interface to query data fields in a certain data representation. It employs an engine to traverse a chain of generalized data transformation rules, which is parameterized by means of "global" and "local" chart objects.

In contrast to alternative data models and their implementations, the fiber bundle model intrinsically supports the following features right from beginning:

- series of consecutive grids (`Grid` evolutions)
- connectivity information across time steps
- relative information among `Grid` objects, e.g. interpolation weights
- various discretization schemes via separate topological properties
- non-manifolds as base spaces
- multiple coordinate systems and multiple representations of data in their respective charts
- abstraction level to allow coordinate-free formulations
- non-vector spaces as fibers
- cell complexes and grouping of `Grid` cells
- data fields (fibers) may reside not only on elements of the base space (points or vertices), but also on cells, edges or lines, faces, cell complexes (patches), etc.
- a recipe for the construction of a large class of generic, reusable `Grid` algorithms (e.g. some visualization method)
- coordinate transformations and chains of chart transitions
- support for temporal interpolation
- multi-patch data via locally bounded Representations
- structured and unstructured grid types, such as:

- particle systems, see A.1.1 and A.1.1
- triangular surfaces, see A.1.2
- regular surfaces, see A.1.2
- uniform grids, see A.1.2
- regular grids, see A.1.2
- adaptive hierarchical meshes (AMR data), see A.1.2
- curvilinear grids
- tetrahedral grids
- unstructured grid

The basic concept is to avoid implicit assumptions about a data set as much as possible and to specify its properties in detail (possibly procedurally for performance issues). The data model is casted into a tree-like structure of five hierarchy levels, called the Bundle, Slice, Grid, Topology and Representation levels. Instead of dealing with actual data objects, it employs abstract identifiers referring to data or a group of data. These identifiers stand for the semantic information that is represented by the data they refer to. It is claimed that the usage of textual descriptions for identifiers is an indicator of the incompleteness of a data model, because textual descriptions are interpreted by a human user, not by the data model. Therefore these identifiers are casted into data structures themselves as far as possible, but some open issues remain. The two identifiers for Grid and Field objects are left open a textual identifiers intentionally, as these are considered to contain semantic information beyond the mathematical capability of the fiber bundle model. These are thus the only identifiers that are exposed to the end-user.

3.3.2 Comparison with Alternative Approaches

Having presented the newly proposed data model, we may inspect its relationship to three exemplary implementations. All these three selected approaches can be found to be a subset of the new data model. The closest model is provided by the widely spread and successful DX model, but the fiber bundle model discussed here adds additional structure such as the concept of index spaces with multiple representations.

Butler's Vector Bundle Prototype

Although fundamental differences exist as compared to Butler's visionary original approach – see diagram (3.4) on p.69 in contrast to (2.46) on p.39, we can seek for coherences to the terms defined in 2.2.2:

1. Butler's "Fiber Constructors" correspond to the type information of the fibers; their implementation is the `ArrayBase` class from 3.1.1.
2. Butler's "Base Constructors" correspond to the `Neighborhood` component in the `Topology` objects and the set of *all* `Position` arrays in the `Representation` objects which are contained in this `Topology` object.
3. Butler's "Bundle Constructors" correspond to the `Topology` objects, which carry the base space and its fibers.
4. Butler's "Section Constructors" correspond to the `Array` classes from 3.1.1, or generally, any `Array` object that may provide actual data.

Procedures on the data model are called "bundle mappings" in Butler's approach. He does not distinguish between operations within a `Grid` object and operations among `Grid` object, but the corresponding operations are:

1. Bundle restriction: adding or removing `Grid` objects
2. Subbundle: adding or removing `Field`s
3. Boolean operations: grouping of `Grid` objects; this is problematic to implement, because the topological spaces of grid objects might be quite different and it might not even be reasonable to perform boolean operations among grid objects with different topological properties.
4. Slicing: Selecting a subset of the vertex-related information of a `Grid` or interpolating a `Grid` onto another `Grid` of lower dimension.

5. Stacking: Merging `Grid` objects; similar to boolean operation, it makes only sense for a certain family of `Grid` objects with sufficient "similarity".

6. Component projection: Has no conceptual correspondence, because it is a coordinate-dependent operation. An implementation might still provide such a functionality on the risk of the user.

7. Component direct product: only tensor operations or covariant expressions are supported.

Summarizing, Butler's model is a good source of inspiration, but it completely lacks any description of important grid features others than vertices. Also, it does not support generic transformation rules on tensorial objects. In direct comparison with the data model discussed here, it is a subset, but also provides information like neighborhood information on the tangential space which is not used in practice.

Data Explorer Model

Data Explorer (DX) model knows about the concept of a a `Grid` object and was actually one of the sources of inspiration for developing this model. It also uses a component model and describes a `Grid` object by a set of arrays:

P .... Positions (contains coordinates for each point)
N ..... Neighborhood (indices to neighboring points)
T ..... Tangential data (data on each point)
C ..... Connectivity (point indices for each cell)
E ..... Edges (indices of two consecutive points)
F ... Faces (series of point indices forming some area)

There may be additional arrays defining extra data referring to existing arrays (e.g. a set of faces), and additional arrays referring to elements of other arrays (e.g. a collection of sets of faces). This functionality can be used to construct cell complexes of arbitrary size and complexity and is therefore similar to this model. However, there is no further grouping or hierarchical relationship among these arrays in DX. In contrast, in the data model presented here the concept of "index spaces" is introduced, which allows a cleaner grouping of `Grid` components. Using the definitions of an array (def. 64) and compatibility among arrays (def. 67), we may inspect the DX components to find such relations. It is easy to see that arrays P, N and T are compatible:

$$\begin{aligned} P : I_0 &\rightarrow \mathbb{R}^n \\ N : I_0 &\rightarrow I_0 \\ T : I_0 &\rightarrow T(\mathbb{R}^n) \end{aligned}$$

Their index space directly refers to vertices (def. 2.2.1), i.e. an index space of depth 0 according to def. 68. Arrays C, E and F are mappings into the power set of this index space $\mathcal{P}(I_0)$, and thus have an index depth 1. C, E and F are not compatible, but compatible arrays with extra data can be added (i.e. when defining data on connectivity elements, edges or faces). However, this compatibility of arrays and semantic grouping is not obvious in the DX data model. Arrays of arbitrarily higher index depth may be added to a grid, but there is no such concept as a `Topology` or `Representation` group, all data arrays are stored plain within a `Grid` object . Their semantics is casted only into the *textual description* of an array. Summarizing, the DX data model is an excellent approach and superior to alternative implementations concentrating on just specific data and grid types. However, in direct comparison with the concepts as conceived here, the following functionalities are missing:

- no concept of index spaces and the programming safety induced by the concept of index depth

- no concept of topological spaces and mapping among topological spaces ("relative representations")

- no support of tangential objects with covariant and contravariant indexing (tensor fields are just specified by their rank, but not via upper and lower indexing), and the corresponding formulation of expressions similar to the Einstein sum convention

- no automated transformation rules and transformation paths (concept of chart objects, local and global charts), together with an high-level interface of unifying these with grid interpolation, grid evaluation and coefficient evaluation

- no support for coordinate-free formulations

POOMA

POOMA has been designed for numerical applications, where data are created from some initial source, like starting parameters or an original data set. The operations which need to be performed on this data set are known yet at program start and the task of POOMA is to execute these operations on the fastest way. This is a huge conceptual difference to an (interactive) visualization process, where unexpected interaction with some user may lead to previously unknown combination of data types. Consequently, the set of possible data sets and operations is *not* known at compile-time in an interactive visualization environment, whereas a numerical simulation may concentrate on a single set of possible data layout. As a result, data objects in POOMA are fixed at compile-time and static. It employs programming techniques from C++ template metaprograms to achieve maximal performance. The implementation of this fiber bundle employs similar technique, but faces an n^p problem when trying to support different n grid types during operations p grid objects. This problem is reduced by formulating grid objects through grid components, such that a huge amount of grid types can be expressed through just few different types of component layouts (e.g. procedural arrays vs. explicit storage).

As POOMA is an actively developed up-to-date application, we will discuss its concepts in relation to the fiber bundle model in more detail:

- **Dynamic vs. Static Construction:** A data model implementation for visualization requires more dynamic capabilities, thus allowing at a certain point to change the properties of objects at runtime.
- **Collection of objects vs. object hierarchy:** POOMA does not support tangential objects and coordinate transformations with associated tensor calculus and/or covariance concepts. Coordinate-free programming is thus not supported by POOMA.
- **Pointwise operators on regular meshes vs. Topological Iterators and Operators:** POOMA allows to define point-wise user functions as templates that may be instantiated over regular arrays. The fiber bundle model allows to implement and use similar concepts to build visualization algorithms as point-wise instantiations over various grid types via topological operators and iterators (3.2.2), but regular grids are just a special case. Moreover, algorithms can be generalized to topological spaces of higher index depth, i.e. the same algorithm can be used ("instantiated") cell-wise, edge-wise, face-wise etc. in addition to just point-wise operations.
- **Explicit vs. recursive handling of dimensionality:** The C++ implementation of the fiber bundle model defines multidimensional algorithms via recursion (see 3.1.2), while POOMA handles dimensions explicitly per dimension.
- **Arrays as Fiber Bundles vs. Arrays as discrete maps:** An array in POOMA is called a "multi-element data structure", each of whose elements is specified by one or more indices; they are ("small") fiber bundles each by itself. In the fiber bundle model, the indices of an array do not carry any meaning by themselves; their only purpose is to uniquely access the array's elements. They neither impose a topological structure by itself nor require some numerical ordering; any additional information about the dimensionality of such a data set is stored somewhere else. This allows many fields to share the same storage layout, like neighborhood and connectivity information.
- **Fields with geometrical information vs. geometrical information with fields:** A Fields is an extension of an array in POOMA which may provide geometry information. In contrast, Fiber Fields are not represented by single arrays but by a set of arrays, each of them providing a representation of the field concept. Geometrical information is shared among multiple Fields. It may also co-exist in various representations.
- **Absolute vs. relative (coordinate-specific) properties:** In POOMA, a "uniform mesh" is a certain class definition, which is then bound to a certain coordinate system. In the Fiber Bundle model, the uniformity is just the appearance of a regular grid in this certain chart, but other (non-uniform, but still regular) views may co-exist at the same time on the same grid object (an example is given in 4.3.3 when discussing the volume rendering of "lazarus data" sets).
- **Independent objects vs. object management:** In POOMA, objects are constructed independently. In the Fiber Bundle model, all such objects are inserted into a hierarchy descending from a central root node, the "Bundle" object. Relationships among objects are part of the fiber bundle hierarchy (formulated via relative representations), whereas POOMA does not support the formulation and storage of such relationships.

POOMA is surely one of the most advanced implementations to allow the specification of grid operations independent from its actual layout. It employs very modern techniques and is optimized to achieve high performance. Some concepts of POOMA are close to those of the Fiber Bundle model. However, from the viewpoint of the Fiber Bundle, POOMA only implements a subset of the desired functionality, although by very impressive technical means. It would certainly be possible to use POOMA as the implementation of various low-level tasks with only minor overhead. Such an interface has not been implemented, because at the time when the implementation process of this data model started, POOMA was not yet in a mature state. Another reason is the military background of POOMA, and it is highly questionable from the ethical point of view if one really wants to work with code that was originally developed and used for the design of nuclear weapons.

3.3.3 Correlation Tables

Each data model has its own terms for describing its concepts. The basic objects in

- Butler's Vector Bundle are point Sets, topological spaces, manifolds, vectorspaces, charts and chart transition functions on the manifold and vectorspaces, vectorbundles, and vectorbundle Sections;
- DX are fields, arrays, components, groups, multigrid groups, composite field groups, series groups;
- POOMA are fields, meshes, arrays, and layouts;
- the Fiber Bundle model are Fields, Representations, Topologies, Grids, Slices, Bundles.

We may roughly demonstrate the correlations among the mathematical terms and the objects in the Fiber Bundle data model in the following table:

Data Model Term	Mathematical Notation	Mathematical Description
Field	$v^\mu \partial_\mu$	stalk, section or field
Representation	$\mathbb{R}^n$	coordinate representations
Topology	X	topological space
Chart	$\{x^\mu\}$	map to coordinate space
Transformation	$\{x^{\bar{\mu}}\} \circ \{x^\mu\}$, $\alpha^{\bar{\mu}}_{\mu}$	transformation rules
Grid	$G_i = (X, \{x^\mu\})$	manifold
Slice	$\bigcup_i G_i$	time slice
Bundle	$\mathbb{R} \times \bigcup_i G_i$	fiber bundle

This mapping is not exact, since e.g. a Grid object can also be a non-manifold, but in most cases it will be a manifold.

The following table will give an overview of the relevant terms in each data model and its raw equivalent within the other data models. An empty entry means that the corresponding concept exists in the respective model, but it cannot be described by a single term and does not have a 1:1 mapping. Dashes "—" mean that the respective concept is not supported by the other model at all, although there might be hand-crafted workarounds to formulate it.

Data Model Term	Butler's Vector Bundle	DX	POOMA
Field	VectorBundleSection	component	array
Representation		positions	mesh
Topology	TopologicalSpace	neighborhood	array
Chart	Chart	—	—
Transformation	Transition	—	—
Grid	Manifold	field	field
Slice		group	
Bundle		series	

The correlation is only approximative, e.g. a POOMA mesh only corresponds to the positional component of a `Representation` object. Also, POOMA does not support the concept of multiple representations, so the terms are not comparable directly. On the other side, a series group in DX must be constructed by equivalent objects, while in the Fiber Bundle model a "Grid evolution" may be constructed from mutating objects, e.g. a sequence of isosurfaces with evolving connectivity. Of course, timelike interpolation is not straightforward in such cases and requires future and past cone arrays available for consecutive grid objects (see 3.1.2). If we inspect the different approaches considered here more closely in their relationship to the mathematical definition of a fiber bundle, def. 50, we find the following coincidences:

Math. term	Fiber Bundle	Butler's Vector Bundle	DX	POOMA
base space	Neighborhood, multiple Positions	Manifold	neighborhood, positions	mesh
fiber space	Fields in multiple Representations	VectorBundle	data arrays	array
total space	`Topology` object	VectorBundleSection	field	field

While the concept of a mathematical fiber bundle can be found in all cases, only the Fiber Bundle model described here and Butler's prototype support multiple representations of the same fiber in different charts. The Fiber Bundle model furthermore generalizes the specification of topological relationships among topological spaces in a manner similar to a chart and thus constitutes the most general non-trivial hull of all these approaches.

Chapter 4

Rendering Algorithms

The main concentration of this work is on symmetric 3×3 tensors, i.e. metric fields from 2.1.1. This includes also the ability to directly visualize other 3×3 tensors like the extrinsic curvature K or the Riemann tensor R, although the given interpretations of the visualization methods might not be physically appropriate for them. The idea is to start with discrete visualization methods, to apply them to large datasets and to modify them until they are suitable for displaying global structures and properties of the tensor field.

4.1 Vector Field Visualization

Before investigating tensor field visualization that involves many quantities per point (where "point" does not necessarily coincidence with the term "vertices of a grid", but also arbitrary locations in space), we will review the process of studying vector fields, which can be seen as "simple" tensor fields that use only three quantities per point. Visualization of vector fields has been topic of visualization efforts historically more than tensor fields. Its issues have extensively been discussed by D.Stalling [Sta98]. It might give inspiration how to do tackle the task of tensor field visualization by generalization of the methods to incorporate at least twice as much quantities per point for a tensor field.

Standard vector field visualization methods usually think of vectors as tangential vectors. However, this is not the only kind of rank-1 tensor, as there also are covectors. By identify vectors and covectors via musical isomorphisms in flat space (see 2.1.1), usually covectors are not considered separately. Since we think in flat space, this does not harm for visualization purposes because we can intuitively map vectors to covectors. However, from the view point of differential geometry there is a distinction among vectors and covectors, and their identification implies the existence of an additional structure beyond the vector field, i.e. a metric tensor field on the underlying space which determines the musical isomorphism. Thus it makes sense to think about visualization methods which do not imply such an additional structure.

4.1.1 Vectors

Icons As we know from school, "a vector is an arrow". Thus a straightforward visualization technique is to display arrows at each point in space where a (discrete) vector field is given. This technique is still the best for "debugging" vector field data and pointwise inspection. However, for getting an intuitive overview of a complete dataset, this technique is insufficient, because too many vector arrows are confusing and hide others laying in the same view direction (known as the "view occlusion" problem). Introducing thresholds (display only vectors which fulfill some certain condition, like their magnitude in a given range) or hyperslabbing (display vectors only in a plane instead of the full volume) helps, but does not solve the problem.

For pointwise inspection also of the local derivatives of a vector field the vanWijk-Glyph has been developed; it encodes quantities like the shear, curvature and something more of the vector field at a given point. However, it makes no sense to render a complete three-dimensional volume based on such glyphs, which are optimized for point-wise inspection only.

Integrating Methods An *integral line* $q \subset M$ on a vector field $v \in \mathcal{T}(M)$ within a spacetime manifold M with starting event $q_0 \in M$ is defined via

$$\dot{q} \equiv \frac{d}{ds} q(s) = v(q(s)) \quad \text{with} \quad q(0) = q_0 \quad . \tag{4.1}$$

Integral lines are also called *trajectories*, *tangent curves*[1] or *path lines*. They describe the path of a point-like particle in the flow of a vector field. In coordinates, q describes spatial and temporal information; usually only three-dimensional, but possibly time-dependent (non-stationary) vector fields are considered. Then equation (4.1) reduces to three equations

$$\dot{q}^a(s) = v^a(q^t(s), q^1(s), q^2(s), q^3(s)) \quad , \quad q^t(s) = s \tag{4.2}$$

whereby $a = 1, 2, 3$ describes spatial coordinates. However, when solving (4.2) by numerical methods, it is preferable to go back to (4.1) to get an autonomous system of ordinary differential equations which can be solved using Runge-Kutta methods[DB02].

Sometimes it is also of interest to investigate a vector field at some instance of time. We can do so by treating the vector field as being stationary, i.e.

$$\dot{q}^a(s) = v^a(q^t(0), q^1(s), q^2(s), q^3(s)) \quad , \quad q^t(s) = q^t(0)$$

yields lines called *field lines* or *stream lines* (see also [Sta98] for a detailed discussion of integration methods and Zoeckler et.al. [ZSH96] for a rendering technique). Stream lines correspond to the flow direction of many particles which are spread around in the volume of the vector field. Path lines and stream lines are both one-dimensional manifolds; they can't intersect themselves, since at each point their direction is uniquely determined by the given vector field (in contrast to e.g. geodesics). A streamline is a static object, all of its points belong to the same time slice, whereas a path line is constructed by points at different times. A path line can be seen as the projection of a streamline within an time-dependent n-dimensional manifold onto an $n-1$ dimensional submanifold time slice (a path-line is the three-dimensional "image" of a four-dimensional streamline) – this projection may intersect itself.

Beneath the inspection of lines that start from a single event q_0 we can also investigate the behavior of groups of lines that start from a set of events, e.g. some "initial seed" line $q_0(\tau) : I \to M$ with $I \subset \mathbb{R}$. The *integral surface* $S \subset M$ within a vector field $v \in \mathcal{T}(M)$ with initial seed line q_0 is then constructed from all integral lines that pass through an event on this initial seed line:

$$S = \{q : \mathbb{R} \to M, \dot{q}(s) = v(q(s)), q(0) = q_0(\tau)\}$$

It contains a natural parametrization $S(s, \tau)$ by the initial seed parameter τ and the integration length s, and carries an induced natural coordinate basis of tangential vectors $\{\vec{\partial}_\tau, \vec{\partial}_s\}$, whereby $\vec{\partial}_s \equiv \dot{q} = v$. The projection of an integral surface onto a static time slice Σ with $dim(\Sigma) = dim(M) - 1$ does not necessarily yield a manifold: it can self-intersect and therefore its projected natural parameterization $\pi S(s, \tau) : \mathbb{R}^2 \to \Sigma$ does not provide a bijective map $\Sigma \to \mathbb{R}^2$.

A commonly used choice is to use a timelike initial seed line $q_0(\tau) = (\tau, q^1, q^2, q^3)$ with fixed spatial coordinates q^1, q^2, q^3 (we can call such a seed line a "location", since it describes a point in space independent of time). The resulting integral surface will then be span by a timelike tangential vector ∂_t and a spacelike tangential vector ∂_s. For a fixed time coordinate t the projection of the integral surface into a time slice $dt = 0$ reveals a line, called a "*streak line*". It is formed by the location of all particles that have passed (or will pass) through a specific point $q_0(t)$ at some time t. For stationary vector fields, integral lines will be independent of time, and so streak lines will coincidence with stream lines.

Another choice is to use a spacelike initial seed line $q_0(\tau) = (q^0, q^1(\tau), q^2(\tau), q^3(\tau))$. The image of the seed line under evolution, the line $S(s, \tau)|_{s=const}$ is called a "*material line*" or "*time line*". The surface S is called a *stream surface*. An improved algorithm for computing a streamsurface based on the original algorithm by Hultquist [Hul92] was given by Stalling [Sta98].

Sometimes higher dimensional initial seed data are used, revealing surfaces or volumes evolving under the flow map of the underlying vector field (the geodesic evolution of a volume will be discussed in 4.3.1).

Feature Detection Integrating Methods as in 4.1.1 run into problems if the underlying vector field becomes zero at some points. Such points are called *critical points* and can be classified by inspection of higher-order derivatives, like the eigenvalues/eigenvectors of the Jacobi matrix[2] $J = (\frac{\partial v^i}{\partial x^j})$ of a vector field v. Via case-study of the Jacobi matrix eigenvalues critical points can be categorized [TWHS03] into topological equivalence relationships like "sinks", "saddles" and "sources". Rendering just critical points with their topologies (i.e. incorporating higher-order derivatives of the vector field) might be sufficient to visualize a complete vector field in a volume with only very few graphical elements and is thus a good approach for geometry reduction, facing the problems of view occlusion and rendering speed.

[1] Usually visualization literature does not distinguish among the terms "lines", a set of points (one-dimensional subspace) $q \subset M$, and "curves", parameterized one-dimensional maps $q(s) : \mathbb{R} \to M$, see def. 20. What is really meant here for visualization purposes is a *line*, while a *curve* is used for the computation and numerical representation of a line.

[2] Note that $\frac{\partial v^i}{\partial x^j}$ is a coordinate expression and reveals a tensor only if the Christoffel symbols $\nabla^i_{jk} = 0$ vanish in this coordinate system, otherwise the covariant derivative (def. 45) would be have to be used for inspection.

Color Coding A very successful technique for *scalar fields* is *volume rendering*, which maps the scalar value at each grid point to optical parameters like color, intensity, transparency etc. Common graphics hardware supports volume rendering (straightforward on uniform grids, other grid types requiring more effort like remapping to uniform grids or subdividing the input grid into a hierarchy of uniform boxes) via three-dimensional texture maps, such that interactive frame rates can be achieved easily. Better graphics hardware moreover supports interpolation of the data instead of just interpolation of the colors (known as the OpenGL colortable extension) and thus is able to display fine resolution details even beyond the original data resolution.

OpenGL-based volume rendering is able to display four quantities at a point: the red, green, blue and alpha (α) channel. Scalar field volume rendering usually works indirectly using some colormap to map a single (data) value to four (color) values. For vector fields, we already have three input values, such that using a colormap is not desirable and the vector component values can be mapped directly to colors, known as the *color coding* technique. Nevertheless we have multiple choices on how to map vector quantities to colors (we call this map a *vector colormap*):

- Map $(v^x, v^y, v^z) \mapsto (r, g, b)$ and $\vec{v} \mapsto \alpha = |\vec{v}|$. This map is straightforward and encodes the magnitude of the vector field as intensity, but does not take into account that colors range from 0 to 1 only. Zero-vectors are hidden, so we only see areas of large vector magnitude. Instead, we can also use a map $\vec{v} \mapsto \alpha = 1 - |\vec{v}|$ which only displays areas of small vector magnitude, thus prominently displaying critical points instantaneously. The color range can be fixed by using an arcus tangens function in the map, such that infinite values are mapped to one, $v_i \mapsto c_i = \frac{4}{\pi} \arctan(v_i)$, or by normalizing the color vector first, $v_i \mapsto c_i = v_i/|\vec{v}|$. A power exponent may also be introduced to allow contrast control.

- In the previous map negative vector components are all rendered black. This is not good, and we have the choice if we want to see the sign of the direction $v_i \mapsto c_i = \frac{v_i+1}{2}$ or just the direction $v_i \mapsto c_i = |v_i|$.

- An alternative approach is not to use the (r, g, b) color space but to employ the (hue, saturation, luminance) triple.

Although color coding of vector fields allow fast insight to an entire vector field volume, its drawback is that it is not easy to understand intuitively and requires some experience. Nevertheless it is an easy technique which can be applied to all vector fields which are given on geometrical objects, e.g. when displaying a vector field on the isosurface of a scalar field.

Texturing A superior way to encode vector fields on a surface than by color coding is using monochrome texturing which contains directional information. The method of *line integral convolution* (LIC) works by smearing white noise with the flow of a vector field. A fast implementation for two-dimensional LIC has been extensively discussed in [Sta98]. Three dimensional LIC has been used as well[IG97], creating three-dimensional monochrome texture maps containing directional information.

4.1.2 Covectors

The definition of the gradient of a function (2.28) involves a given metric. Using locally planar sheets of constant function value is a more direct way to visualize a function gradient, which does not imply some metric.

Covectors are often identified and even confused with vectors. Vector visualization methods can be used for covectors as well and provide an intuitively fast understanding. However, covectors have a different meaning than vectors and can be interpreted directly and visualized in differently in a more appropriate way.

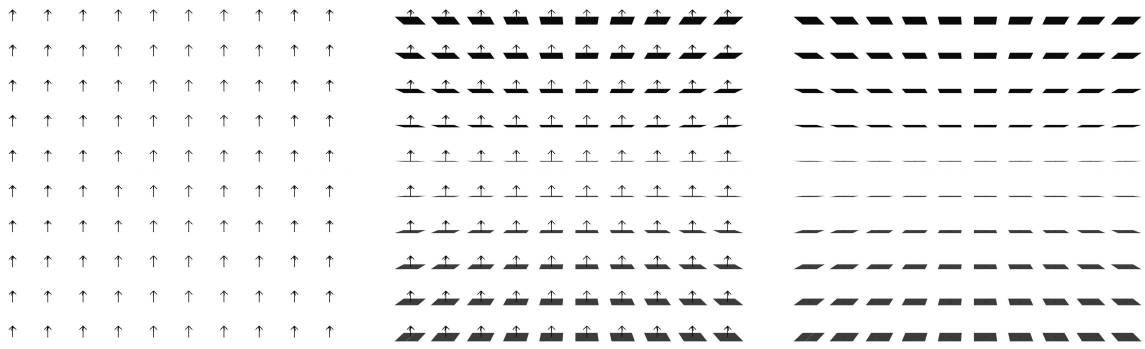
By definition (see also 2.1.1), a covector W is a function, which linearly maps a vector v to a number $W(v)$:

$$\begin{aligned} W : T_p(M) &\rightarrow \mathbb{R} \\ v &\mapsto W(v) \end{aligned}$$

An appropriate visualization of a covector is rendering the set of vectors that reveals the same number when using the Euclidean scalar product, i.e. with $V \in T_p^*(M)$:

$$V \cong \{\vec{v} \,|\, \vec{v} \cdot \vec{w} = V(w) \forall \vec{w}\}$$

The visualization of all vectors, that yield the same Euclidean scalar product with a fixed $\vec{w}$, is a plane (in general: a $n-1$ dimensional hyperspace on a n-dimensional manifold). A covector describes the change of a function f along a direction as specified by a tangential vector $\vec{v}$. A covector V can thus be visually imagined as a sequence of coplanar (locally flat) planes at distances given by the magnitude the covector, that count the number of planes which are crossed by a vector $\vec{w}$. This number is $V(w)$. For instance, if we chose the cartesian coordinate function x then a covector dx "measures" the "crossing rate" of a vector w in the direction along the coordinate line x.

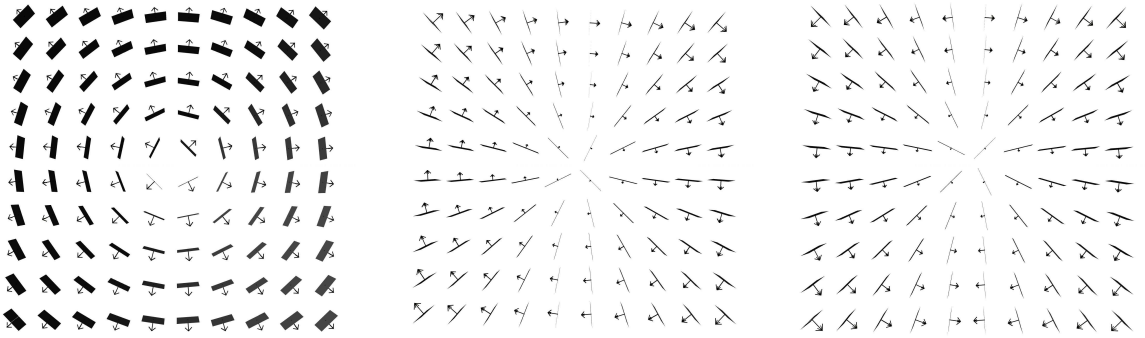


(a) Vector field ∂_z

(b) Duality relationship among ∂_z and dz

(c) Covector field dz

Figure 4.1: The trivial constant vector field along the z-axis viewed as vector field ∂_z and as covector field dz.



(a) Radial field dr

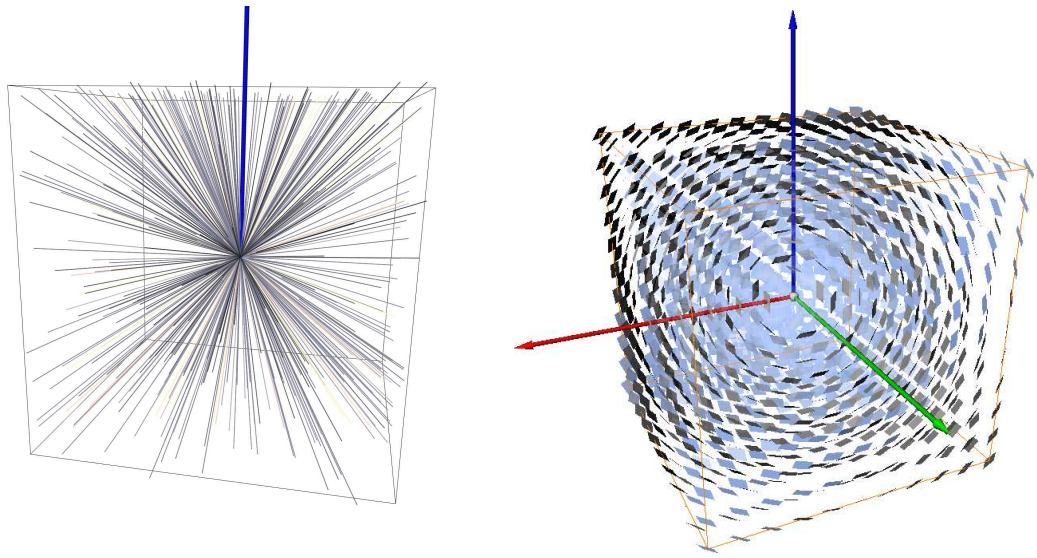
(b) Azimuthal field $d\varphi$, view of the equatorial plane (z-axis towards eye).

(c) Altitudal field $d\vartheta$, slice along the z-axis.

Figure 4.2: The polar basis covector fields.

4.1.3 Approaches for Tensor Fields

Usually visualization of tensor field concentrates on symmetric tensor fields of rank 2 in three dimensions. Only few approaches exist to deal with general rank 2 three-dimensional tensors like Demarcelle and Hesselink's widely known hyperstreamline technique [DH93] or A.Pang's deformation fields [BP98]. R. Kriz [KGM95] even investigated tensors of higher rank. Here, for now the term "tensor" will be limited to symmetric 3×3 tensors. Most visualization methods are constrained even more to the domain of positive definite tensors; we will call these tensor field "metric tensor fields" in accordance to def. 28 (but still keeping in mind that the underlying tensor fields might have nothing to do with a metric, i.e. it could be a stress tensor field, as well as a spacetime metric field in 4D is not positive definite, but only its spatial projection).



(a) ∂_r (b) dr

Figure 4.3: Radial vector and covector field.

Iconic techniques

The most straightforward way of visualizing tensors via ellipses (in 2D) and ellipsoids (in 3D) has been carried out very often. The capabilities of this method is discussed in 4.3.2. Alternative icons have been invented to enhance certain properties more clearly, like the Haber Glyph [Hab90], demonstrated in 4.3.2, and the Reynolds Glyph [MSM95], mentioned in more detail in 4.3.2. In section 4.3 a selection of newly invented or improved iconic techniques will be discussed, whereby the best methods allow a smooth, continuous overview of a three-dimensional tensor data volume (especially the newly invented "Tensor Splats" method 4.3.2).

Deformation Fields

As seen by eq. (2.15), providing an arbitrary vector field v, a tensor field G defines a new vector field w:

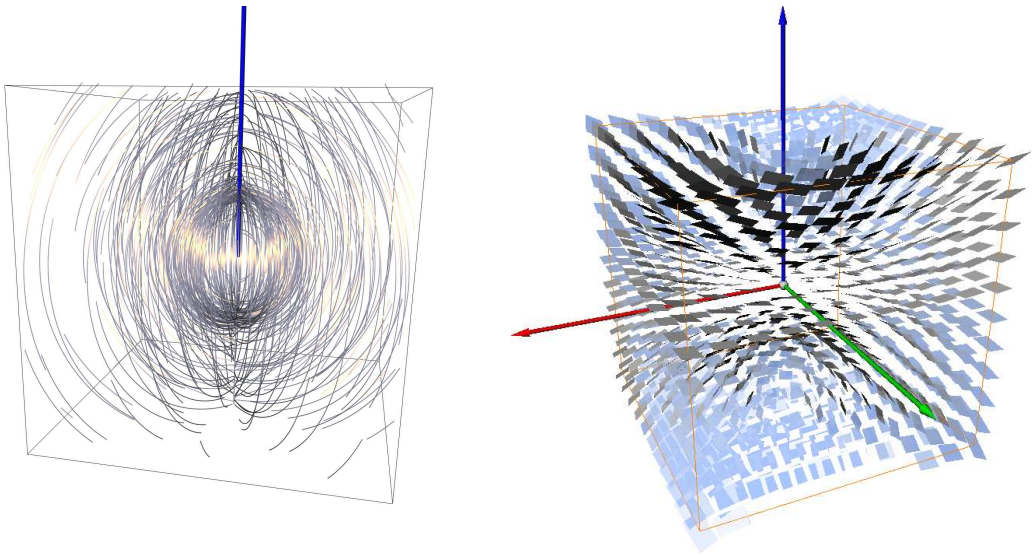
$$w := G(v, -).$$

Boring [Bor98] and Pang [BP98] concentrated on the idea of deformation fields, making use of the interpretation of a tensor field as a map from vectors to other vectors and compute the deformation of surfaces under the action of a tensor field. Each point P within a plane with normal vector $\vec{n}$ is mapped to another point $\tilde{P} = P + sG_P(n)$ with s a free scale factor. This technique also works for non-symmetric tensor fields and can be extended to volumes [ZP02], but is limited to display only a fraction of the tensor field at once as it requires some user-chosen input vector field. The full information content of the tensor field is not visualized until it has been probed with three linearly independent vector fields, resulting in three views that have to be mentally combined.

The advantage of this method is that it can be done interactively, is fast and intuitive. Tensor deformation can be applied to arbitrary surfaces, not only planar slices. It is thus useful to e.g. inspect spherical symmetry properties. The contiguous visualization results are a great plus for this technique. However, each visualization only displays a part of the tensor field, depending on the chosen surface. It cannot visualize a complete tensor field at once, each image by itself visualizes just three of six (or nine) tensor field components and is coordinate-dependent. Mental combination of separate images is require to get a complete view of the tensor field.

Integrating Methods

A problem common to all visualization techniques using icons is the problem of visual clutter. In vector field visualization the most direct way of drawing a vector arrow at each point in space is replaced displaying critical points and integral lines, like streamlines, streaklines and pathlines.



(a) ∂_ϑ (b) $d\vartheta$

Figure 4.4: Altitudal (latitudal) vector and covector field.

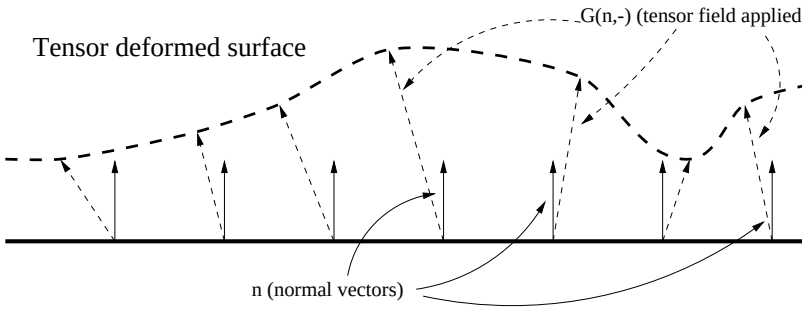


Figure 4.5: Tensor deformation surface: The result of a surface when the action of the tensor field on the surface normal vector field is computed.

The natural equivalent of vector field integral lines in a metric tensor fields are geodesics [Bry92]. Geodesic paths also depend on the derivative of the metric tensor field. A visualization of a geodesic bundle is a direct visualization of the Ricci tensor, which involves second-order derivatives of the metric tensor field. It is also known as Ricci focusing in gravitational lens theory. The consideration of lightlike geodesics also opens the path to the most physically correct and coordinate independent way of visualizing a metric tensor field, the computation of photorealistic images via raytracing in curved space [Yam89, Ben96, Wei00], but this approach is computationally very expensive and of limited usability when inspecting simulation data for numerical purposes. In the domain of scientific visualization the computation of geodesic paths is rarely used, on the one hand due to its low interactivity, on the other hand because it requires not only seed points (like for vector field integral lines) but also initial velocities, thus providing a large parameter space which makes automations hard.

Delmarcelle and Hesselink developed the technique of "Hyperstreamlines" [DH93]. These are streamlines of the maximum (or minimum) eigenvector "field" within a tensor field, but instead of drawing simple line segments the hyperstreamline is drawn with an elliptical cross-section that incorporates the minor and median eigenvalue, thus displaying five quantities in one geometrical object. The only left quantity, the maximum eigenvalue, is encoded as color along the hyperstreamline, thus depicting all six independent components of a symmetric tensor field. This technique is also able to deal with asymmetric tensor fields by sweeping a cross instead of an ellipsoid along the eigenvector streamline. The "rotation" of this cross along the streamline displays the antisymmetric part of the tensor field. Hyperstreamlines are implemented as a standard tensor field visualization tool in the Visualization Toolkit [Kit].

However, integration of the eigenvector "field" runs into problems in isotropic regions of a tensor field, where all three eigenvalues (or the two largest ones) are of the same size and the maximum eigenvector becomes ambigous/undefined (similar "isotropy artefacts" also happen with the previously mentioned glyph methods that are designated to enhance anisotropy) – an integral line of the maximum eigenvector field may therefore wildly jump from point to point. Weinstein et.al. invented Tensorlines[WKL99] to cure this problem by weighting the influence of the eigenvector field on a integral line by the isotropy of the tensor field at each integration point.

Like streamlines themselves, hyperstreamlines and tensorlines don't make much sense if the underlying tensor field does not provide meaningful integral curves, e.g. when large regions are isotropic or zero. Another disadvantage is their limited use for quantitative examination of two-dimensional data slices.

A "poor man's" raytracing of three-dimensional tensor fields was used by Pang [ZP03] to use integral lines of deviated vector fields as light paths, leading to lensing effects similar to relativistically moving refractive objects[LSP01].

Color Coding

The quality of integral lines strongly depends on the initial seeds and depicts just a subset of the complete data volume. Kindlmann, Weinstein et.al.[KW99, KWH00] investigated the capabilities of volume rendering for tensor fields by mapping of tensor field quantities to colors and shading parameters. This technique provides a broader overview of the entire tensor field but requires some learning of the visual effects and user interaction within the parameter space. Volume rendering can only display four quantities in a volume via mapping to RGBA, thus displaying only a (user-chosen) subset of the six tensor field components at once ("hue-ball" visualization technique). Additional information can be casted into reflection parameters, thus encoding anisotropy into anisotropic light reflection effects on a surface ("lit-tensor" visualization technique).

Texture pattern

Laidlaw[LAK+98] uses a technique inspired by the keystrokes of Van Gogh's oil paintings to draw two-dimensional slices of a tensor field by small normalized ellipses, encoding the absolute size of the projected tensor field as a stripe texture on the ellipses and drawing isotropic areas transparently. The maximum eigenvector component orthogonal to the slice plane is used to determine the saturation of the ellipse color/ The result is a texture pattern that provides a qualitative overview of the tensor field properties when viewed from a distance, but also provides quantitative details when inspected closely. However, this technique is limited to two-dimensional slices and intrinsically assumes that two eigenvalues of the tensor field are similar.

Sigfridsson et.al.[SEHW02] are using a technique similar to volume LIC[IG97] to smear some spot noise along the dominant directions contained in the tensor field. However, their technique appears to be computationally expensive and is hardly applicable to time-dependent data due to the random noise introduced (a problem common to LIC methods). Moreover, the artificial noise structures introduce visual information that is not really contained in the data set and is larger than one voxel, thus smearing eventual fine structures in the original data sets.

Tensor fields of higher rank

Riemann Glyphs [KGM95] are an interesting approach to visualize tensors of rank larger than two. Usual tensor field visualizations do not deal with tensors of orders higher than two. The idea here is to a "typical surface" similar to a quadric surface of a rank two tensor. For higher orders, we get surfaces of higher order as well, resulting in "bumped spheres" or "bumped ellipsoids". The location and amplitude of such a "bump" on the resulting surface stands for the properties of the tensor.

However, the resulting graphical object is not straightforwardly intuitive to interpret. As it is an iconic technique, it is hardly applicable to a three-dimensional collection of points, similar to the tensor ellipsoids. It has not been re-implemented here, but is mentioned because of its special approach.

4.2 Lamina Technique

Rendering transparent surfaces is of special interest for visualization purposes. The OpenGL rendering library allows to specify attenuation factors, called "alpha values", with each vertex of a surface element (e.g. triangles), which specify to which extend the background should shine through when rendering a new graphics primitive like a polygon. This functionality can straightforwardly be used to render the cells of a surface with constant transparency. However, the result is not visually pleasant because this process does not represent the intuitively imagined physical situation of a glass surface. The transparency of a

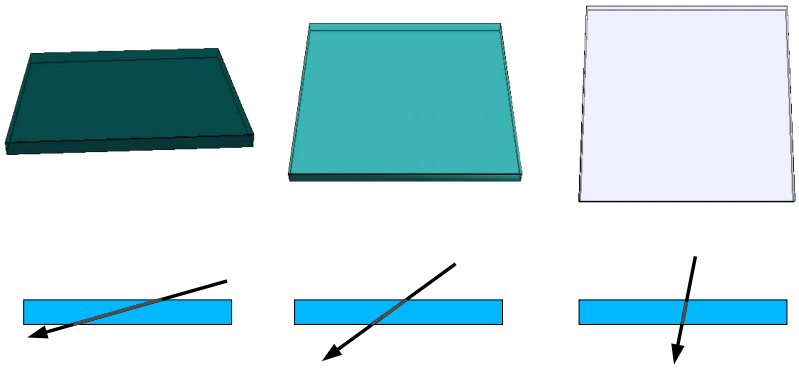


Figure 4.6: The transparency of a planar glass plate depends on the view angle.

true planar glass plate depends on the view angle (fig.4.6), because the length of the light path l through the glass medium with a thickness D will depend on the impact angle β via

$$l = \frac{D}{\cos\beta} \tag{4.3}$$

The transparency (fraction of transmitted light) of a material with optical depth σ is exponentially related to the length of the light path as (known as a solution of the radiation transfer equation):

$$T(l) = e^{-\frac{l}{\sigma}} \tag{4.4}$$

The transparency of a small glass plate, some "lamina" with constant optical depth, depends on the view angle via eq. (4.3) and (4.4):

$$T(\cos\beta) = e^{-\frac{D}{\sigma\cos\beta}} \tag{4.5}$$

For $\beta = 0$ (normal view) the transparency becomes a minimal value $T_{min} = e^{-\frac{D}{\sigma}}$. For $\beta = \pi/2$ (tangential view) the transparency becomes zero $T = 0$. The alpha value used by OpenGL is given via the transparency as $1 - T$ (an alpha value of zero corresponds to $T = 1$ or zero path length $l = 0$).

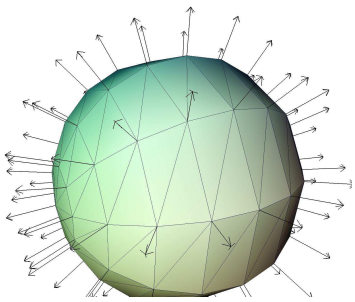


Figure 4.7: Let OpenGL allow to smoothly interpolate the transparency texture from vertices by specifying surface normal vectors as texture coordinates.

The OpenGL alpha values thus need to depend on the view direction for a realistic appearance. Ideally OpenGL rendering is done via "display lists", which allow to execute some rendering code very fast, but such rendering code must not contain any view dependency. A solution to this contradiction is shown by the technique of illuminated streamlines by Zoeckler,Stalling,Hege [ZSH96], which demonstrates a way how to simulate view dependent appearance while still rendering via OpenGL display lists. The idea is to make use of a texture matrix within a display list, but to encode the view point information $\vec{L}$ in the

texture matrix. The view-independent display list code will then make use of a view-dependent texture matrix, which is set up outside the display list as a function of the view direction $\vec{L}$ as:

$$\begin{pmatrix} L_x & L_y & L_z & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \tag{4.6}$$

This texture matrix results in a texture coordinate u as a function of the texture vector $\vec{t}$ which is proportional to the view angle:

$$u(\vec{t}) = \frac{1}{2}\left(\vec{L}\cdot\vec{t}+1\right) \equiv \frac{1}{2}\left(\cos\beta + 1\right) \tag{4.7}$$

whereby $u \in [0,1]$. The (cosine of the) view angle can be reconstructed from the texture coordinate u by the simple relation:

$$\cos\beta = 1 - 2u(\vec{t}) \tag{4.8}$$

The combination of (4.8) with the transparency (4.5) yields a one-dimensional texturing function $T(u)$, that can be directly fed into an OpenGL alpha texture (see fig.4.8) and results in much faster display-list supported surface rendering (table 4.1). To achieve using the texture function (4.7) at each vertex of

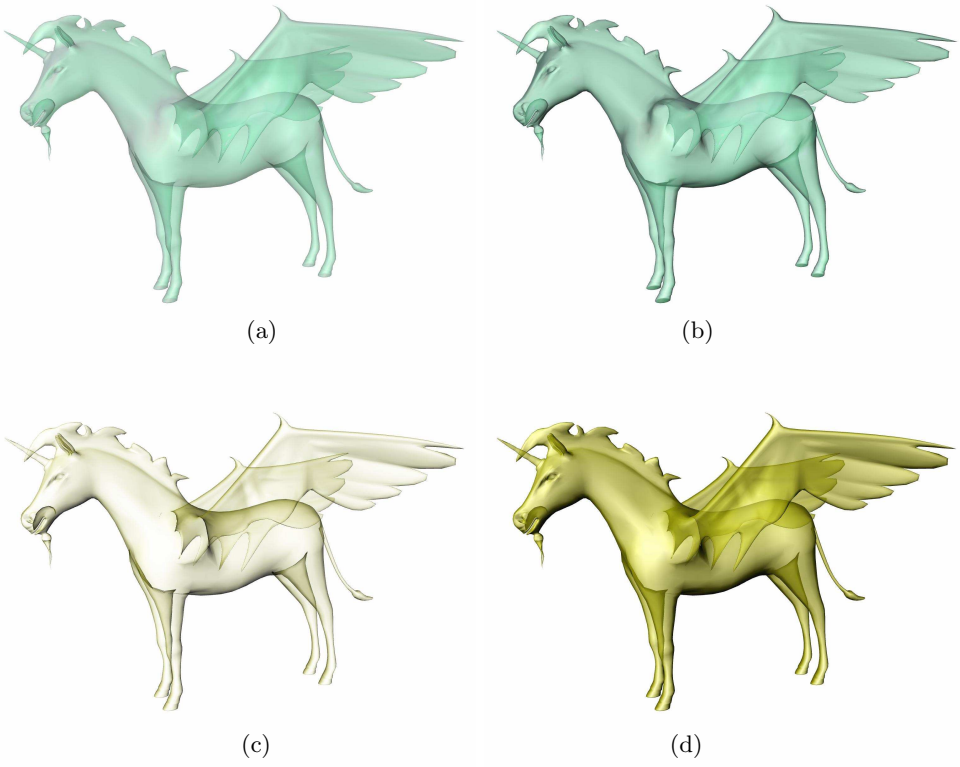


Figure 4.8: Rendering of a triangular surface with different methods. upper row: (a) view-independent constant alpha (possible via display lists) and (b)view-dependent "fancy" alpha (done by new rendering for each view direction) (upper row); lower row: fast rendering via display lists using a one-dimensional view-dependent alpha texture, with (c) 50% and (d) 100% scaling of the overall transparency.

a surface, the surface normal vector needs to be specified at each vertex as three-dimensional texture coordinate. The texture matrix (4.6) transforms this three-dimensional texture into a one-dimensional texture index. Hereby OpenGL also performs smooth interpolation of surface points between the surface vertices, while sustaining any color information. The lamina transparency can thus be used easily *in addition* to a color field that is specified on the surface (see fig.4.11).

Using (4.5) as texture function results in a physically correct appearance of surfaces built from a thin semi-transparent material. However, we are not forced to use exactly this formula, choosing another alpha texture function will not influence the rendering performance. Alternative view-dependent transparency

	Infinite Reality 2	SGI O2
Opaque Shading	70fps	3fps
Transparency w. depth sorting	14fps	1.2fps
Laminae technique via Display Lists	60fps	5fps

Table 4.1: Comparison of the lamina rendering technique with standard rendering. The achieved frames per second is excellent in all cases, while the lamina technique still allows high quality parametrized transparent surface rendering.

functions allow fine-tuning of the surface appearance via unphysical parameters. An empirically useful transparency function is given by:

$$T = \begin{cases} 0 & \cos\alpha \leq \varepsilon \\ 1 - (\cos\alpha - \varepsilon)^{\kappa} & \cos\alpha > \varepsilon \end{cases} \tag{4.9}$$

The two parameters ε ("edge visibility") and κ ("transparency falloff") allow smooth transitions from totally opaque surfaces via physically correct appearing surfaces like using (4.5) to a pure contour view of surfaces (see fig.4.9). Another interesting transparency function is $1 - T$ with T from (4.9). This

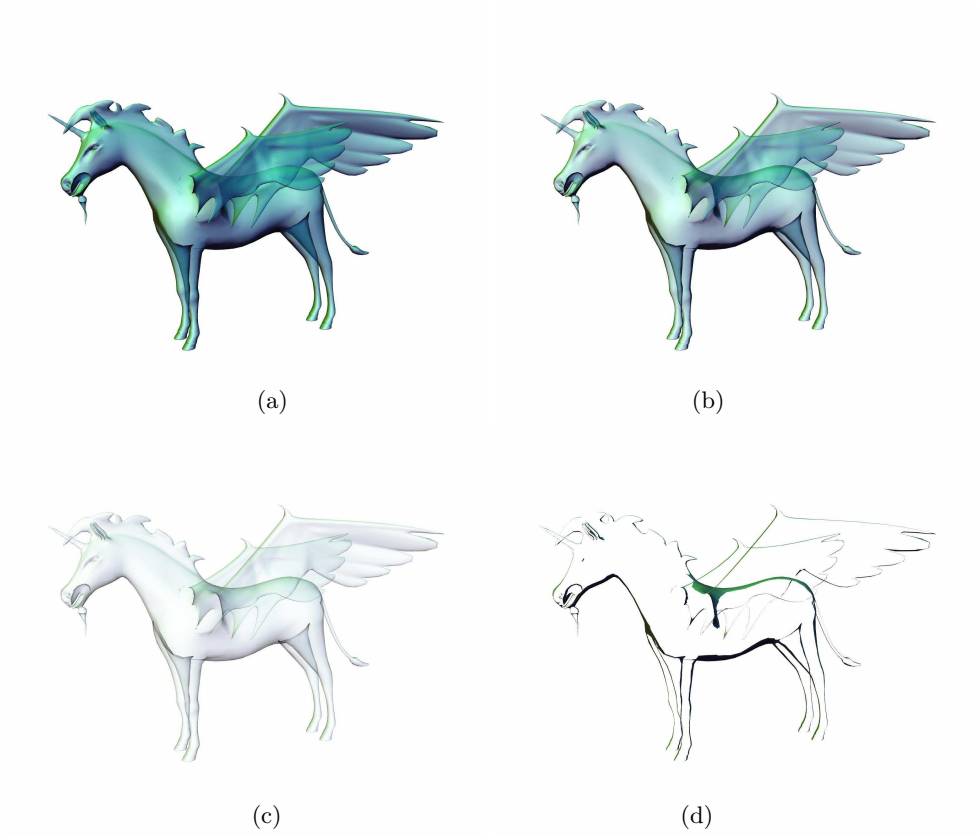


Figure 4.9: Variations of transparency parameters using the adjustable lamina transparency function 4.9: (a) emphasizing edges with $\kappa = 1, \varepsilon = 0.5$, (b) reduced transparency with prominent edges $\kappa = 0.5, \varepsilon = 0.5$, (c) reduced overall transparency $\kappa = 0.5, \varepsilon = 0$, (d) contour view with zero transparency but edge enhancement $\kappa = 0, \varepsilon = 0.5$.

transparency function results in dissolving contours, while view-orthogonal surface section stand out prominent. Surfaces rendered with such a transparency function look like objects filled with some cloud-like material, thus visually transforming two-dimensional surfaces into voluminous objects (fig.4.10).

In general, rendering transparent surfaces requires depth-sorting of the involved surface elements, because the OpenGL depth buffering in insufficient to track transparency information. However, for very transparent surfaces when mainly a surface contour is visible, depth sorting can be omitted. Depth sorting as a view-dependent operation is contradictionary to using OpenGL display lists. A compromise but still useful solution is to use multiple display lists, which correspond to selected view angles. They

Figure 4.10: Using an inverted transparency function $T \Rightarrow 1-T$ leads to an voluminous effect like filling an invisible surface with some cloud-like material.

are generated on demand an re-used when a view angle is within a certain deviation close to the exact view angle. This deviation angle can be some input parameter that allows the user to select the desired compromise between speed and precision. The exact solution would set the deviation angle to zero, thus practically disabling display list rendering at all. In practice, using a deviation angle of $\pi/2$, resulting in six display lists overall, is sufficient to achieve fast display list-enabled transparent surface rendering.

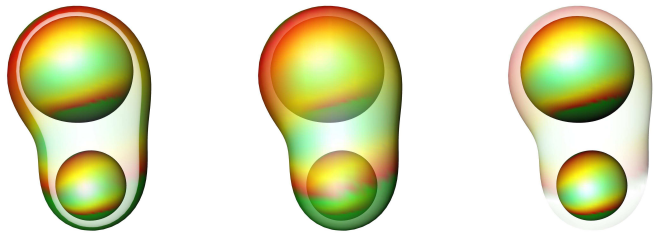


Figure 4.11: A colored surface with parameterizable transparency. (Data are the apparent horizons of just merged black holes with the gaussian surface curvature displayed as colors.)

Figure 4.12: Standard opaque surface rendering, left image, may be enhanced by combining it with the lamina surface rendering technique, right image. The lamina overlay results in a prettier image and may be used to emphasize the edges.

4.3 Studying Relativistic Tensor Fields

The central task of visualizing tensor fields is to depict multiple quantities instantaneously at each point of a volume. The number of independent quantities for tensors of rank 2 is demonstrated in the following table:

dim(M)	symmetric	antisymmetric	general
3	6	3	9
4	10	6	16
n	$n(n+1)/2$	$n(n-1)/2$	n^2

In the succeeding paragraphs a collection of various tensor field visualization methods is given, some of them as review of existing approaches, but also including newly developed techniques. Table 4.2 compares various tensor field visualization methods. Bold entries indicate methods that have been newly developed or re-implemented. The table is ordered according to the quantities within the tensor field which are used for the visualization. It is not possible to give an overall evaluation for these methods and to determine a "best" visualization method – each method has advantages that might be superior to others in a special case. E.g. tensor ellipsoids are straightforward to understand, but suffer under visual clutter problems. Tensor splats clearly display relevant features of a tensor field even in 3D volume, but require little learning and understanding of the visual effects - as with most tensor field visualization methods.

Nevertheless some criteria might help to select an appropriate method for a particular case or to estimate the benefits vs. costs of an implementation approach:

- **Number of quantities:** Does the method make use of the full information content of the tensor field or does it work by reduction to fewer quantities? Often methods displaying a reduced set include parameters that allow to browse other quantities as well, such that many images are required to get a complete impression of the tensor field. E.g. some methods also include the derivative of the tensor field; they can't display all quantities at once, e.g. geodesics only display a fraction of the tensor field depending on their initial data. We call such an input parameter set a "probe". It accompanies a visualization method in a graphical user interface. It may be implemented as a simple GUI slider or a complex graphical 3D dragger. Its purpose is to specify a subset of the tensor field via interactive means. A complete visualization of the tensor field is then provided by a set of images that contribute to multiple views based on different input parameters.

- **Robustness against visual clutter:** Is a method suitable for three-dimensional data volumes or is it limited to two-dimensional slices only?

- **Isotropy artifacts:** As mentioned earlier (p.50), visualization methods relying on eigenvectors usually suffer under eigenvector ambiguities in isotropic regions, thus leading to undesired apparently chaotic behavior. All eigenvectors are undefined and may point into any direction if all eigenvalues are equal ("major isotropy artifact"). A similar, but minor issue arises when two of three eigenvalues are identical, leading to two undefined eigenvectors within a plane ("minor isotropy artifact").

- **Limitation to positive definite tensors:** Can the method handle tensor fields with negative or zero determinant?

- **Limitation to symmetric tensors:** Can the method display asymmetric tensor fields?

As mentioned before, selecting an optimal choice is problematic due to various constraints outside the pure technical issues of a visualization method. Nevertheless, if some evaluation would have to be done, one of the best *general purpose* methods appears to be the "Tensor Splats" method (4.3.2), which has been developed as part of this thesis.

Algorithm Classification

This section relates the tensor field visualization work to the data model as described in chapter 3. Reading this section is not required to understand the succeeding text. However, knowledge of the concepts presented in chapter 3 is essential for comprehending this section.

In accordance with the concepts of Grid algorithms 3.2.2 and Grid operations 3.2.2 in the Fiber Bundle model the presented visualization methods can be categorized into:

Tensor Encoding	Q	Viz Method	$\smile$	$tr(G)$	"+"	sym
G_{xx}, G_{xy}, G_{yy}, G_{yz}, G_{zz}, G_{zx}	6	Quadric surface	bad	good	no	yes
		Metric ellipsoids	bad	good	yes	yes
		Geodesic emission	bad	good	yes	yes
		Reynolds Glyph	bad	good	yes	yes
$\vec{v}_{max}, \lambda_{max}, \lambda_{med}, \lambda_{min}$	6	Haber Glyph	bad	bad	yes	yes
		Tensor Schlieren	good	bad	yes	yes
G_{00}, G_{01}, G_{11}, G_{12}, G_{22}, G_{20}	6+3	**Tensor Cones**	bad	bad	yes	yes
G_{00}, G_{01}, G_{11}	3	Deformation Surfaces	bad	good	no	no
		Tensor Rain	good	good	no	no
G_{00}, G_{01}, G_{11}, $tr(G)$	4	Color Coding	good	good	no	no
$\vec{v}_{max}, tr(G), c_p, c_l$	6	**Tensor Splats**	good	good	yes	yes
$G(\vec{x}, \vec{x}), G(\vec{x}, \vec{y}), G(\vec{x}, \vec{y})$ $\vec{x}(\vartheta, \varphi), \vec{y}(\vartheta, \varphi)$	3-6	**Tensor Shadow**	good	medium	yes	yes
		Tensor Glow	good	medium	yes	yes
G_{xx}, G_{xy}, G_{yy}, $tr(G), \lambda_{max}$	5	Van-Gogh keystrokes	bad	good	yes	yes
$\vec{v}_{max}, \lambda_{max}, \lambda_{med}$, $\lambda_{min}, \partial\vec{v}_{max}$	6(9)	Hyperstreamlines	good	bad	yes	no
	6(9)	Tensor Lines	good	good	yes	no
$G_{mn}, \partial G_{mn}$	"24"	Geodesic	good	good	yes	yes
$G_{mn}, \partial G_{mn}, \partial\partial G_{mn}$	"78"	Geodesics Bundle	good	good	yes	yes

Table 4.2: Tensor field visualization methods comparison table. The entry "$\smile$" stands for the robustness against visual clutter, $tr(G)$ indicates how far the methods suffers under eigenvectors ambiguities in isotropic regions, "+" shows if the method requires the tensor field to be positive definite and "sym" stands for requiring symmetric tensor fields.

- **Vertex Fiber Grid Algorithms:** Methods which inspect each point and draw something there. A typical example are all iconic techniques. Although iconic techniques are usually problematic, because they suffer under view occlusion difficulties in 3D data sets and thus hide the perception of global properties, a lot of option exists to make them powerful. to heraushol eine menge.

- **Cell Fiber Grid Algorithms:** Methods which traverse each cell of a grid and draw something per cell; the chances to get a smoother result than by vertex traversal are good, because cells touch and occupy a finite volume ins space, whereas a vertex is infinitely small by definition. Once the problem of being compatible at touching cell faces is solved, the result will be smooth. A typical example is the computation of isosurfaces.

- **Integral Operations:** Methods that start at some point, traverse to neighboring ones and create some smooth geometrical object, which is casted into a new grid object. An example is the computation of geodesics, which is the evolution of a particle system or some surface.

The advantage of Fiber Grid Algorithms is that they operate on some index space (e.g. on the vertices or on the cells of a grid), which can be traversed partially as well as the full index space. The configuration of the index space fraction is identical for all Topological Operators (which actually perform the graphical operations) and depend only on the Topological Iterator, which is grid-dependent. The work of "configuring an Iterator" thus needs to be done only once per Iterator type and is available immediately for all visualization methods (and all of them can be configured through the same user interface as well). Possible Iterator configurations are:

- **Hyperslabbing:** Only applicable for regular grids: select a regular hyperspace from within the entire grid, e.g. via border with and subsampling

- **Point Inspection:** Select a single point in space via some interactive coordinate selection tool (e.g. some 3D dragger, which may be driven by some 3D mouse device in some VR environment) and display a detailed vertex icon at the grid location. A modified version may also easily display multiple locations at once and e.g. show various icons along the history of a user defined path

- **Randomization:** Given the number of inspection points, display randomly distributed points on the grid. This is useful for data mining on very large datasets, where displaying the full data set is not possible, but some subsampling without aliasing and/or moire-effects is required.

- **Scalar Field Threshold:** Given an arbitrary scalar field on each point, display icons only on those points where the scalar field is within a given range.

- **Time Limitation:** Stop the rendering process if it exceeds a given time limit. It ensures defined reaction time of the applications and retains interactivity, which is of certain importance for VR purposes, large datasets and huge parameter space inspection. It assumes that the computational effort per point is constant, which is not necessarily true, but is still a good estimated. However, when OpenGL display lists are employed, time limited rendering becomes troublesome, because the primary rendering – which will be used for determining the allowed number of points – takes much longer than succeeding renderings. A possible solution is to adjust the number of points after a display list rendering has been performed and to remeasure it again, but this method may lead to multiple iterations or even cycles, which become very confusing to the user.

  Time limited rendering requires configuring an iterator to perform operations on just a subset of the data, the hyperslabbing or randomization configurations serve well here.

- **Evolutions:** Instead of traversing points spatially, also the same point index can be inspected for different time steps and fed to the Grid Operator, thus providing a clue for the dynamic evolution.

- **Screen-Sorted Traversal:** Transparency is not handled sufficiently by OpenGL hardware graphics. The workaround is no sort the graphical primitives to perform back to front rendering. An Iterator Object can be used to reorder the points (vertices, cells) of a Topology Objects according to their coordinate information in a certain chart and to call a Topological Operator on this reordered set of points. Thus any Topological Operator that has been developed for direct rendering of a Topology Object can be re-used to perform rendering with depth-sorting. A typical example is an Operator which renders opaque primitives (e.g. surface triangles or iconic objects), and its variation to include transparency (e.g. a transparent triangular surface or transparent icons).

4.3.1 Non-local Algorithms

Eigenvector Streamlines

As the maximal eigenvectors of a tensor $G \in T_p^{*2}(M)$ play an important role, it is a straightforward approach to employ vector field visualization methods to inspect them. The integration of maximal eigenvectors as streamlines is the basis of one of the widest known tensor field visualization techniques called hyperstreamlines by their inventors Demarcelle and Hesselink [DH93]. However, there are two relevant aspects which distinguish a "field of eigenvectors" from a true vector field:

- The maximal eigenvector is undefined in isotropic regions. Its direction is ambiguous and may vary due to slight numerical instabilities.
- The sign of eigenvectors is undefined, since $-\vec{v}$ is a solution of the eigenvalue equation $G \cdot \vec{v} = \lambda\vec{v}$ as well.

In contrast to a real vector field where all components are uniquely defined at each point in space, the eigenvector ambiguities can only be resolved via additional information. Thus the term "eigenvector *field*" should only be used in quotes, there is no true "vector field" derivable from eigenvectors of a tensor field. Unmodified vector field visualization methods would find and display features which are not a property of the data field but stem from the numerical eigenvalue extraction algorithm (isotropy artifacts). Modified interpolation and/or integration methods are required for eigenvectors:

- When interpolating eigenvectors within a cell, all vectors contributing to the interpolation must be oriented such that they point into the same half-space, i.e. $\vec{v}_i \cdot \vec{v}_j \geq 0$.
- Interpolating eigenvectors yields different results than interpolating the tensor field with computation of the eigenvector at each interpolation point.
- Streamline integration advances a point $q(s)$ of the streamline q to the next point $q(s + ds)$ by a small step size ds via
$$q(s + ds) = q(s) + ds\ \dot{\vec{q}}(s) \quad ,$$
whereby the new tangential direction is the direction of the vector field $\vec{v}$ at the point of interest $q(s)$
$$\dot{\vec{q}}(s) = \vec{v}|_{q(s)} \quad .$$
Here, $\vec{v}$ is the solution of the eigenvalue equation $G\vec{v} = \lambda\vec{v}$ at the point $q(s)$ such that $\vec{v} \cdot q(s) \geq 0$. This last condition is essential and needs to be added to a usual streamline integration algorithm.
- Orienting the sign does not cure the problems arising from isotropy artifacts. Streamlines of the maximum eigenvector only lead to reasonable results in regions of high linearity (see p.51). An alternative, less vulnerable integration algorithm is thus to start streamline in a region with high linearity and to advance it according to the deviation vector as given under the action of the tensor field[3]:
$$\vec{v} = \sharp G\left(\dot{\vec{q}}, -\right) \quad \rightarrow \quad q(s + ds) = q(s) + ds\ \vec{v} \quad . \tag{4.10}$$
Weinstein et.al.'s Tensorlines algorithm [WKL99] combines this method by blending the oriented maximal eigenvector $\vec{v}_{max}$ and the deviation vector with the shape factor c_l at point $q(s + 1)$:
$$\vec{v} = c_l\ \vec{v}_{max} + (1 - c_l)\left[(1 - w)\ \dot{\vec{q}}(s) + w\ \sharp G\left(\dot{\vec{q}}(s), -\right)\right] \quad .$$
Hereby w is a user-controlled parameter in the range $[0, 1]$ which is said to be selected depending on the type of data.
- Both integral lines as solution of (4.10) and tensorlines don't provide an unique direction at each point in space, thus intersections of lines may occur - in contrast to non-intersecting streamlines based on true vector fields.

[3] Note that in Euclidean space, the right side is just a multiplication of the tensor's matrix representation and the vector, we may also write "$G \cdot v$" in this case. In general, the metric of the underlying space needs to be involved via the $\sharp$-operator (see p.25), e.g. when performing this integration on a sphere or other curved surface.

Taking into account all the limitations that distinguish "fields of eigenvectors" from actual vector fields, the best we can still do is to

- start streamlines in regions of high linearity c_l (see 2.3.2)
- set transparency along the streamlines proportional to the anisotropy, using sphericity c_s or planarity c_p (if the two largest eigenvectors are ambiguous harms the same as if all three are undefined)

An example is given by fig. 4.13. These eigenvectors are not oriented. Thus, we see artifacts in streamlines "turning" around when crossing a coordinate plane. Actually, the maximum eigenvector field is just $\vec{\partial}_r$ here, so we would expect only radial streamlines.

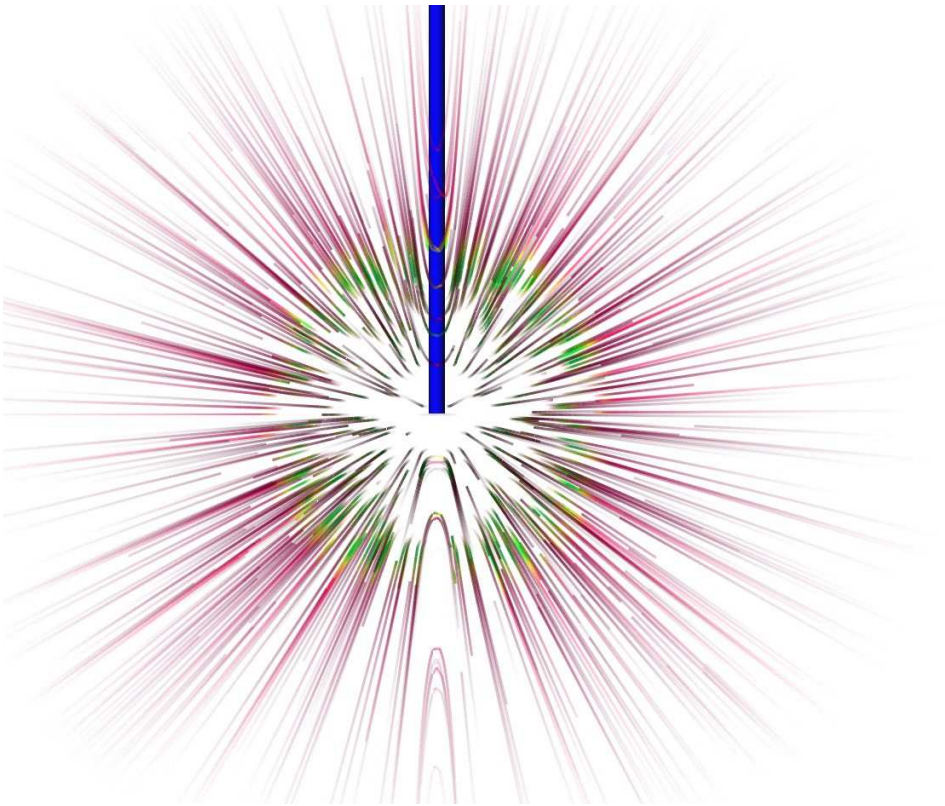


Figure 4.13: Vector field stream lines applied to the field of maximum eigenvectors of the Kerr metric of a rotating black hole. Colormap and distribution density are set from linearity, transparency is set from sphericity.

As the numerical eigenvector computation algorithm does not know anything about spherical or axial symmetry, it orients the eigenvector along a coordinate axis, e.g. x. Consequently, the radial eigenvector is oriented "left to right", i.e. pointing radially inwards left of the coordinate origin and radially outwards right of it. This behavior leads to the visible "turnaround" at $x = 0$. Beside this artifact, the dominantly radial structure of the spacetime is clearly depicted. However, streamlines are not able to display planar regions appropriately. Color and transparency coding is limited to indicate the validity of streamlines, but give no notion of a second direction where one might be of relevance. Especially, eigenvector streamlines are *not* able to depict the anisotropy features in the Kerr metric as in fig. 4.13.

A similar behavior is depicted by fig. 4.14, discovering linear grid stretching in the metric tensor field of two colliding black holes. Still, we find the eigenvector orientation coordinate artifact. As the black holes are colliding, they drag around their space with them. This feature is indicated by the eigenvector streamlines morphing from a radial into spiral-like shape. However, the eigenvector streamlines don't provide information about the actual geometrical properties of the spacetime, although they might provide some clue about it.

Summarizing, maximum Eigenvector stream lines provide a visually attractive and quite clear view of the prominent features of a three-dimensional tensor field. On the other hand, they contain some danger of appearing "too pretty", as they might falsely suggest a flow; however, a tensor field does not describe a "flow", but just "the possibility of a flow". An actual flow as determined by the tensor field may occur in more than one direction. This property of a tensor field is not depicted by eigenvector stream lines.

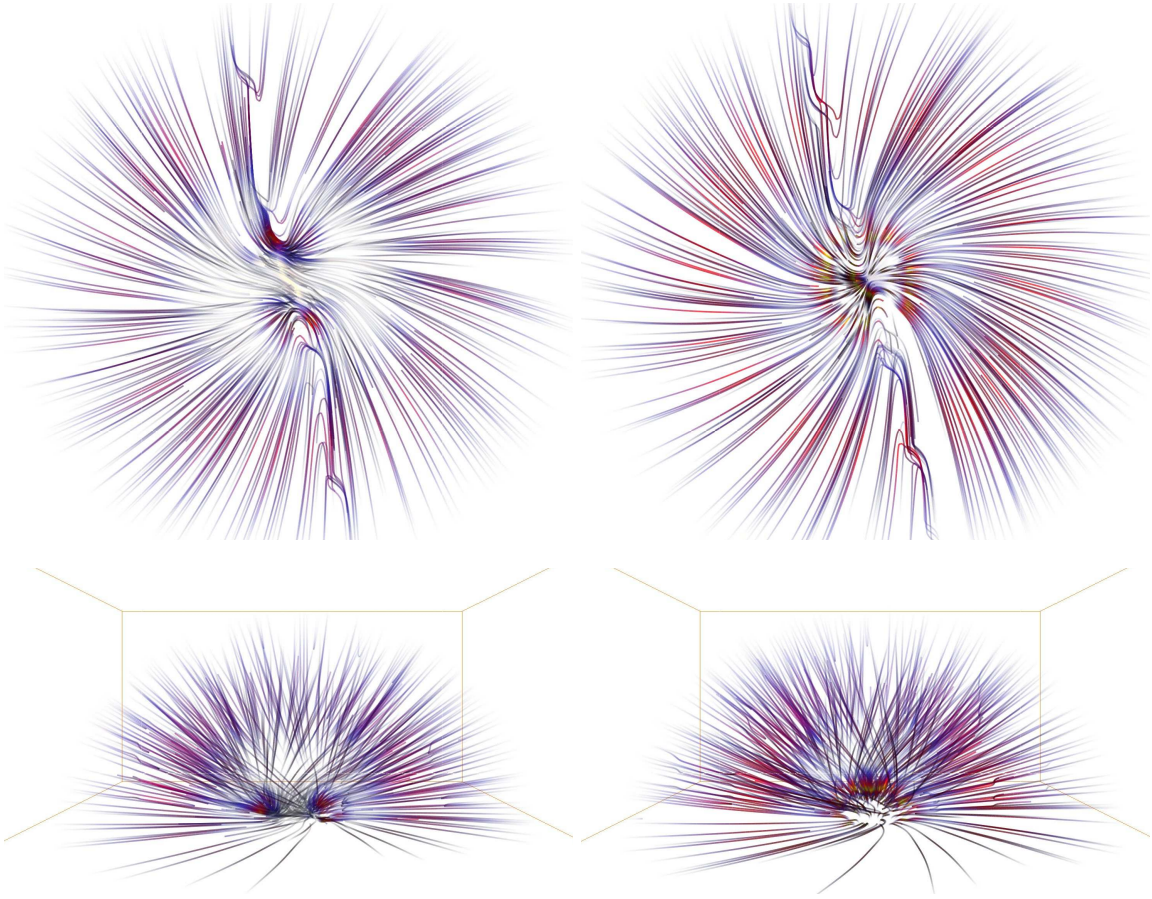


Figure 4.14: Streamlines of the maximum eigenvector "field" of the metric tensor field of colliding black holes ("Discovery data"), top and side view at time units $T = 25$ and $T = 35$. They provide a rough, but incomplete overview of the data set. Strong artifacts arise due to the arbitrary flips of sign of the eigenvectors.

Nevertheless, as long as this limitation is kept in mind, this rendering method provides an effective way of visualizing some (but not all) important properties of a tensor field.

Summary:

+ Useful for rough overview and feature detection.
+ Builds on existing vector field visualization methods.
• Provides 3D views, but not applicable to surfaces (e.g. slices).
- Provides only incomplete information, but danger of looking "too pretty".
- Must not be used as the only method of interpreting a tensor field.
- Streamlines are not applicable to time-dependent data.

Tensor Rain

A tensor field may be studied under its action on a vector field. A nice descriptive image coining the name "tensor rain" is rain falling down in parallel stripes that is diverted by some gust of wind. Here, the tensor field plays the role of the wind and the rain is some arbitrary input vector field. Our interest is in the variations of the tensor field itself, but we explore it by inspecting the deviated vector field, the "rain" instead. As this result is a vector field, we may employ a large variety of well-experienced vector field visualization techniques. Mathematically, we make use of the duality properties (2.14) of the tangential and co-tangential space. At first, we can interpret an arbitrary tensor field $g \in T_p(M) \otimes T_p(M)$ as a function that maps a vector to a covector:

$$\begin{aligned} g : T_p(M) &\longrightarrow T_p^*(M) \\ v &\longmapsto g(v,-) := V \in T_p^*(M) \text{ such that } \forall w \in T_p(M) : < V, w >= g(v,w) \quad . \end{aligned}$$

The coordinate expression is $g(v,-) = g_{\mu\nu} v^\nu dx^\mu$. Thus, the application of a tensor field onto an (arbitrarily chosen) vector field yields a covector field, which is computed by index contraction. We may employ the Lamina technique to directly visualize the resulting covector field as in fig. 4.15. The Laminae

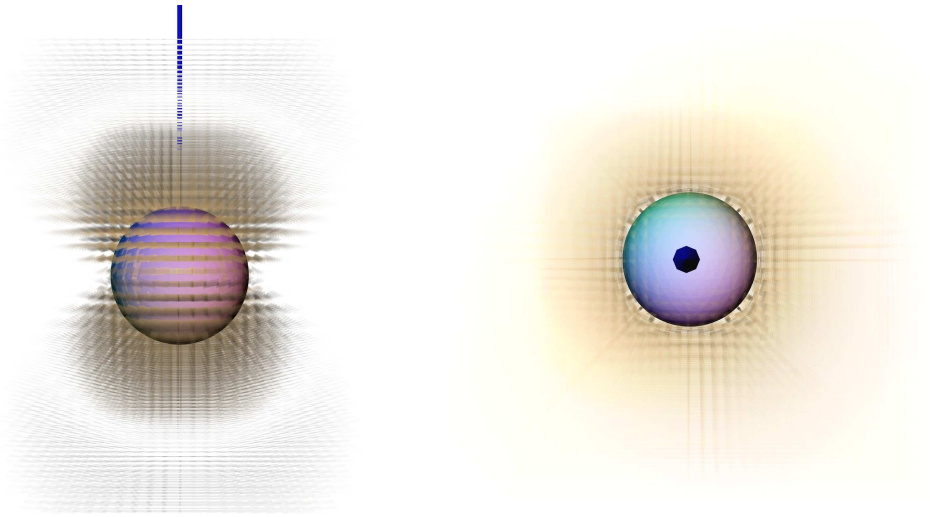


Figure 4.15: Covector Laminae applied to the Kerr metric probed with the ∂_z axial vector field, seen from the side (left) and from the axis (right).

covector visualization clearly enhances deviations from the "zero"-action of the tensor field such that we only get visual structure where the tensor field deviates from the flat-space metric.

The covector field may also be mapped to a vector field u via index raising by employing the musical isomorphisms 2.1.1 in flat rendering space $u = \sharp g(v,-)$. We thus get some input data appropriate for vector field visualization algorithms. In regions where the tensor field is isotropic, the direction of the probing vector field v will be sustained, in anisotropic regions it will be deviated. This method is most useful for parallel input vector fields, but other vector fields like a radial vector field ∂_r may be used to study the deviation from a certain symmetry (e.g. radial symmetry). Depending on the direction of the infalling "rain", different output patterns arise due to different deviation produced by the tensor field, e.g. fig. 4.16. By using a vector field as some input data, the number of visible tensor field components is reduced. The resulting output vector field contains three of the six components for a metric tensor field (or three of ninefor an arbitrary tensor field - there is no restriction of this method to metric tensor fields). The "vector field probing" is a projection of the tensor field in the direction of the vector field; e.g. if the input vector field is ∂_x, then the resulting vector field will be constructed by the components (g_{xx}, g_{xy}, g_{xz}). Consequently, inspecting a tensor field with a single vector field probe is insufficient to really detect all features of the tensor field. For a metric tensor field, at least a second vector field probe has to be used, an arbitrary rank two tensor field requires three vector field probes. While a single instance of a "tensor rain" visualization still contains "artifacts" due to the arbitrary projection operation, cycling between multiple input fields finally discovers all features of the underlying tensor field – although not in a single vector field visualization image, but in the combination of them, either by overlaying multiple images in the screen or in the user's human brain. Interactivity and smooth transition among intuitively understandable input vector fields is an important issue here.

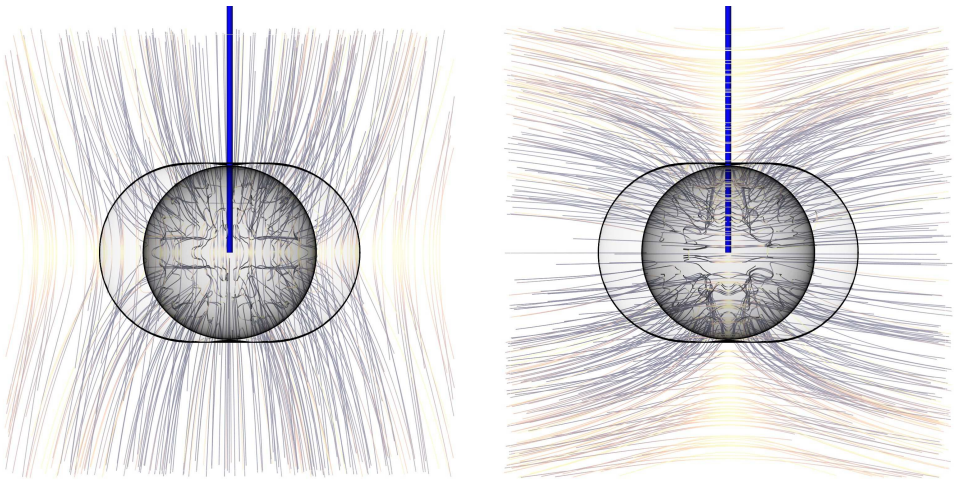


Figure 4.16: Streamlines of the Kerr metric probed with the ∂_z axial vector field (left) and the ∂_x coordinate vector field (right).

The advantage of the vector field probing ("tensor rain") method is a contiguous visualization of the tensor field in 3D if good vector field visualization methods like streamlines are used. Its disadvantage is that this method is not good for still images, like for publications in papers, because an understanding of the complete tensor field requires user interaction.

Possible extensions Certain extensions on the vector field probing technique are possible. Two ideas that might be worth trying are mentioned, although they have not been implemented yet:

Tensor fields of higher rank The idea of vector field probing could also be extended to tensor fields of higher rank by involving multiple vector fields: Let the provides two probing vectorfields (e.g. by specification of two directions, which could be visualized as light from a red and a green source), then visualize the effects of an tensor field acting on these vectorfields. E.g. when the tensorfield is of order two, the result is a scalar field which can then be visualized by ordinary volume rendering. A tensor field of order three results a vector field, which can be visualized with usual vector field visualization techniques, like streamlines. Even a tensor field of order four, like the Riemann tensor, can be visualized, as the result of the interactive probing with two vectorfields are tensor ellipsoids (or any other second-order visualization techniques). In the case of the Riemann tensor, it would be zero at large distances from a black hole (asymptotically flat space), but close to it "something" would happen to the probed tensor ellipsoids, depending on the input fields.

Acceleration field A metric tensor field determines acceleration on a vector field $v \in \mathcal{T}(M)$ or a curve $q : \mathbb{R} \to M$:

$$a = \nabla_v v = v^\mu{}_{;\nu} v^\nu \partial_\mu = (v^\mu{}_{,\nu} + \nabla^\mu_{\sigma\nu} v^\sigma)\, v^\nu \partial_\mu = (\ddot{q}^\mu + \nabla^\mu_{\sigma\nu}\, \dot{q}^\sigma \dot{q}^\nu)\, \partial_\mu \quad , \tag{4.11}$$

see def. 46. For freely falling particles, the acceleration vanishes, i.e. $a = 0$. In an inertial system, the connection coefficients $\nabla^\mu_{\nu\sigma} = 0$ vanish, and the acceleration is just $v^\mu{}_{,\nu}\, v^\nu \partial_m u$ and the second derivative of a curve becomes $a = \ddot{q}^\mu \partial_m u$. However, in general Christoffel symbols will be non-zero and act like a "virtual force" that depends on the velocity of some particle in the given coordinate system. This "coordinate acceleration" $\ddot{q}^\mu$ of a freely falling particle with $a = 0$ corresponds to

$$\ddot{q}^\sigma = -\nabla^\sigma_{\mu\nu} v^\mu v^\nu \quad . \tag{4.12}$$

This "coordinate acceleration field" $\ddot{q}^\sigma$ is not a tensor field. It may vanish in one chart while in another chart it is non-zero. Visualizing this acceleration field a depending on some input vector field v is thus a visualization of the connection coefficients (def. 2.40). In the case of a Levi-Civita connection it depicts the derivatives of the metric tensor field. Any vector field visualization method can be used.

Geodesics

The world line $q(s)$ of a particle with mass m in a spacetime is described by the solution of Newton's law

$$m\nabla_{\dot{q}}\dot{q} = F$$

whereby F is an external (non-gravitative) force (see also def. 46). If no external forces, i.e. only gravitational forces, are present, then the particle is said to be free-falling and obeys the geodesic equation (2.35). In a coordinate system we get a second order differential equation:

$$\ddot{q}^{\mu} = -\nabla^{\mu}_{\sigma\nu}\ \dot{q}^{\sigma}\dot{q}^{\nu} \qquad ,$$

thus a geodesic is uniquely defined by a starting point and an initial velocity. These properties distinguish metric geodesics from integration lines vector field. In particular an infinite number of geodesics may pass through a certain point in space, i.e. geodesics may cross whereas integration lines of a vector field cannot.

In a pseudo-Riemannian spacetime, we distinguish among timelike geodesics that are defined by $g(\dot{q}(s), \dot{q}(s)) = 1\,\forall s$ and null geodesics with $g(\dot{q}(s), \dot{q}(s)) = 0\,\forall s$. Spacelike geodesics have a physically relevant meaning only in static spacetimes. The timelike geodesics describe the path of particles with non-zero rest mass; the path depends on the initial 3-velocity (the energy) of the particle. Null geodesics describe the path of light and depend on the initial direction only. Particle trajectories thus involve a larger initial data parameter space. A straightforward but not unique and highly coordinate-dependent choice is to set particles at rest in a certain coordinate system, example given in fig. 4.17.

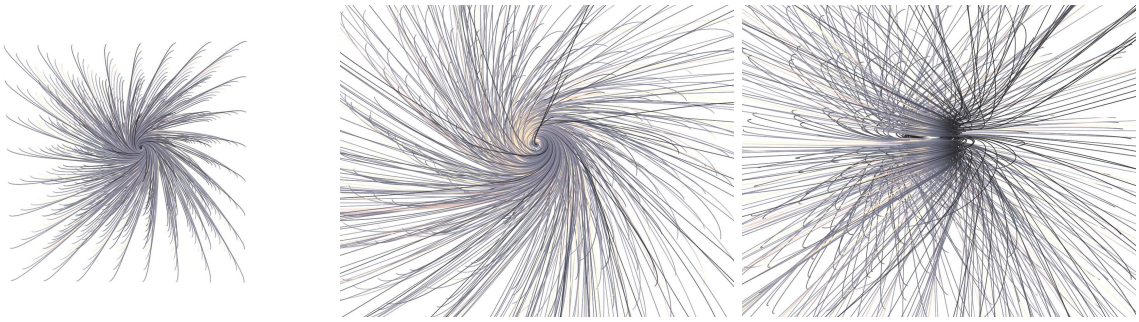


Figure 4.17: Timelike geodesics in the Kerr metric, particles are uniformly distributed in space and start with zero velocity. Their paths visualize the rotational drag of the spacetime, known as "gravitomagnetism" that lets particles "feel" the rotation of a mass. This effect is absent in Newtonian gravity.

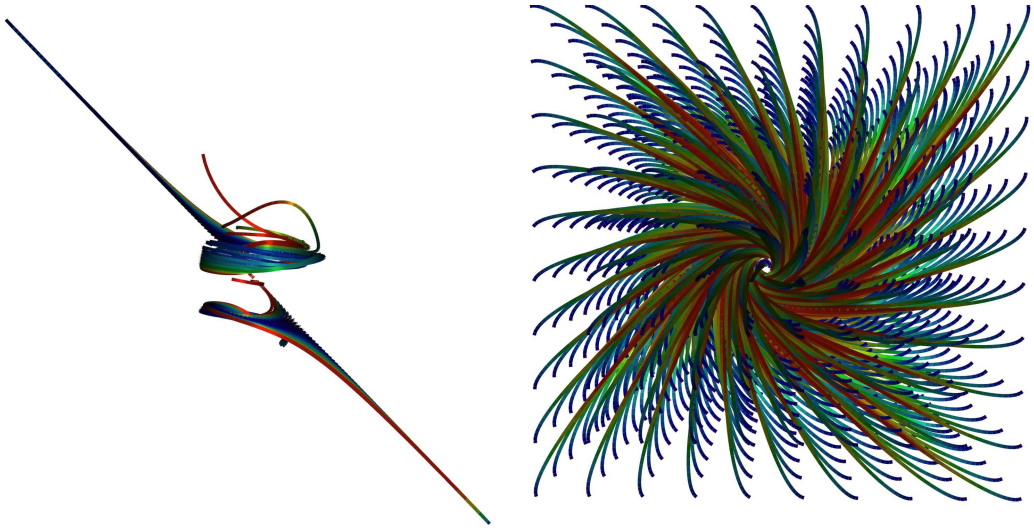


Figure 4.18: Timelike geodesics in the Kerr metric, particles are distributed along the diagonal coordinate axis (left) or uniformly in space (right). Particles are dragged around in the spacetime, but come to an apparent rest at the event horizon.

Distortions of extended freely falling objects Based upon a set of timelike geodesics we may also compute and trace the distortion of complete geometries. It were possible to use the vertices of some spatial geometry as the initial points of some geodesics, but that would require to re-compute the geodesic paths if the geometry is changed. Especially when computing geodesics in a numerical spacetime this is problematic because the evolution of the spacetime itself takes significant computation time. Instead, we distribute the initial points uniformly in coordinate space and evolve a geometry relative to these points. The initial velocity can be set independently, the easiest choice is to set it to zero. The cactus code [MPIfGP03, BMSW98] allows to evolve a (numerical) spacetime with particle point locations and proper time being computed via its geodesic equation. No interaction of the particles with each another and no back-reaction on the gravitational field is assumed. This assumption corresponds to an object made out of dust, where no forces influence the object. Equivalently, we may assume that the gravitational forces are much stronger than any electro-magnetic forces which try to keep the object stiff. The topology of the freely falling particles is kept constant during evolution. While the particle's location are evolved in coordinate space, the *computational coordinates*, we may also consider a *comoving chart* which maps each particle to a constant location in $\mathbb{R}^3$ with a yet to be defined scaling factor.

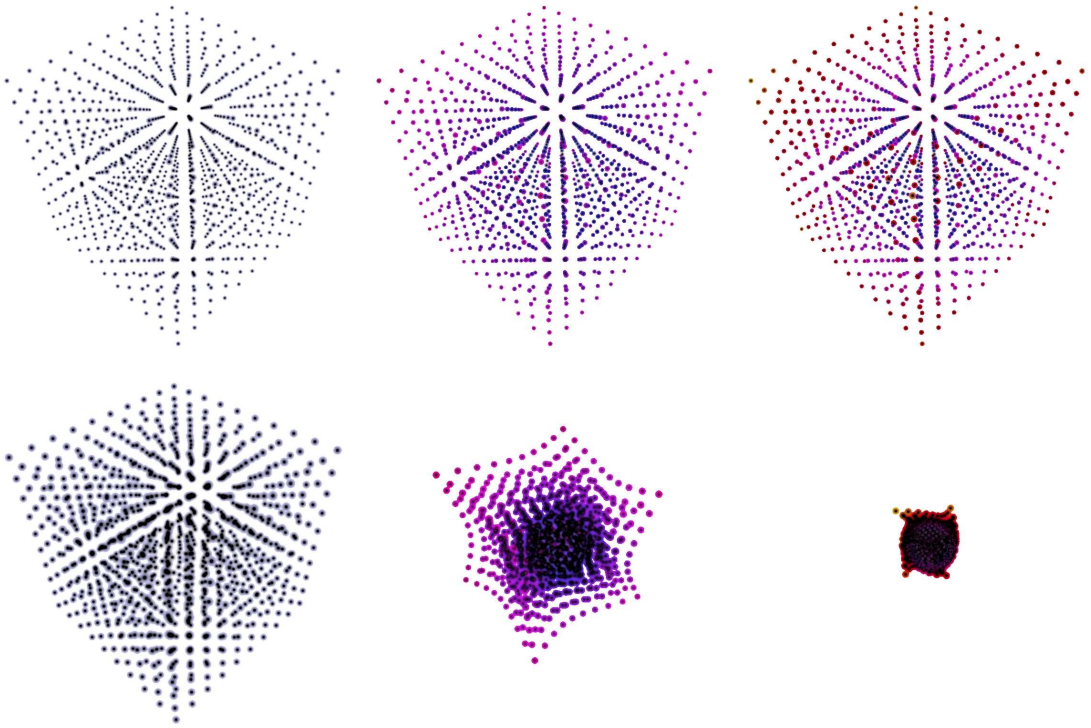


Figure 4.19: Initially uniformly distributed particles in the vicinity of a Kerr black hole, in comoving coordinates (upper row) and computational coordinates (lower row). Color indicates the proper time, the age of the particles.

The coordinates of an arbitrary extended object's vertex, which may be the output of a computer graphical modeling tool, may now be interpreted as if they were comoving coordinates. The object's size is scaled such that each vertex is enclosed within a uniform comoving cell (i.e. the objects needs to be scaled to fit into the bounding volume of particles in comoving coordinates). A certain vertex v may then be described via linear combination of its surrounding particle points p_i, i.e. written as an interpolation $v^\mu = \sum W_i p_i^\mu$ (which is a coordinate expression in the vector space $\mathbb{R}^3$). Now to get the shape of the object in the original coordinate system, we exchange the comoving coordinates p_i by the computational coordinates $\bar{p}_i$ while keeping the weights constant. We assume that the test particles are distributed sufficiently dense such that each regular cell in comoving coordinates can be interpreted as locally Minkowskian (def. 43) [otherwise the number of particles in the numerical simulation would have to be increase]. Then the coordinate of the vertex in computational coordinates $\bar{v}^\mu$ can be found via the simple vector space operation $\bar{v}^\mu = \sum W_i \bar{p}_i^\mu$.

Additionally, the comoving coordinate weights W_i (which, for a static object, are kept constant over time) also allow the evaluation of any field given in computational coordinates in comoving coordinates. An interesting example is the evolution of the proper time at each particle. For a certain time slice of

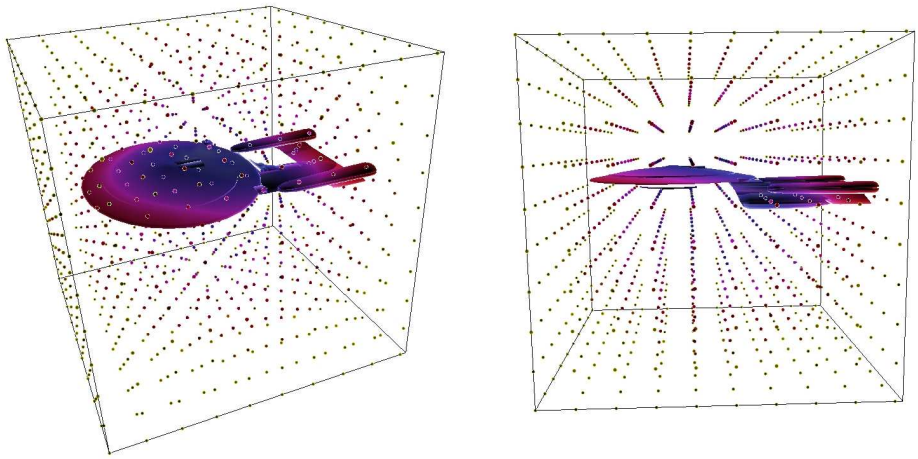


Figure 4.20: An extended object kept static in comoving coordinates. The colors indicate the proper time, the object thus is actually also stretched in time.

Figure 4.21: Geodesic distortion of an extended object. The movement of freely falling particles is projected by vertex interpolation on the object.

computational coordinates the proper time is not constant for all particles but varies significantly. Thus by the method described above we do not see the distorted object only distorted in space, but also in time.

The algorithm described here does not provide a true physical distortion as it would happen in nature. For that it were required to always render object vertices with the same proper time instead of slices of constant computational coordinate time. However, this is not implemented.

The change of the object's vertex locations in the computational chart can be formulated as a chart transition in the fiber bundle model. In contrast to analytic coordinate transition functions, the transformation rule is given here numerically via an interpolation, i.e. the operation of *grid evaluation*. The visualization procedure can be described as follows:

Task: Cactus [MPIfGP03] is able to compute the trajectories of freely falling particles (timelike geodesics) in an arbitrary spacetime. The initial conditions of these particles can be set as uniform regular distribution. While the regularity will be sustained during evolution (except when particles disappear or new ones appear), the uniformity will be destroyed rapidly. An arbitrary geometric object will evolve under the gravitational field as the particles behave (assuming there are no physical forces involved which constrain the geometry physically). The relationship among the uniformly distributed particles and the initial geometric appearance of the object is conserved for all times and can thus be used to compute the shape of the object during evolution.

Input: A uniform distribution of particles at $T = 0$, a regular distribution of particles for $T > 0$, the shape (vertices) of a geometrical object at $T = 0$.

Output: The shape of the object for $T > 0$.

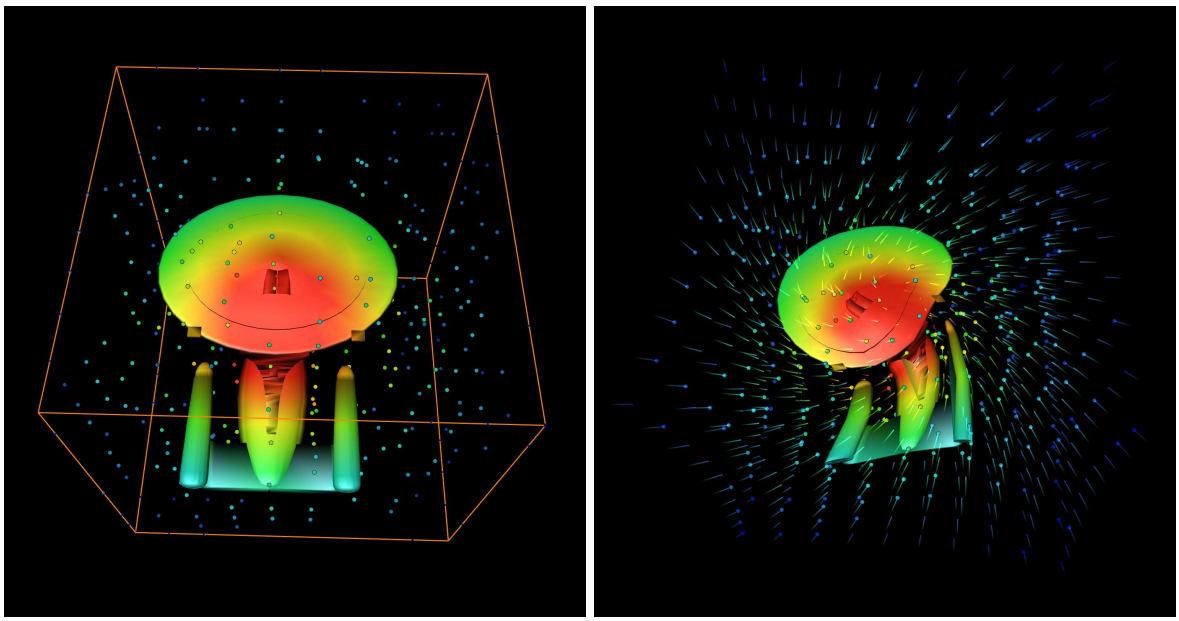


Figure 4.22: The spaceship enterprise in "free falling coordinates" (left) and "standard coordinates". In the first case, the reference vertices are uniformly distributed and the object appears undistorted. In the latter case, the reference points are spiraling inwards and the object is distorted accordingly.

Procedure:

1. Copy the grid of the geometric object with vertices given in the free falling (proper) chart to all time steps.
2. At $T = 0$, compute the weights of each object vertex via the uniform particle positions.
3. Propagate the vertex weights from $T = 0$ to all succeeding timesteps and let them refer to the current slice.
4. Compute the object vertices at a given time via the vertex weights and the current particle positions.

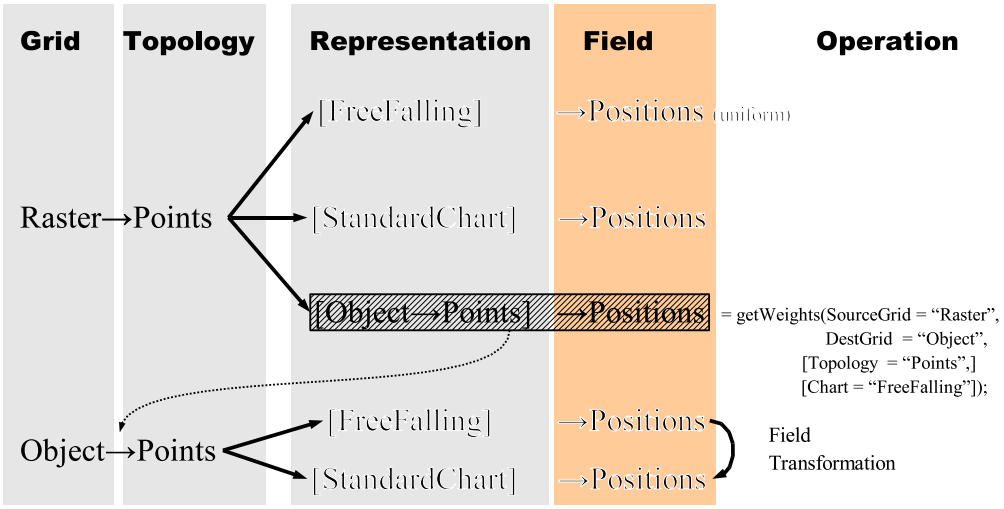


Figure 4.23: Fiber bundle objects and operations involved in computing the apparent shape of an object a freely falling coordinate system.

Lightlike Geodesics Bryson [Bry92] used bundles of lightlike geodesic to visualize spacetimes already back in 1992. He allowed to select certain initial data sets of lightlike geodesic bundles:

- **"cones":** Geodesics are originating at the same event p, with initial direction set up circular with the same angle α relative to a certain direction of interest $\vec{d}$:

  $$q_i = p \qquad \dot{q}_i = \cos\alpha\ \vec{d} + \sin\alpha(\cos\varphi_i\ \vec{X} + \sin\varphi_i\ \vec{Y})$$

  where q_i and $\dot{q}_i$ are the initial position and direction of the i^{th} geodesic out of N geodesics, with $\varphi_i = i2\pi/N$ and $\vec{X}$, $\vec{Y}$ two arbitrary 3-vectors orthonormal to $\vec{d}$.

- **"spray":** Geodesics are originating at the same event, with initial directions in the same plane but varying angle relative to a certain initial direction:

  $$q_i = p \qquad \dot{q}_i = \cos\varphi_i\ \vec{d} + \sin\varphi_i\ \vec{X}$$

  where $\varphi_i = (i - N/2)2\pi/N$.

- **"tubes":** Geodesics are pointing into the same direction, but originating at a circle around the same point:

  $$q_i = p + r(\cos\varphi_i\ \vec{X} + \sin\varphi_i\ \vec{Y}) \qquad \dot{q}_i = \vec{d}$$

  where $\varphi_i = i2\pi/N$.

- **"planar":** As an extension of the "tube" initial data, we may also shoot geodesics from the same plane, according to

  $$q_{ij} = p + i\vec{X} + j\vec{Y} \qquad \dot{q}_i = \vec{d}$$

  where i, j are two dimensional indices of a geodesic bundle.

Geodesic Deviation The difference among close geodesics as depicted by the cross-section of a geodesic bundle depends on differences of the Christoffel symbols and thus directly visualizes the Riemann tensor. We show a short proof based completely on coordinate free notation. Let $\Phi(s,t) : \mathbb{R}^2 \to M$ denote a two-dimensional array of geodesics such that for fixed parameter t the curves $\gamma(s) := \Phi(s, t = const.)$ are geodesics. Let $\delta := \frac{d}{dt}\Phi(s,t) \in \mathcal{T}(M)$ denote the deviation vector of points on the geodesics with same parameter $s \in \mathbb{R}$, also known as the *Jacobi field* of the geodesics [O'N83]. Here the dot denotes the derivative by the geodesic parameter s, which is given by the directional derivative along the geodesic:

$$\dot{\delta} := \frac{d}{ds}\delta \equiv \nabla_{\dot{\gamma}}\delta$$

We may also describe the deviation by an vector field $\delta \in \mathcal{T}(M)$ that is transported along the geodesic bundle, i.e. its evolution is described by the flow map along the geodesics. This requires its Lie derivative $\mathfrak{L}_{\dot{\gamma}}\delta$ (see p.20 for definition the flow map and its relation to the Lie derivative) along the geodesics to vanish:

$$0 = \mathfrak{L}_{\dot{\gamma}}\delta \equiv [\dot{\gamma}, \delta] = \dot{\gamma}\delta - \delta\dot{\gamma} = \nabla_{\dot{\gamma}}\delta - \nabla_{\delta}\dot{\gamma} \tag{4.13}$$

and we see that

$$\nabla_{\dot{\gamma}}\delta = -\nabla_{\delta}\dot{\gamma} \qquad .$$

If we compute the second derivate by the affine parameter we get

$$\ddot{\delta} := \frac{d^2}{ds^2}\delta \equiv \nabla_{\dot{\gamma}}\nabla_{\dot{\gamma}}\delta = -\nabla_{\dot{\gamma}}\nabla_{\delta}\dot{\gamma}$$

Recalling definition (2.42) of the Riemann tensor $K(u,v)w$ and inserting $u = \delta$, $w = v = \dot{\gamma}$ yields:

$$K(\delta, \dot{\gamma})\dot{\gamma} = \nabla_{\delta}\nabla_{\dot{\gamma}}\dot{\gamma} - \nabla_{\dot{\gamma}}\nabla_{\delta}\dot{\gamma} - \nabla_{[\delta,\dot{\gamma}]}\dot{\gamma} \qquad .$$

$\nabla_{\dot{\gamma}}\dot{\gamma} = 0$ is just the geodesic equation and from (4.13) we know that $[\delta, \dot{\gamma}] = 0$, thus we see that the second derivative of the deviation vector is linearly related to the Riemann curvature tensor:

$$\boxed{\ddot{\delta} = K(\delta, \dot{\gamma})\dot{\gamma}}$$

In flat space $K = 0$ in any coordinate system and no focusing happens. The deviation vector then describes just a linear expansion of a geodesic bundle like a cone, depending on its initial cross-section δ and opening angle $\dot{\delta}$. The evolution of the deviation vector in a chart is given by the coordinate expression

$$\ddot{\delta}^{\mu}\partial_{\mu} = K^{\mu}{}_{\alpha\beta\nu}\delta^{\alpha}\dot{\gamma}^{\beta}\dot{\gamma}^{\nu}\ \partial_{\mu} \quad .$$

The influence of curved space on a geodesic bundle is also known as Ricci focusing and plays a central role in gravitational lens theory, a extensive discussion of theory and application can be found in Ehlers et.al. [SSE94, SEF99].

4.3.2 Local Algorithms

Tensor Legend

The following sections contain a collection of various tensor field visualization methods using glyphs (icons) as the basic primitive. For evaluation and comparison of these methods, a "legend" is helpful that displays the extreme cases of tensor shape factors (see p.51 for definition of these). For such a legend we arrange the tensors within a triangle and construct the tensor shape from its barycentric coordinates according to the shape classification from 2.3.2:

$$\begin{array}{ccc} & c_s = 1 & \\ {\scriptstyle c_l=0}\nearrow\swarrow & & \searrow\nwarrow{\scriptstyle c_p=0} \\ c_p = 1 & \overset{c_s=0}{\longleftrightarrow} & c_l = 1 \end{array}$$

The appropriate tensor can be constructed as a diagonal tensor with eigenvalues computed from the shape factors. A possible way is to impose constant trace of the tensor along the edges of the tensor legend triangle, i.e. when writing a tensor as an eigenvalue triple $G = (c_s/3, c_s/3 + c_p/2, c_s/3 + c_p/2 + c_l)$ with the maximum eigenvector direction left open for visual adjustments:

$$\begin{array}{ccc} & \frac{1}{3}(1,1,1) & \\ {\scriptstyle c_l=0}\nearrow\swarrow & & \searrow\nwarrow{\scriptstyle c_p=0} \\ \frac{1}{2}(1,1,0) & \overset{c_s=0}{\longleftrightarrow} & (1,0,0) \end{array}$$

However, this choice would lead to unpleasant results, because the according quadric surfaces would degenerate at the $c_s = 0$ line: remembering (2.68), the maximum half axis of the tensor ellipsoid is the inverse square root of the minimum eigen value. The minimum eigenvalue is zero at the $c_s = 0$ line, thus the tensor ellipsoid becomes infinitely long in the direction of the minimum eigenvector (and at the same time becoming infinitely thin, therefore sustaining the trace).

A better choice is to normalize the tensors by sustaining the minimum eigenvalue. This was used for fig. 2.10 and provides a more visually intuitive clue of "a sphere flattening to a disc shrinking to a needle". Such a tensor shape is constructed via (2.73) from the eigenvalue triple as

$$(\lambda_{max}, \lambda_{med}, \lambda_{min}) = \left(\frac{3c_l}{c_s} + \frac{3c_p}{2c_s} + 1, \quad \frac{3c_p}{2c_s} + 1, \quad 1 \right) \lambda_{min}$$

whereby we may choose the minimum eigenvalue (the maximum diameter of the tensor ellipsoid) as a free parameter. For $\lambda_{min} = 1$ this tensor shape corresponds to a legend triangle of:

$$\begin{array}{ccc} & \frac{1}{3}(1,1,1) & \\ {\scriptstyle c_l=0}\nearrow\swarrow & & \searrow\nwarrow{\scriptstyle c_p=0} \\ \frac{1}{2}(\infty,\infty,1) & \overset{c_s=0}{\longleftrightarrow} & (\infty,1,1) \end{array}$$

We see that the minimum eigenvalue is kept constant by the cost of other eigenvalues becoming infinite; accordingly the trace of the tensor is not constant along the triangle edges. Instead, it is inversely proportional to the sphericity via $tr(G) = 3\lambda_{min}/c_s$, thus becoming infinite on the $c_s = 0$ edge as well. Nevertheless keeping λ_{min} =const. along the triangle edges is a better choice than keeping $tr(G)$ =const. . For numerical purposes, the $c_s = 0$ line should be avoided by a small $\epsilon > 0$, otherwise glyphs like a

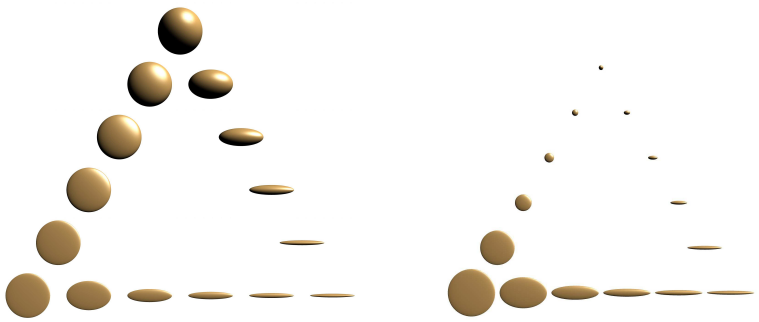


Figure 4.24: Legend with constant minimal eigenvalue (left) and constant trace (right).

quadric surface would become infinitely thin, ergo invisible and thus useless. Extending the tensor legend to include negative eigenvalues has not yet been investigated.

Metric Ellipsoids

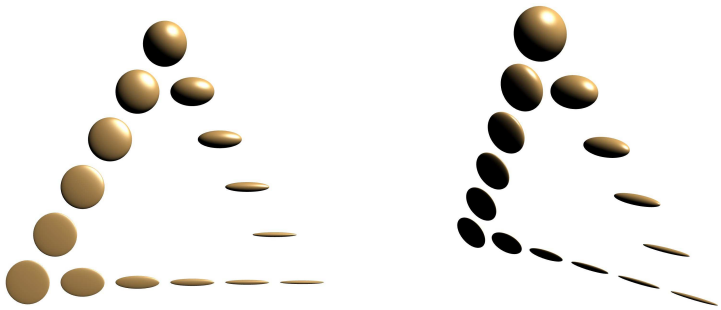


Figure 4.25: Tensor Legend for Metric Ellipsoids: At each point the quadric surface of the tensor at that point is drawn. To depict the appearance of the respective glyphs of a visualization method upon various tensor shapes in their full three dimensions, we display the tensor legend in front view and rotated from the side. Exactly the same setup will be used for all glyph-based visualization methods.

Rendering the quadric surface, p.49, of a tensor field at each point is the natural first choice when visualizing a tensor field. The quadric surface can be interpreted as the close[4] set of points $\varepsilon(p_0)$ that have a constant geodesic distance to a certain point $\varepsilon(p_0)$:

$$\varepsilon(p_0) = \{p : q(0) = p_0, q(1) = p, \int_0^1 g(\dot{q}, \dot{q}) \equiv \mathcal{C}\} \tag{4.14}$$

whereby q is a geodesic and the constant $\mathcal{C}$ is chosen according to the signature of the metric field. Equation (4.14) defines a *geodesic sphere*, which will be the basis of other visualization methods as well. For now, we restrict ourselves to a three-dimensional manifold. On p.155 we discuss a generalization to the four-dimensional case. In first order geodesics can be computed from a given initial direction $\vec{v} \in T_p(M)$ by a linear expression $q(s) = p + s\vec{v}$. Then, the geodesic sphere reduces to the (local) expression:

$$\varepsilon(p) := \{p : g(\vec{v}, \vec{v}) \equiv \mathcal{C}\} \quad ,$$

which is the equation of the quadric surface (2.59). We may interpret $\varepsilon(p)$ also as (the square of) the length (see 2.1.1) of tangential vectors around p.

The metric properties of the space can be read off by counting the ellipsoids to cross along a given path, see fig.4.27: an orbit around a black hole crosses quite few ellipses, while moving radially requires passing through many of them (because they are squeezed radially). Thus radial distances are largely stretched as compared to tangential distances, and the relation of circumference to radial distance is no longer 2π like in Euclidean geometry.

It may be useful to emphasize the anisotropic properties of the tensor field by subtracting the isotropic part, similar to the definition of the tensor deviator (2.62), via a user-specifiable tuning factor $s \in [0, 1]$:

$$\bar{g} = g - s \cdot \frac{1}{3}\mathrm{tr}(g) \cdot \eta \tag{4.15}$$

Eqn. (4.15) is a coordinate-free tensor equation as well, so it sustains the properties of the underlying tensor field. A caveat is that too large anisotropy enhancement parameters s will make $\bar{g}$ semi-definite, i.e. ellipsoids (with bounded volume) become hyperboloids (with infinite extent).

[4] "close enough" refers to the near domain where the metric tensor field can be taken constant – i.e. no derivatives of the metric tensor field are involved

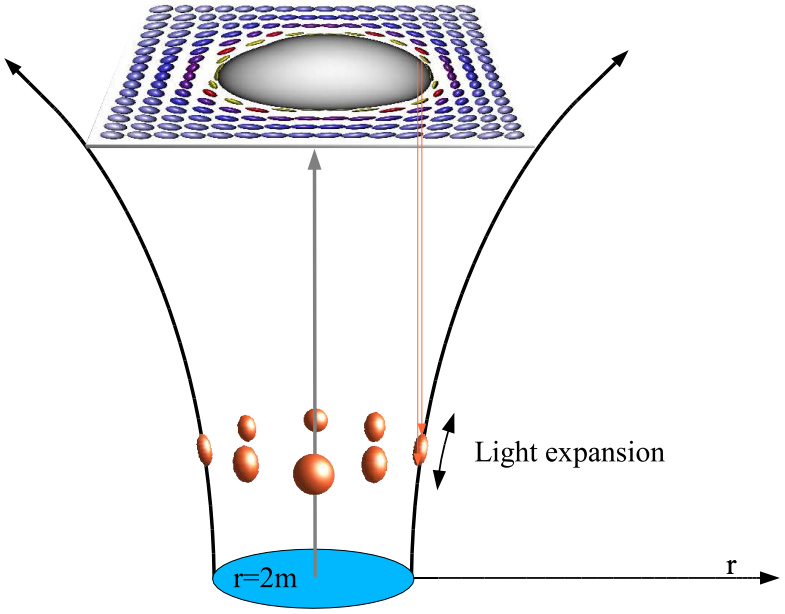


Figure 4.26: Metric Ellipsoids and Embedding surface: While light propagation takes place with equal speed in all directions *within* the embedding surface (see also fig. 2.11), metric ellipsoids display the projection of light spheres onto the coordinate plane.

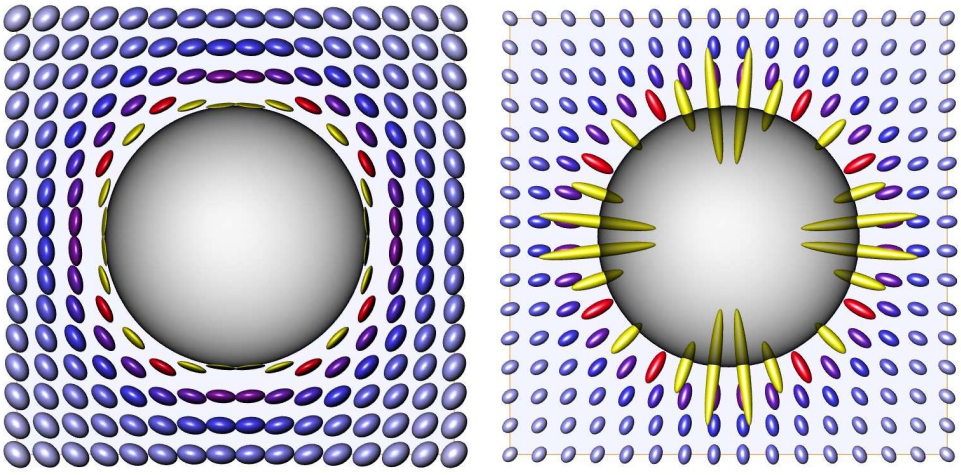


Figure 4.27: Metric ellipsoids for the Schwarzschild black hole: ellipsoids correspond to the appearance of a (local) sphere of expanding light in this coordinate system.

Metric Ellipsoids via Transformation Rules As we know from (2.66), a symmetric tensor can be written as a transformed unit matrix, $T^{-1}GT = \Lambda$, which is equivalent to $G = T\Lambda T^{-1}$. An implementation to render metric ellipsoids may thus use (2.61) to project each vertex of a unit sphere into its proper position on the metric ellipsoid, or, alternatively, compute the eigen decomposition and employ a transformation matrix on a given geometry according to (2.66). The first variant has the advantage that

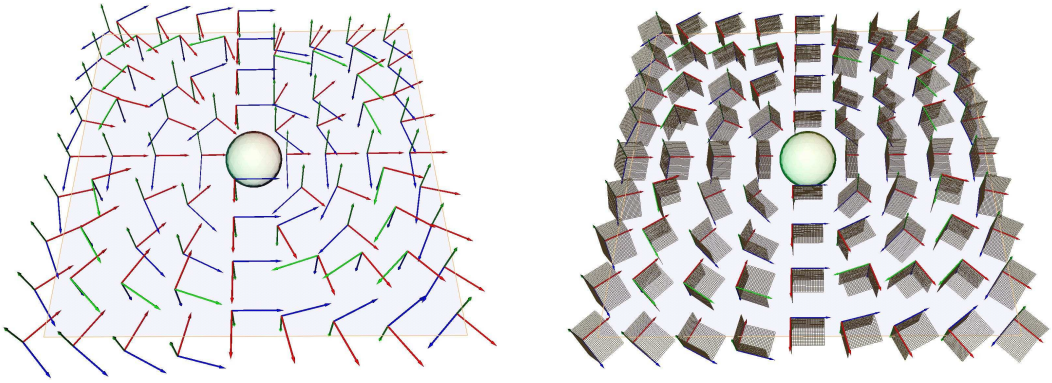


Figure 4.28: Distortion of an arbitrary geometry via a metric tensor field. Multiple transformed instances of the same freely chosen user-specifiable geometry provides flexibility to read off certain tensor field properties.

no eigenvectors/eigenvalues need to be computed, which might be an expensive operation. The second variant allows to employ hardware-accelerated geometric transformations upon arbitrary objects, yielding "metric-distorted" shapes. E.g., fig. 4.29 is an example of a complex geometry, rendered "under the influence" of the gravitational field of a Schwarzschild black hole. The spaceship object appears radially squeezed, while tangential distances remain untouched under approach. As seen from the spaceship, radial distances appear accordingly longer and become infinite just when touching the event horizon - a way to communicate to a non-scientific audience that radial spatial distances to the event horizon are infinite (mathematically: $g_{rr} \to \infty$). However it needs to be mentioned that this interpretation is valid for a

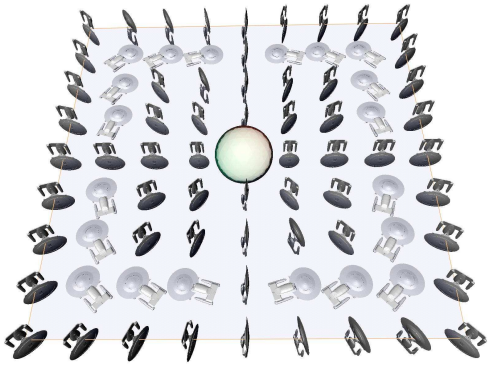


Figure 4.29: The spaceship Enterprise as it appears in the Schwarzschild coordinate system when approaching the event horizon of a black hole. It appears radially shortened, which visualizes that as seen from the space ship radial distances increase, whereas tangential distances remain unchanged.

static observer only, since that is what the Schwarzschild coordinates have been made for. For a moving observer, special relativistic Lorentz contraction would need to be taken into account. By that, large spatial distances for a spaceship at rest may shrink to a short distance for a moving one. By approaching the speed of light, even the spatially infinite distance becomes finite and a fast moving observer will see a finite distance and will cross the event horizon without further notification of it. The infinite spatial distance that static observers see towards the event horizon is thus nothing else than the fact that there can be no static observers *at* the event horizon, every object there is pulled inwards.

Tensor Glow

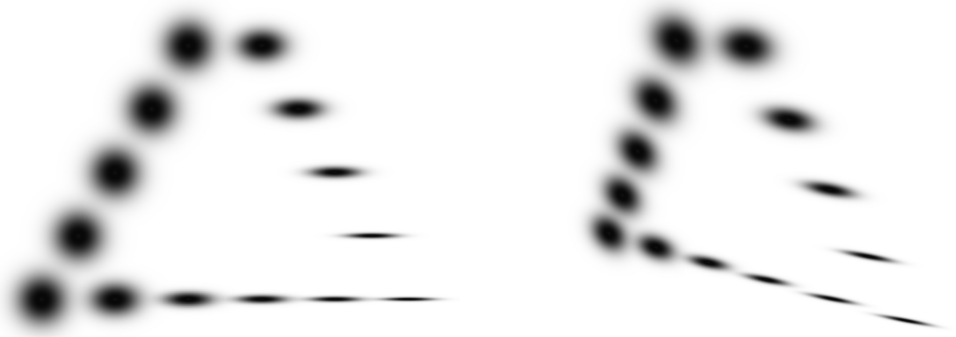


Figure 4.30: Tensor Legend for "Tensor Glow": drawing projected 2D textures instead of 3D geometries.

This technique is a slight variation of the tensor ellipsoids. The disadvantage of tensor ellipsoids are its opaque icons, leading to visual clutter when applied to a three-dimensional set of points. The idea of "Tensor Glow" is based on the obvious observation that when looking at some ellipsoid, one can only see the projected ellipsoid from a given viewpoint. Thus, it is sufficient to just draw an two-dimensional ellipse instead of a truly three-dimensional object that is projected by the 3D graphics engine. For drawing such an ellipsoid, we can draw a rectangle with an arbitrary texture on it. This rectangle needs to be oriented perpendicular to the view direction and transformed by a transformation matrix according to the projection of the tensor field in the view plane.

Let $\vec{z}$ be the view direction and $\vec{x}, \vec{y}$ be two orthonormal vectors describing the view plane. A point $\vec{\varepsilon}$ on the view plane can be computed from two parameters (a, b) via $\vec{\varepsilon} = a\vec{x} + b\vec{y}$. We get the projected ellipsoid by considering a ray $\vec{p} = \vec{\varepsilon} + \lambda\vec{z}$ that is orthogonal to the view plane (for orthogonal projection, we could formulate rays for perspective projection as well). Points on the ellipsoid obey $g(\vec{p}, \vec{p}) = 1$, which yields a quadratic equation in the ray parameter λ

$$1 = g(\vec{\varepsilon} + \lambda\vec{z}, \vec{\varepsilon} + \lambda\vec{z}) \equiv g(\vec{\varepsilon}, \vec{\varepsilon}) + 2\lambda g(\vec{\varepsilon}, \vec{z}) + \lambda^2 g(\vec{z}, \vec{z}) \tag{4.16}$$

For the projection of the ellipsoid on the view plane we are interested in the set of parameters (a, b) where the ray is tangential to the ellipsoid, i.e. where the discriminant of (4.16) vanishes:

$$\frac{g(\vec{\varepsilon}, \vec{z})^2}{g(\vec{z}, \vec{z})^2} - \frac{g(\vec{\varepsilon}, \vec{\varepsilon}) - 1}{g(\vec{z}, \vec{z})} = 0 \equiv g(\vec{\varepsilon}, \vec{z})^2 - g(\vec{\varepsilon}, \vec{\varepsilon})g(\vec{z}, \vec{z}) + g(\vec{z}, \vec{z}). \tag{4.17}$$

We model $\vec{\varepsilon} = a\vec{x} + b\vec{y}$:

$$\begin{aligned} g(\vec{\varepsilon}, \vec{\varepsilon}) &= a^2 g(\vec{x}, \vec{x}) + 2abg(\vec{x}, \vec{y}) + b^2 g(\vec{y}, \vec{y}) \\ g(\vec{\varepsilon}, \vec{z}) &= ag(\vec{x}, \vec{z}) + bg(\vec{y}, \vec{z}) \\ g(\vec{\varepsilon}, \vec{z})^2 &= a^2 g(\vec{x}, \vec{z})^2 + abg(\vec{x}, \vec{z})g(\vec{y}, \vec{z}) + bg(\vec{y}, \vec{z})^2 \end{aligned}$$

and search for a quadratic expression in (a, b) when inserting into (4.17):

$$\begin{aligned} a^2\big[g(\vec{x}, \vec{z})^2 - g(\vec{x}, \vec{x})g(\vec{z}, \vec{z})\big] + 2ab\big[g(\vec{x}, \vec{z})g(\vec{y}, \vec{z}) - g(\vec{x}, \vec{y})g(\vec{z}, \vec{z})\big] \\ + b^2\big[g(\vec{y}, \vec{z})^2 - g(\vec{y}, \vec{y})g(\vec{z}, \vec{z})\big] + g(\vec{z}, \vec{z}) = 0 \end{aligned} \tag{4.18}$$

The coefficients in (4.18) for a^2, $2ab$ and b^2 are the components of a bilinear form describing the *shadow* of the metric ellipsoid in the coordinates (a, b). Note that in this derivation we never used coordinates

on the 3-vectors, i.e. this derivation was completely coordinate-free. We may also write (4.18) as

$$\begin{pmatrix} a & b \end{pmatrix} \underbrace{\left[g(\vec{z},\vec{z}) \overbrace{\begin{pmatrix} g(\vec{x},\vec{x}) & g(\vec{x},\vec{y}) \\ & g(\vec{y},\vec{y}) \end{pmatrix}}^{=:\pi(g)} - \begin{pmatrix} g(\vec{x},\vec{z})^2 & g(\vec{x},\vec{z})g(\vec{y},\vec{z}) \\ & g(\vec{y},\vec{z})^2 \end{pmatrix} \right]}_{=:\sigma(g)} \begin{pmatrix} a \\ b \end{pmatrix} = g(\vec{z},\vec{z}) \tag{4.19}$$

whereby $\pi(g)$ is the projected ellipsoid $g(\vec{\varepsilon},\vec{\varepsilon})$ as in (2.74) (the intersection of the ellipsoid with the view plane) and $\sigma(g)$ is the "shadow ellipsoid". With (v, w) the eigenvectors of this 2×2 metric and (λ, μ) the corresponding eigenvalues, i.e.

$$\sigma g \cdot v = \lambda v \qquad \sigma g \cdot w = \mu w \qquad ,$$

the orientation of the resulting projected ellipsoid in 3D is given by evaluating the eigenvectors as linear combination of the basis $\{\vec{x},\vec{y}\}$:

$$\vec{p}_1 = v_x/\sqrt{\lambda}\ \vec{x} + v_x/\sqrt{\lambda}\ \vec{y} \tag{4.20}$$

$$\vec{p}_2 = w_x/\sqrt{\mu}\ \vec{x} + w_x/\sqrt{\mu}\ \vec{y} \tag{4.21}$$

The two three-dimensional vectors $\vec{p}_1, \vec{p}_2$ are orthonormal with respect to the metric tensor g (i.e. $g(\vec{p}_i, \vec{p}_j) = \delta_{ij}$) and are completely contained in the view plane $\vec{x}, \vec{y}$.

Since the eigenvalue equation of $\sigma(g)$ is just quadratic, it can be solved faster and more precise than the eigenvalue equation of the full 3×3 tensor matrix. From the visualization side, the advantage of this method is that we can use an arbitrary image as texture on the distorted rectangle. For instance, we may use a photograph of a nice ball to get a prettier appearance of tensor ellipsoids than by drawing ellipsoids from graphics primitives. Also, we may use a gaussian spot, which is fading out smoothly at the edges.

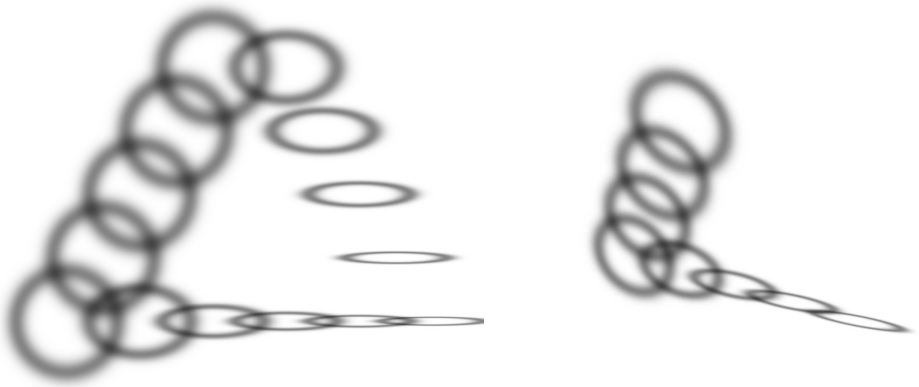


Figure 4.31: Using a two-dimensional texture to simulate the appearance of a three-dimensional distorted object. The texture can be animated, e.g. to yield the impression of an expanding light sphere.

This choice yields smooth images approaching volume rendering and is less vulnerable to visual clutter. The tensor field appears like "glowing stars" or "shining nebula", where its veils indicate the orientation of the tensor ellipsoids. Furthermore we may use animated textures, leading to visually dynamic effects like bursts light flashes emanating into space, which is the physically correct interpretation of a metric tensor's quadric surface.

The "Tensor Glow" idea is a predecessor of the more advanced "Tensor Splats" technique 4.3.2, but provides a more straightforward visual interpretation. Regarding rendering speed, the disadvantage of the Tensor Glow is that it is view-dependent and the resulting geometry cannot be compiled into OpenGL display lists. Depending on the hardware capabilities, this is compensated by the smaller numerical effort as compared to other methods which require preprocessing and might thus finally be faster.

Tensor Cones

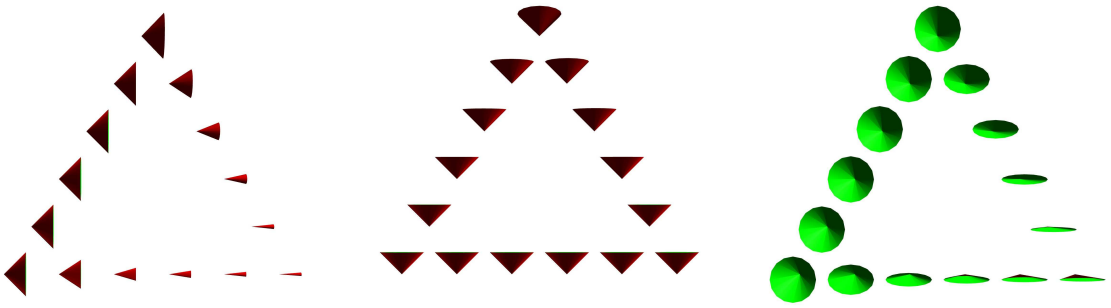


Figure 4.32: Tensor Legend for "Tensor Cones": Cones along a vector field with cross-section and orientation according to the action of the tensor field on the vector field.

The properties of a spacetime are often visualized in textbooks and lessons by light cones. They are "points of proper unit distance" from an event when omitting one spatial dimension from the spacetime. This idea inspires to a similar method for a 3-metric: By specifying some direction vector (or a vector field), a projection of the 3-metric to the plane orthogonal to this direction vector is determined. This 2-metric – the projected ellipsoid $\pi(g)$ from (4.19) – can be used to draw ellipses in the respective plane. Pulling these ellipses along the original vector with increasing scale yields a cone. Although only a subset

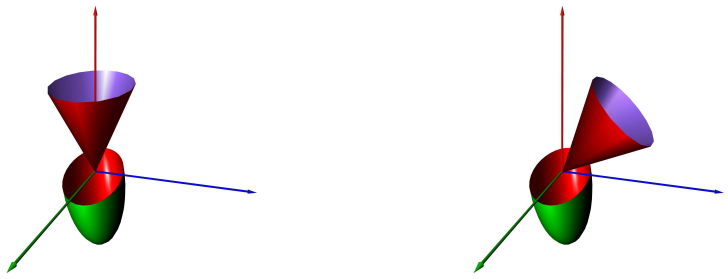


Figure 4.33: The cross-section of the cone equals the projected tensor ellipse (left). An additional rotation is used to display the residual components of the tensor (right).

of the metric components are shown, this method is still useful, because it allows to display a part of a metric field together with a vector field at once. The method may be extended if the cone is oriented along the action of the metric tensor on the directional vector, i.e. along $G(v, -)$. Then the orientation of the cone together with its cross-section displays the full metric tensor and is a complete replacement for the tensor ellipsoids. The information about the original direction vector field is lost hereby, but this vector field can still be added through vector glyphs, or via color encoding of the tensor cones.

This method is able to display the full metric components plus a vector field within one icon. Optionally transparency might be chosen according to an independent scalar field, thus revealing a glyph that is able to display 10 independent quantities at once, i.e. a full 4-metric:

$g(\vec{x},\vec{x}), g(\vec{x},\vec{y}), g(\vec{y},\vec{y})$	cone cross-section
$g(\vec{\beta}, -)$	cone axis direction
$g(\vec{\beta}, -)$ or $\vec{\beta}$	cone color
α	transparency

whereby α is an independent scalar field, e.g. the 4-metric's lapse function; $\vec{\beta}$ is an independent vector field, e.g. the 4-metric's shift vector field and $(\vec{x},\vec{y})$ are two 3-vectors orthogonal to $\vec{\beta}$ at each point.

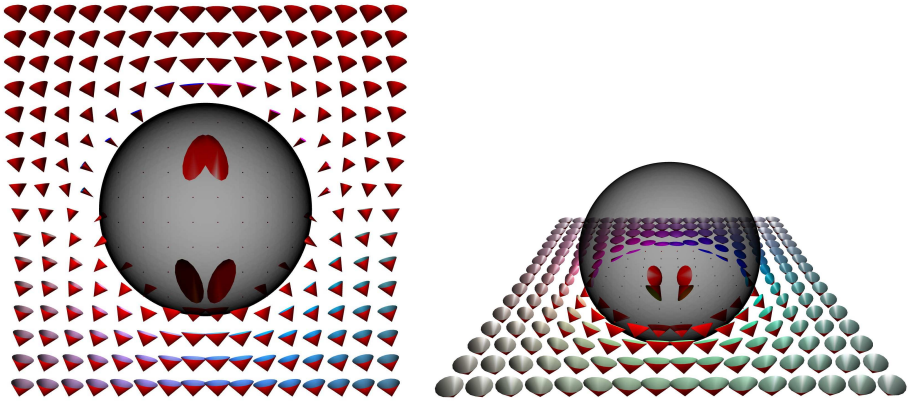


Figure 4.34: Rotated Tensor Cones in the Schwarzschild metric, axial and equatorial slice. Tensor Cones appear to become "attracted" to the black hole, similar to light cones.

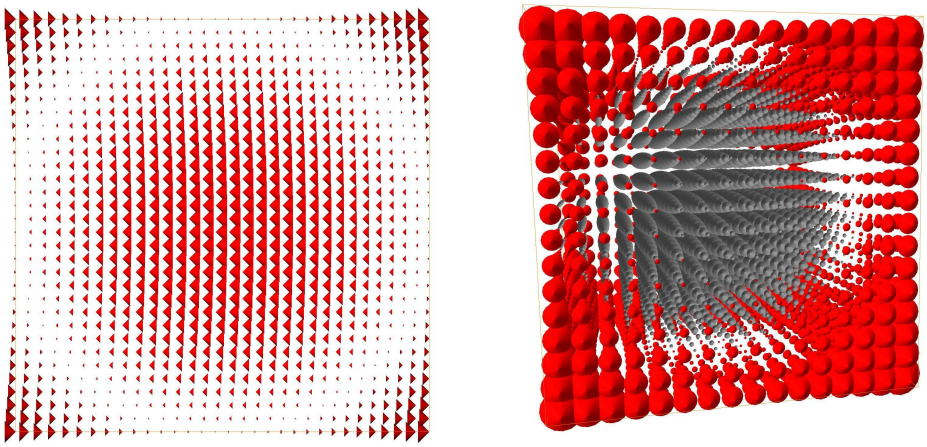


Figure 4.35: A 2D slice and 3D volume of the Warp metric rendered with tensor cones oriented along the shift vector field of the 4-metric.

Choosing the shift vector field allows us to visualize the Warp metric from p.47 in an interesting way, see fig. 4.35. Actually the spatial part of the warp metric is flat, Minkowskian, so there is nothing particular to see and usual 3D tensor field visualization methods must fail. However, by orienting and scaling the tensor cones, the visualization nicely shows the direction of the shift vector field and the locally Minkowskian behaviour of the spatially flat warp metric. This idea works well even in full 3D, where the spherical shape of the "warp bubble" comes out clear.

Reynolds Glyphs

Moore, Schorn and Moore [MSM95] mentioned glyphs are a variation of 2.61, where the length of the vector is set to

$$\vec{w} = \vec{v} \cdot \sqrt{g(\vec{v}, \vec{v})} \quad , \tag{4.22}$$

i.e. they are the "inverse" of an ellipsoid: a surface which is "inversed" at the unit sphere. Due to this inversion, differences from the unit sphere appear more prominent than using an ellipsoid or an ellipsoid of the inverse tensor, thus the Reynolds glyph is useful to enhance anisotropy more clearly than the quadric surface can do by itself.

Reynolds Glyphs have not been re-implemented in this work, because they are just a minor modification of metric ellipsoids and more advanced methods exist to enhance the anisotropic properties of a tensor field.

Haber Glyphs

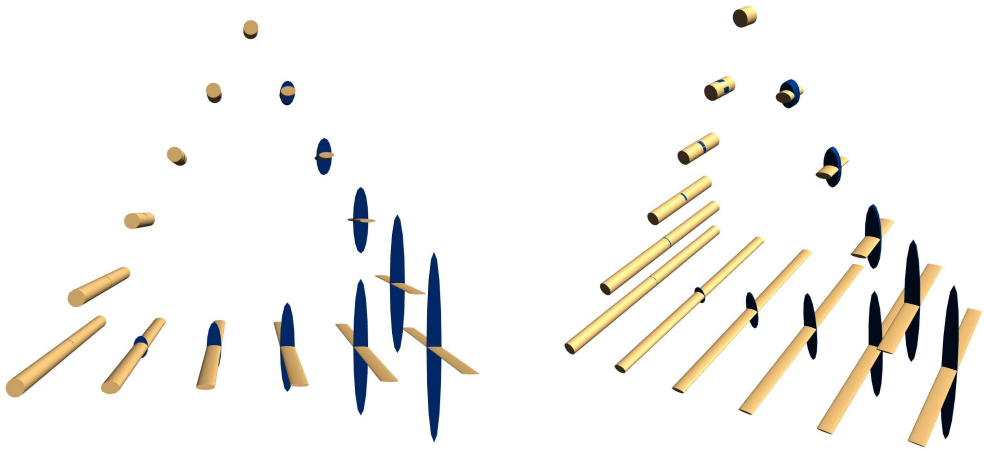


Figure 4.36: Tensor Legend for the Haber glyphs.

Haber [Hab90] invented a glyph object which is constructed by a disc and a cylinder to visualize the principal axis of a stress tensor field. It enhances the eigenvectors and eigenvalues of a tensor field better than tensor ellipsoids and clearly identifies anisotropic regions. In the implementation used here, the following mapping of eigenvalues to the glyph geometry is used:

cylinder length	$\sqrt{\lambda_{max}}$
disc thickness	$1/\sqrt{\lambda_{max}}$
cylinder's elliptical cross-section ...	$1/\sqrt{\lambda_{med}}, 1/\sqrt{\lambda_{min}}$
disc elliptical diameters	$\sqrt{\lambda_{med}}, \sqrt{\lambda_{min}}$

Both geometrical components, the cylinder and the disc, by itself also show the full parameter space of the tensor. As the cylinder is the inversion of the disc, deviations from the unit tensor come out very prominent. As a disadvantage of Haber glyphs, they are based on a full three-dimensional geometry and thus are slower to render than projection methods. This can be cured by reducing the geometry to just the essential elements, the outline of the cross-section, see fig. 4.38. This simpler rendering provides the same information as the fully geometric Haber glyph and also reduce the problem of visual clutter when applied to a tensor field in a three-dimensional space. However, such visual clutter remains a severe problem in this technique. Even worse, Haber glyphs are victims of strong isotropy artifacts, i.e. the orientation of the cylinders with spherical diameter becomes chaotic in isotropic regions.

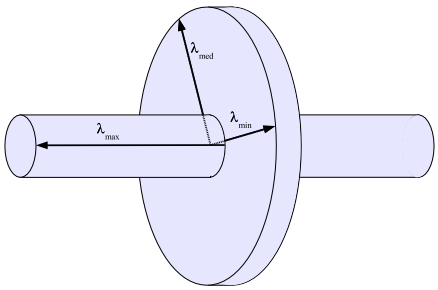


Figure 4.37: The Haber glyph: the a rod is drawn with the length of the largest eigenvalue and an elliptic disc whose axis are scaled according to the median and minor eigenvalue.

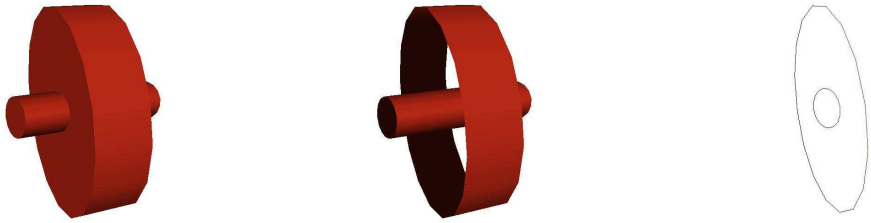


Figure 4.38: Variations of the Haber glyph: Closed, open, outlined.

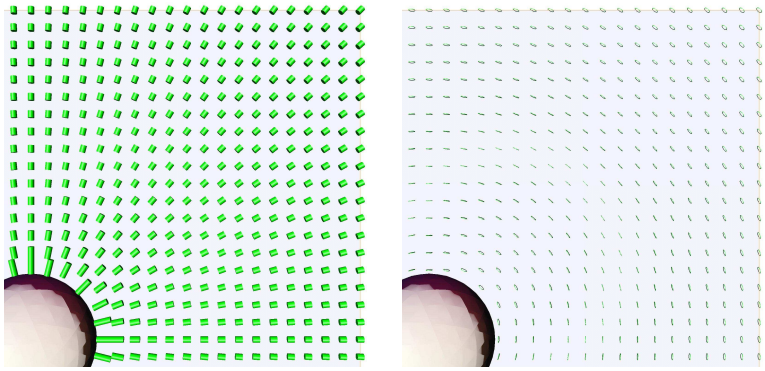


Figure 4.39: Haber Glyphs applied to the equatorial plane of the Schwarzschild metric, left in full geometry, right image as reduced outline. They depict the radial stretching of spacetime.

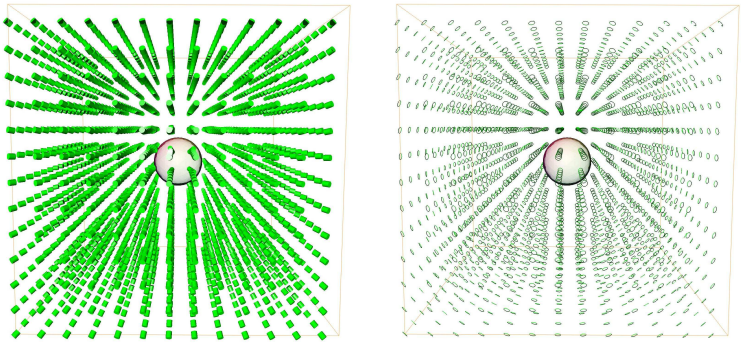


Figure 4.40: The usability of Haber Glyphs for three-dimensional tensor fields is quite limited, even when their are reduced to their outline.

Tensor Shadow

The "Tensor Shadow" technique is an experimental modification of the tensor rain technique from p.120 using covector laminae. The covector image as in fig. 4.15, left, displays a drawback of the "Tensor Rain" visualization idea: The laminae are invisible when seen edge-on, which happens far away from the black hole, but also in the equator plane. Thus the visual appearance does not fit to the intuitively expected behavior of spherical symmetry, but that is the central structure of the data. If our view direction is parallel to the probing vector field, we see the expected spherical symmetric behaviour as in fig. 4.15, right. Thus the idea is to always chose the view direction as the probing vector field. Consequently, the resulting visible structures become dependent of the view direction. So while a single image represents only three of six components of the tensor field, the interactive rotation of the view volume or the observer movement depicts different features at each view position.

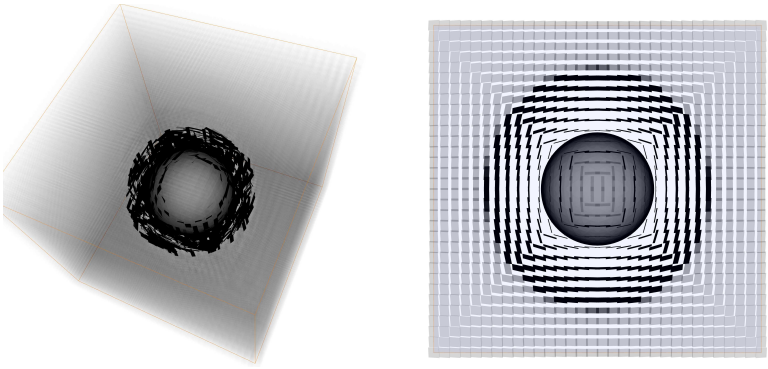


Figure 4.41: Tensor Shadow for the Kerr metric. We see the action of the tensor field on the view direction. In regions where the deviation is small, the laminae remain perpendicular to the view plane and transparent. Tangentially oriented laminae are emphasized, such that the radial "stretch" becomes visible, in a 3D volume as well as in a 2D slice. Note that the view right through to the center is not obscured by this technique, whereas techniques like the tensor splats hide the central region.

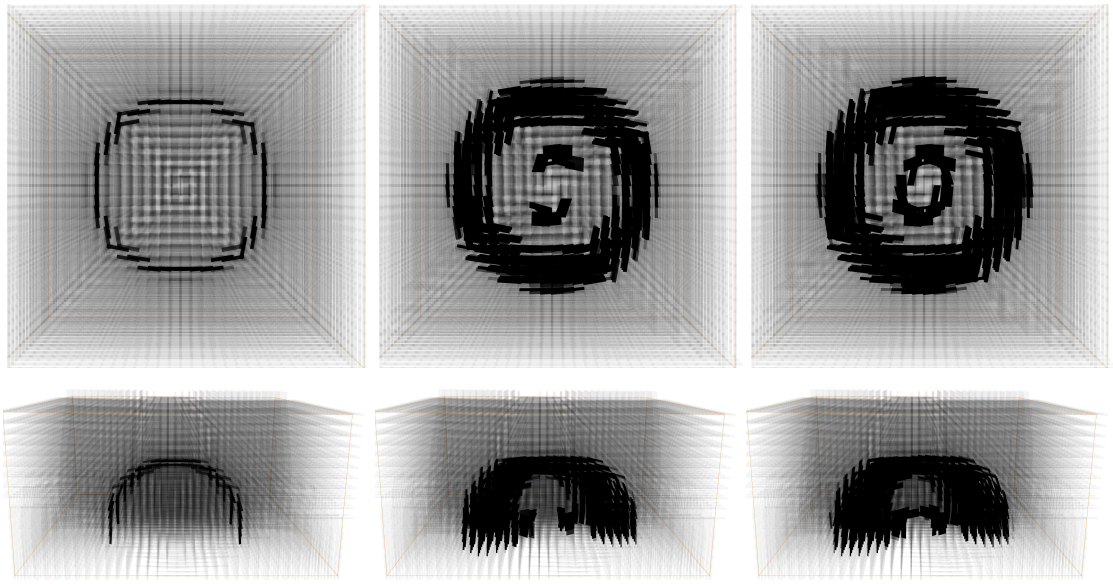


Figure 4.42: Tensor Shadows for a numerical metric of two colliding black holes, shown for different time steps at $T = 1, 26, 30$ (from left to right). The upper row displays the top view, lower row is the side view. Note the mutating geometry of the dark structure as it changes with the view direction. Regions of increased space stretching become visible.

Advantages of this visualization method is are its perfect suitability to time-dependent data and independence from positive definiteness and symmetry of the tensor field. Also, it is fast because no eigenvectors/eigenvalues computation are involved, but it is also slow because images are view-dependent and need to be recomputed with every change of view. Medium-sized datasets can still be used interactively with several frames per second, depending on available hardware.

Tensor Schlieren

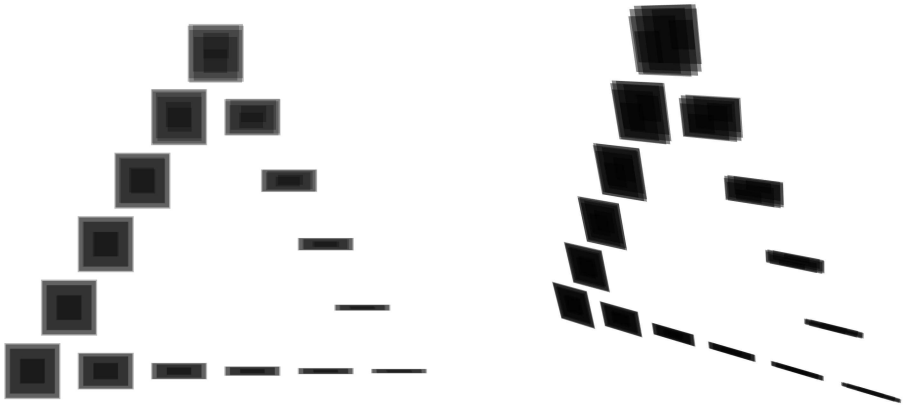


Figure 4.43: Tensor Legend for the Schlieren technique: The glyphs are Laminae oriented according to the dominant eigenvector and shaped by the median and minor eigenvalues.

Let us try to imagine "the optimal case" of tensor field visualization. One question arising first is what we want to achieve. For point-like inspection of a data set, some iconic technique visualizing a tensor object at a single point with all its properties there might be desirable. Maybe also including some small neighborhood around, like a plane or a small cube. However, for inspection of a huge three-dimensional data set and finding global structures and features such a technique is not appropriate, because pointwise icons require too much concentration on each point and they usually suffer under the view occlusion problem. For vector fields thus pointwise arrow icons are replaced by integral lines which may be more appropriate to display the global flow, so we may ask what is the "global flow" and the most intuitive interpretation of the action of a tensor field?

Thus we ask: what actually happens in curved space, what is the meaning of the metric tensor field as given in a coordinate volume? The answer is that it specifies the physical distance among neighboring points: even if they might reside regularly distributed in a coordinate space, the distance from one point to another one might be stretched very large, while others are compressed. So the physical grid of points is actually geometrically strongly distorted as compared with the coordinate grid. When thinking of a metric tensor field, it can be seen as the description of space stretching and contraction in each point of a given data volume. At each point the physical distance to its neighboring points is given by the metric field. In some sense, it gives the "density" of (physical) space as seen in a given data cube (coordinate space).

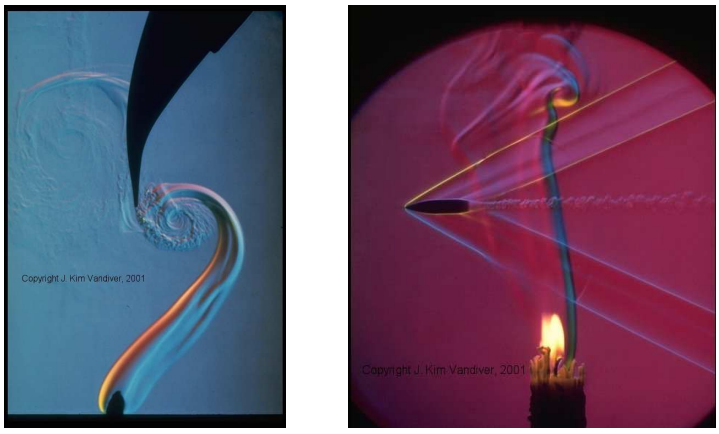


Figure 4.44: Schlieren photography: Small fluctuations in the the density are enhanced a special point-like intense light source. Images courtesy by J. Kim Vandiver [Set01].

Thus, when the "best" solution of computing an embedding of all points is not possible, were to at least display this "stretching" density. It is not a scalar field, because it is different in each direction. A nice inspiration how this could be done is given by the technique of Schlieren photography (fig. 4.44), which is used to display small fluctuations in the density of waves and turbulences within air: non-turbulent air is

kind of a "basic color", whereas wave fronts appear as brighter and darker areas. However in contrast to a true (scalar) density, the "tensor field density" one "depends on the direction". Schlieren photography is able to display the small convections of flowing air as sharp, small crisp details, thus encoding also directional information, i.e. the flow direction of the air in addition to the density fluctuations. So ideally a rendering technique for tensor fields could cope a similar impression: display density fluctuations and encode directional information at the same time.

The idea is to use the same appearance for a tensor field: "positive stretching" should be rendered lighter than the background, "negative stretching" (contraction) darker than the background. In contrast to a scalar field, we also need to encode directional information: in which direction does the stretching occur? That can be done via the orientation of the "local wavefront" of the artificial Schlieren image. It is given by the maximal eigenvector, the coloring is determinable by the maximal eigenvalue. The corresponding technique, hereby called "Tensor Schlieren" is to use small cell-sized Laminae which are oriented normal to the maximal eigenvector and transparency set according to the maximal eigenvector. The area size of the laminae is further determined by the minimal and median eigenvectors as well, such that this information is available as well.

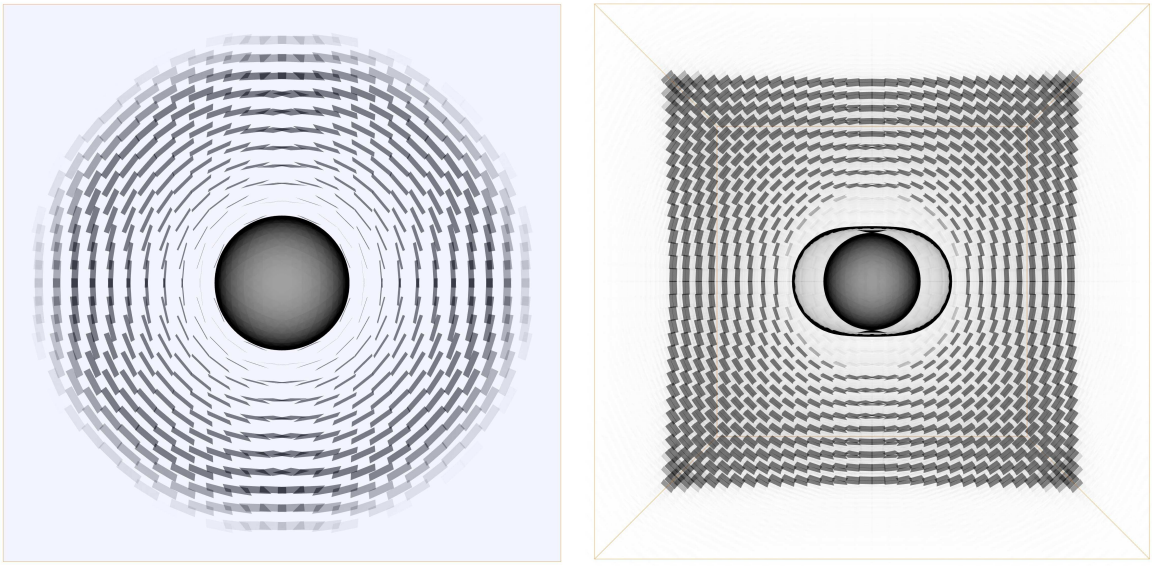
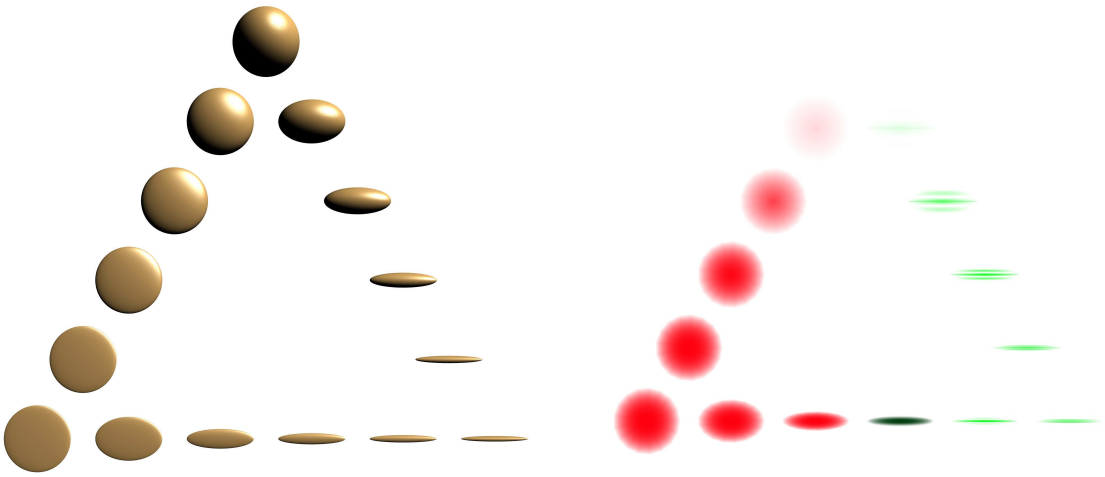


Figure 4.45: Tensor Schlieren for the Kerr Metric. Two-dimensional slice and three-dimensional data volume. In both cases, the laminae (see 4.2) indicating the maximum eigenvector are rendered transparent when they are parallel to the view plane.

Consequently, if we look at a tensor field from a direction orthogonal to its maximal eigenvector orientation, it will be rendered by the background color, i.e. become invisible, since the laminae are oriented orthogonal to the screen as well. The more the eigenvector field deviates from this direction, the more prominent it becomes. In the case of the Schwarzschild or Kerr metric, the dominant eigenvector field is $\vec{\partial}_r$, so all laminae will be oriented like portions of a spherical surface, being invisible tangentially but standing out like a spherical surface when looking directly toward the black hole, see fig. 4.45. The result are spherical shells with increasing visibility close to the event horizon.

Tensor Schlieren provide means to visualize an entire three-dimensional volume in a coordinate-independent, smooth way. However, the results are not convincing because the directional information of the laminae is lost within multiple layers of laminae. Also, the transition between neighboring laminae leads to confusing artifacts.

Tensor Splats



(a) Tensor Ellipsods (b) Tensor Splats

Figure 4.46: Tensor Legend: Ellipsoids incorporating the tensor field's value at each point in space is replaced by transparent textured splats. Isotropic regions are rendered transparently to discard any visual directional preference there.

Tensor Splats [BH03, BH04] is a newly invented technique that is derived from the Tensor Schlieren idea. The idea is to reflect the properties properties of planarity, sphericity and linearity (2.3.2) in a better way, since these quantities provide a visual clue to the tensor behavior in a more direct way than eigenvalues. In [WMK+99] a combination of a line segment, a disc and a sphere were used to draw an icon at each vertex point; each of these icon components is scaled by the respective quantity, yielding mostly spheres in regions high isotropy, disks in regions with dominant planarity and line segments like in mainly linear regions. Combinations of these icons show the relative size of these components. It gives correct visual impressions for isotropic regions, because the (undefined) maximum eigenvector does not contribute to the spherical icon component.

However, while this technique appropriately displays the tensor classification at each point and is thus good for pointwise analysis of the data, it suffers from view occlusion problems. The interesting part of a tensor field are actually the non-isotropic regions, where certain directions are preferred to others. To hide these isotropic regions we can just map the sphericity component to the transparency of the icons, thus clearing the view to the interesting areas. To further reduce the view occlusion problem, instead of drawing an opaque disc a polygon is drawn which is completely transparent at is edges (a "splat" [CM93a]), thus "fading out" and smoothly overlapping with the splats of the neighboring vertex icons. This splat is oriented by the maximal eigenvector, but fades out to become invisible in isotropic regions. Furthermore, it is not axially symmetric around its normal vector (which is the maximal eigenvector), but elliptically scaled according to the two orthogonal eigenvalues.

The idea is to employ flat, planar but transparent icons that are able to encode the same information content as an ellipsoid. These icons resemble splats which fade out smoothly. They may also overlap, similar as in the spot noise technique [CM93b], but sustaining the directional information provided by the tensor field. In contrast, splats utilized for volume rendering are always oriented parallel to the view plane and cannot communicate directional information. Keeping in mind that we are primarily interested in this information, it is reasonable to render isotropic regions completely transparent. As the eigenvectors have no meaning there, the only information available in isotropic regions is the trace of the tensor field, a single scalar, which can also be rendered by standard volume rendering, if of interest at all. Otherwise, in planar regions with two possible directions, a transparent disc with gaussian-like spherical transparency is an appropriate icon, while in linear regions a line segment is appropriate. To blend these two cases we may impose a one-dimensional periodic texture, e.g. some sinusoidal intensity, on the disc which indicates the direction of the larger eigenvector/eigenvalue pair on the dominant plane of the two largest eigenvector/eigenvalue pairs. The texture is scaled by the linearity factor c_l, thus stretched infinitely in planar regions with $c_l = 0$ and occurring exactly once per splat for $c_l = 1$. The resulting icons are shown in figure 4.46.

This technique is similar to the splatting technique which is employed for volume rendering of unstructured grids [Wes90]. However, for volume rendering purposes these splats are always oriented to be parallel to the screen (i.e. they are are view-dependent), whereas the Tensor Splats are oriented fixed in space (i.e. they are view-independent and can be put into an OpenGL display list to enhance rendering speed). This view-independent orientation, an artifact for volume rendering purposes, is a desired effect for this tensor field visualization, because it contains the directional information in the tensor field. It tells in which direction the tensor fields allows fastest/slowest traversal, yielding a nice "hurricane-like" cloud effect of overlapping layers as seen in fig. 4.47.

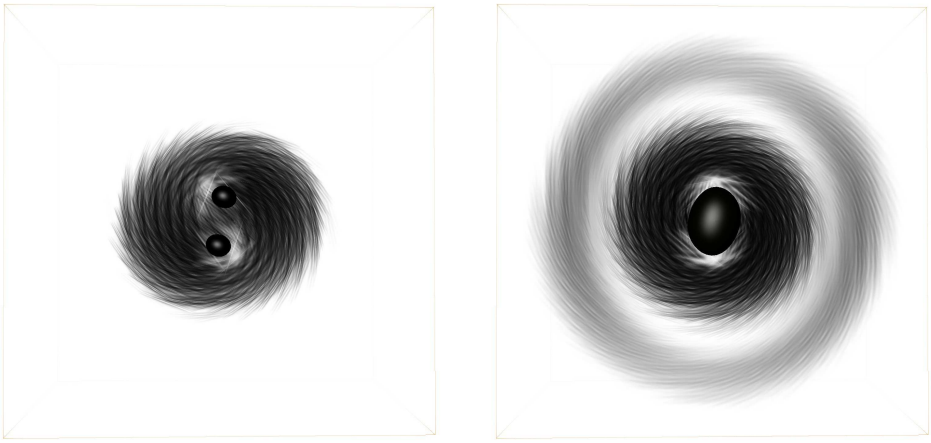


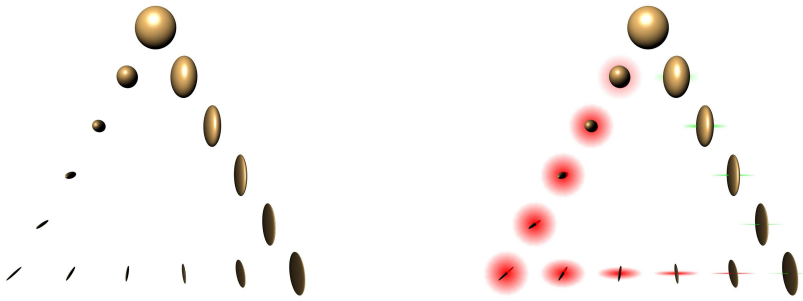
Figure 4.47: Tensor Splat applied to the metric tensor field of colliding black holes, displaying the whirling of spacetime around the gravitational throat.

The overall intuitive perceptibility of the resulting image is good even for large, complex tensor fields. It provides sufficient parameters to tune the visualization from raw data mining purposes (qualitative analysis) to detailed per-point property inspection (quantitative analysis). The technique is able to display multiple tensor quantities at each point in an a-priori independent way, much more than contained in the original field, so the visualization method is basically over-determined. However, this is not bad, because by using several ways to display basically the same quantity the spectator gets a visual impression of these relationships and can then fill in information in case it is visually hidden in some regions by switching to the visible components.

$\vec{v}_{\max}$	splat orientation	3
$\lambda_{\text{med}}/\lambda_{\text{med}}$	splat ellipticity	1
trace	splat size	1
sphericity	transparency	1
linearity	first color	1
planarity	second color	1
linearity/planarity threshold	icon change	1

Since the perceptiblity of the splat orientations is relevant for this visualization method, it is useful to enhance its appearance by using a non-spherical splat shape – which even improves the rendering speed by reducing the number of triangles to be drawn – and by adding a one-dimensional radial stripe texturing. This texturing may reduce the rendering speed somewhat (depending on the graphics hardware), but helps to dissolve the per-vertex appearance into a shade of smoothly overlapping stripes which indicate the action of the metric tensor field through some data volume. In this respect, the tensor splats method is similar to the two-dimensional method of van-gogh inspired tensor icons[LAK+98].

We also remember that by representing a tensor ellipsoid via its eigen decomposition, we also know the quadric surface of the inverse tensor $(G_{ij})^{-1}$ by inverting the eigenvalues $\lambda_i \to 1/\lambda_i$. It is thus easy to switch among a visualization of the tensor field and a visualization of the inverse tensor field; both might help to understand the properties of the tensor field, although the physical interpretation might be different. Examples are given in the application sections. The one-dimensional texturing on each splat can be freely scaled by a user-adjustable parameter to reveal the most pleasant overall appearance. In regions with non-zero linearity there is thus the freedom to specify how many stripes per splat, or visually: "needles per voxel", shall be used for a fixed c_l value. In any case, regions with higher linearity



(a) Ellipsoids of the inverse tensor.

(b) Inverse tensor ellipsoids with splats of the original tensor.

Figure 4.48: Glyphs of the inverse tensor. They are always orthogonal to the original tensor. whereby linear and planar shape are exchanged.

will be built from more and smaller needles, whereas regions with lower lineary will fade out to smoother appearance. Consequently, since the structural information is rendered with a higher resolution than the original data source, the resulting image should be inspected with higher pixel resolution as the data source.

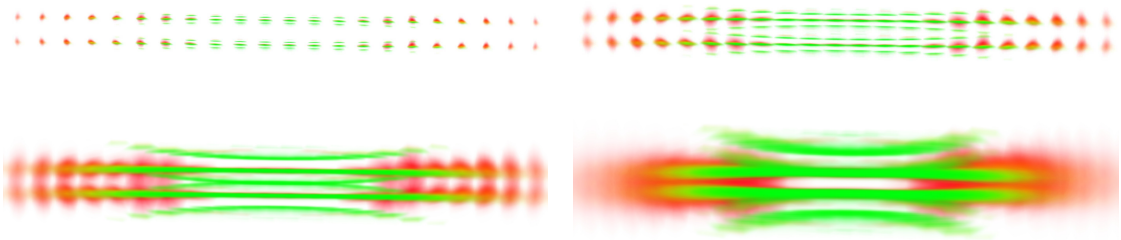


Figure 4.49: Transition from separate icons to continously smooth lines via scaling of the transparent splats: scale factor .2 (top left), .4 (top right), .8 (bottom left), 1.6 (bottom right). The data set is highly linear at the center but mostly planar at its borders.

If the splat diameter is an integer multiple of the cell size – which is constant only on a uniform grid –, secondary maxima of the periodic splat texture will coincidence, thus smoothly forming a line, as illustrated in fig. 4.49. The visual result looks like an integration line along the minimum eigenvector, but without no actual integration at all.

To enhance the transition from linear to planar regions, we use a colormap depending on

$$c_f := \frac{c_l}{c_l + c_p} \equiv \frac{\lambda_{max} - \lambda_{med}}{\lambda_{max} + \lambda_{med} - 2\lambda_{med}}$$

that specifies the location of a tensor shape on the bottom edge of the tensor legend triangle fig. 2.10. The colorization factor c_f becomes undefined in completely isotropic regions, where all eigenvalues are identical. However, this does not harm, because isotropic regions are rendered completely transparent (i.e. without issuing any OpenGL directives there, thereby also improving rendering speed), such that an eventual color value is irrelevant. Our primary encoding is to use complementary colors, e.g. mapping $c_p = 1$ to red and $c_l = 1$ to green. Other color codings are possible as well, and especially encoding the

Figure 4.50: Color Encoding of c_f, i.e. the $c_s = 0$ line in Fig. 2.10.

trace of the tensor field as luminosity together with color hue as linearity measurement is a reasonable option.

Benefits of the Rendering technique

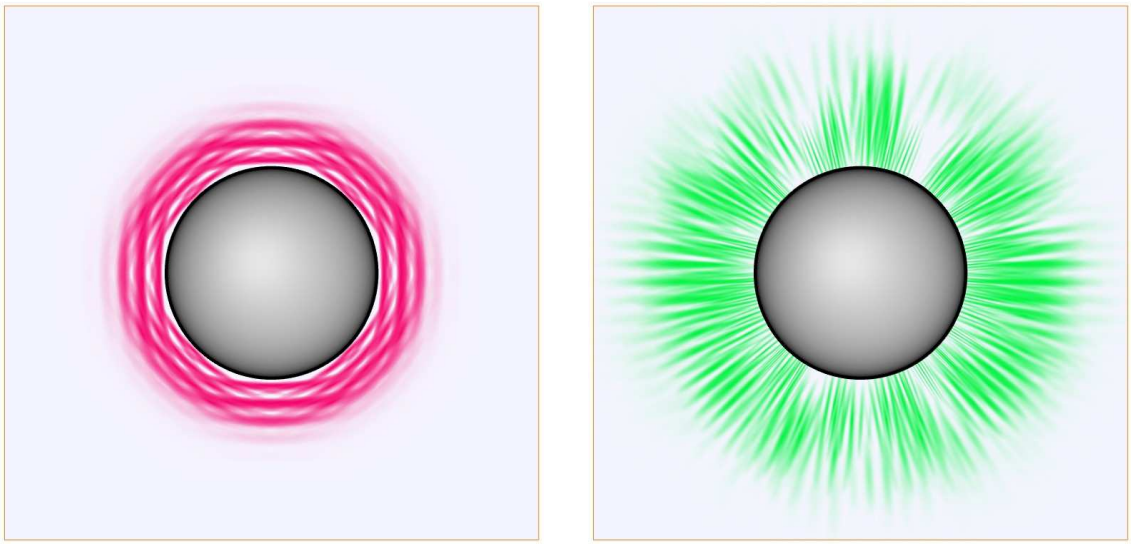
The tensor splat technique offers the following advantages:

- **no isotropy artifacts:** in contrast to many anisotropy enhancing visualization methods, nothing "weird" happens in isotropic regions, where the eigenvectors become undefined, i.e. no direction is suggested where none is preferred.
- **grid independence:** The rendering technique works point-wise; it is thus independent from the topological structure of the underlying data discretization scheme and is not limited to uniform cartesian grids, but can be applied directly to tetrahedral grids or adaptive mesh refinement (AMR) data.
- **applicable to dynamic data:** The technique is completely deterministic, i.e. it does not make use of random noise like in LIC [Sta98, IG97], and thus is perfectly suited for time-dependent data, revealing smooth animations.
- **fast:** The graphics primitives can be put into OpenGL display lists, thus achieving interactive rendering speeds and high-speed interactive animations of large data sets. Furthermore, some rendering parameters – like the number of stripes per splat element – can be set outside of the display list, thus fine-tuning of the visualization can be done interactively in real time.
- **adjustable:** by changing the diameter of the tensor splats, the number of texture stripes per tensor splat and transparency parameters the image can be adjusted from a global smoothed appearance that provides a raw qualitative overview of large structures into fine, point-wise icons that can be inspected point by point for data debugging and quantitative analysis of tensor values.

Application to the Schwarzschild Metric

For our visualization purposes we are only interested in the spatial part of the four-dimensional Schwarzschild metric (2.52). We see immediately that the radial eigenvector $\vec{\partial}_r$ is dominant with eigenvalue $g_{rr} = 1/(1-2m/r) > 1$, whereas the angular eigenvalues corresponding to $\vec{\partial}_\vartheta$ and $\vec{\partial}_\varphi$ are exactly like on a sphere, so no direction within concentric spherical shells is preferred. The radial eigenvalue $g_{rr} = 1/(1-2m/r)$ increases for $r \to 2m$ with $r \geq 2m$, the location of the event horizon. This indicates that spatial radial distances between static observers become longer close to the event horizon and even infinite at the event horizon itself (see also fig. 2.11).

Fig. 4.27 shows the tensor ellipsoids of the spatial part of the Schwarzschild metric (2.52), discretized on a uniform cartesian grid. The tensor ellipsoids of the metric can be interpreted as the appearance of a "flash of light" in the current coordinate system. In flat space, light would propagate into all directions equally, thus yielding a perfect sphere. When looking at a curved spacetime in some coordinate system, this is no longer the case and light appears to propagate slower in some direction due to the larger distances it has to travel. We see this behavior reflected in the tensor ellipsoids, Fig. 4.27 (left), flattening radially close to the event horizon, indicating that the physical distance between two radially separated points becomes very large there. Nevertheless light traveling tangentially is uninfluenced by the gravitational field. The metric ellipsoids can be understood as the size of a "unit sphere" or as "grid compression". The inverse tensor, Fig. 4.27 (right), is a direct measure for the grid stretching and displays how much a unit coordinate distance is stretched in physical space.



(a) Schwarzschild Splats (b) Inverse Splats

Figure 4.51: Tensor splats on a 2D slice revealing the radial grid stretching close to the event horizon.

We can imagine the tensor ellipsoids as circles imprinted on the surface of the isometric embedding surface in Fig. 2.11, but we view them as projection on the flat (coordinate) plane above. Clearly, the projected circles appear distorted there, exactly as in Fig. 4.27 (left). The tensor ellipsoids directly display this behavior, while sustaining the spherical symmetry within a three-dimensional volume.

The tensor field thus contains no linear but only planar regions, whereby the anisotropy is highest for $r \to 2m$ (the "event horizon" of the black hole). This behavior of radial stretching is easily depicted by the tensor splats, fig. 4.51. as radially oriented planar discs (fig.4.51(a)) indicating light propagation. The inverse metric (fig.4.51(b)) appears as radial "needles", indicating the pure radial stretching toward the event horizon as well. While both images, the tensor and the inverse tensor, provide the same information content, it is still useful to be able to switch between both visual representations.

The visualization method is not free from view occlusion problems, so in cases where one method or one region is hardly visible in one representation, the inverse representation might still provide useful insight. It is thus good to become familiar with both views, even though the direct physical interpretation is different.

Application to the Kerr Metric

The Kerr metric (2.53) is no longer spherically symmetric for angular momentum $a > 0$, but just axially symmetric around the rotation axis. We can see this property reflected in fig. 4.52 and expect linear

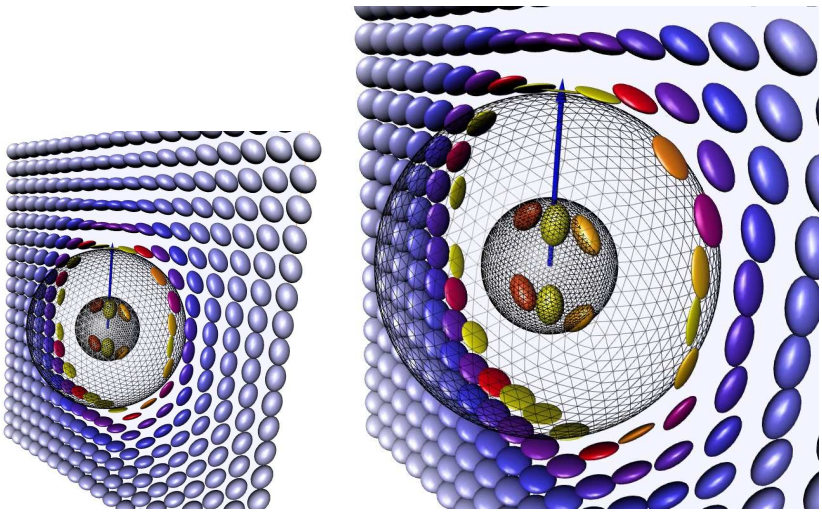


Figure 4.52: Kerr Ellipses. It is hard to see anything, but with good will we can spot that the ellipses flatten radially and are squeezed longitudinal while stretched axially at the equator.

regions for $a > 0$. The physical interpretation thus reads off that light may travel easier towards the pole than around the equator, i.e. when measuring a circumference then a (coordinate!) ring around the pole will be shorter than a ring residing within the equatorial plane. So physically a coordinate sphere close to the horizon is more like an ellipsoid that is flattened at the poles. However, it is not an exact ellipsoid, the correct geometry of the shape is an embedding[NH96, BAS02]. For large angular momenta, such an embedding no longer exists; the gaussian curvature becomes negative at the pole and there is no way to display an axially symmetric surface with negative curvature within $\mathbb{R}^3$.

Tensor Splats are able to exhibit the contrast among linear (in the equatorial plane) and planar regions (at the poles) quite prominently. We find radially oriented discs close to the poles; in the equatorial plane they morph to needles parallel to the rotation axis, see Fig. 4.53. We easily read off that light traversal towards the pole is easier and thus latitudal distances are shorter than longitudinal ones. In other words,

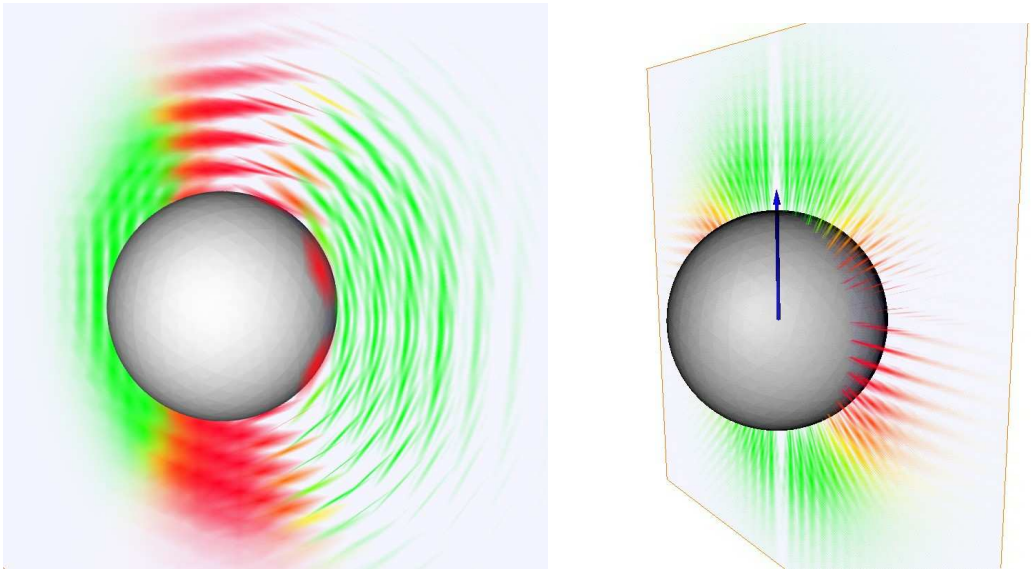


Figure 4.53: Tensor splats (left) and inverse tensor splats (right) of the anisotropic Kerr Metric, seen from the side, exposing linear and planar regions. Remarkably, the highest planarity values occur just outside of the event horizon (as depicted in more detail in fig. 4.74) and is not continously growing towards the horizon, as one might expect at first.

the axial stretching close to the equatorial plane indicates that a coordinate sphere is actually a flattened "bubbloid" with large equatorial circumference and short distance around the pole. The inverse splats

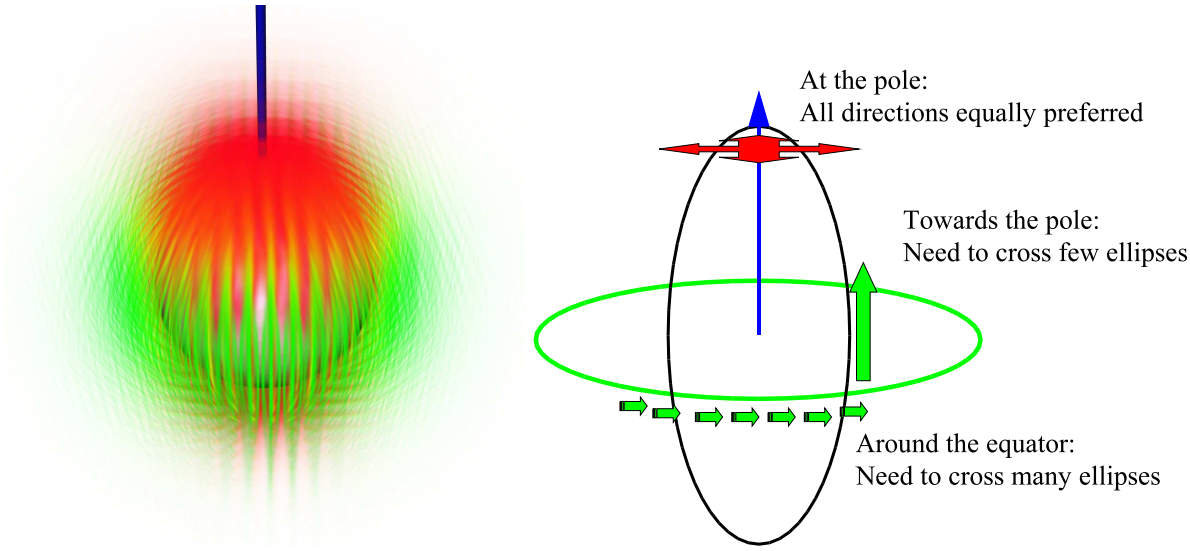


Figure 4.54: Reading off the geometrical properties of the Kerr spacetime via Tensor Splats: The more metric ellipsoids (visible as "tensor splat stripes") to cross in a certain direction, the larger is the physical proper distance along this direction.

depict the same behavior by spherical laminae in the equator plane, depicting that space is not only stretched radially, like at the poles and as we know from the Schwarzschild metric, but also tangentially in the equatorial plane.

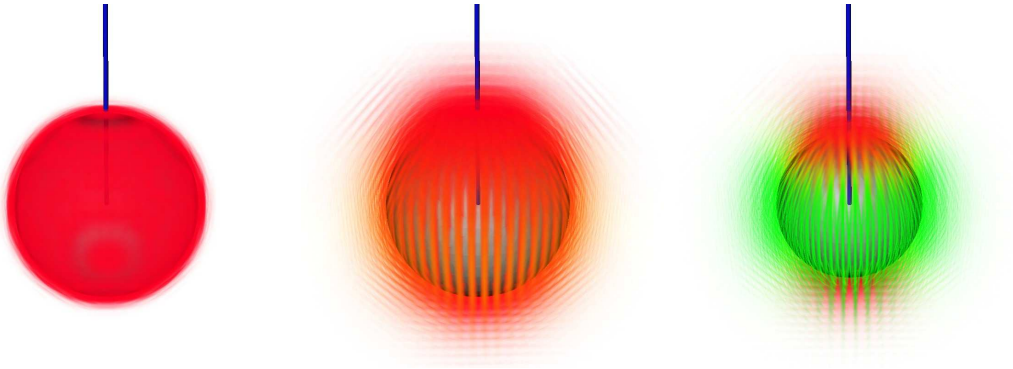


Figure 4.55: Kerr black holes spinning at 30%, 60% and 96%. The linearity around the equator increasing with the angular momentum of the black holes becomes easily visible.

Application to Numerical Data

Tensor Splats provide a direct visualization of the spatial metric γ of a numerical metric as described in 2.2.3. A current research topic is the numerical computation of the final orbit of two black holes before they collide. Of special interest in numerical relativity is the occurance of "grid stretching", the physical distance between neighboring points on the numerical grid, which is determined via the numerically computed metric γ. Due to physical or coordinate singularities they lead to numerical instabilities, ultimately killing the entire simulation sequence. Their early detection and propagation properties is thus essential for the development of improved evolution schemes.

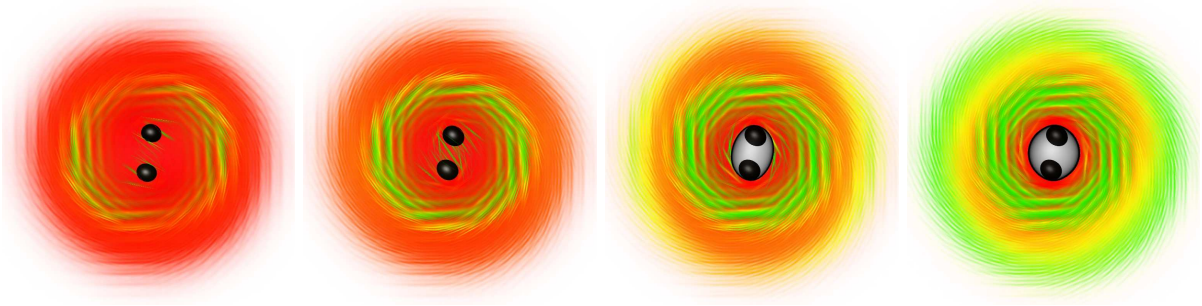
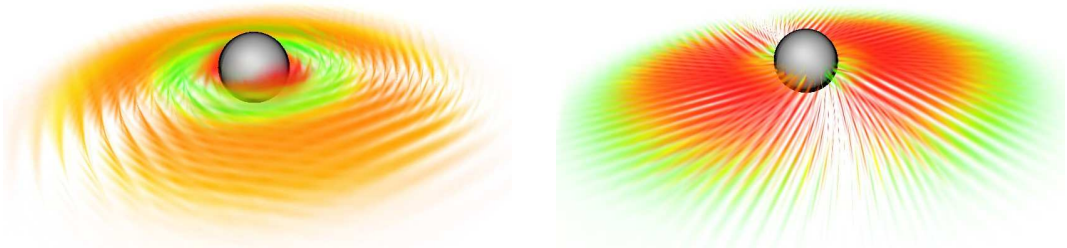


Figure 4.56: Snapshots from a numerical black hole merger sequence, revealing a region of highly linear grid stretching within a region of more planar grid stretching.

Fig. 4.56 demonstrates the tensor splats technique applied to a time sequence of such a numerical evolution from two colliding black holes, just around the moment where these black holes merge to form a common "apparent horizon". The $t \to \infty$ limit of such collision process is known to be the Kerr metric (2.53), although not necessarily in Boyer-Lindquist coordinates.

To determine the properties of the numerically resulting spacetime, we inspect the tensor field in the equatorial plane in more detail. Tensor splats in the equatorial plane, Fig.4.57(a), display light propagation, each splat element stands for e.g. one light second distance. The splats are mostly elongated around the equatorial direction, so light can travel very easily around the black hole. They are also somewhat elongated towards the pole, so light can travel tangentially to the pole with some hurdles. They are hardly elongated towards the center itself, so radial distances are very large. We see that the equatorial circumference is shorter than a a polar circumference. Alternatively we may inspect the inverse



(a) Metric (b) Inverse Metric

Figure 4.57: Equatorial plane. Two interpretations. Both tell that radial distances are very large, and circumventing the black hole via the pole is harder than orbiting it in the equatorial plane.

metric fig. 4.57(b), which directly displays distances in the spacetime. The splats are stretched radially very much, so radial distances are very large. They are also somewhat stretched in the axial direction, so going around the pole is a larger distance than circumventing at the equatorial plane. A quadrant of a dataset just after the merger of two black holes may be analysed using the Tensor Splats technique. Inspecting the region that is rendered in red in fig. 4.58, we see these red planar splats oriented radially, indicating mainly radial grid stretching, similar to the Schwarzschild metric. Additionally we see linear grid stretching in the equatorial plane, indicating that light can hardly propagate in the ∂_ϑ direction. The linearity is stronger closer to the black hole, visible as the color change to green together with smaller splat stripes. However, no such green region occurs close to the rotation axis, indicating that there is

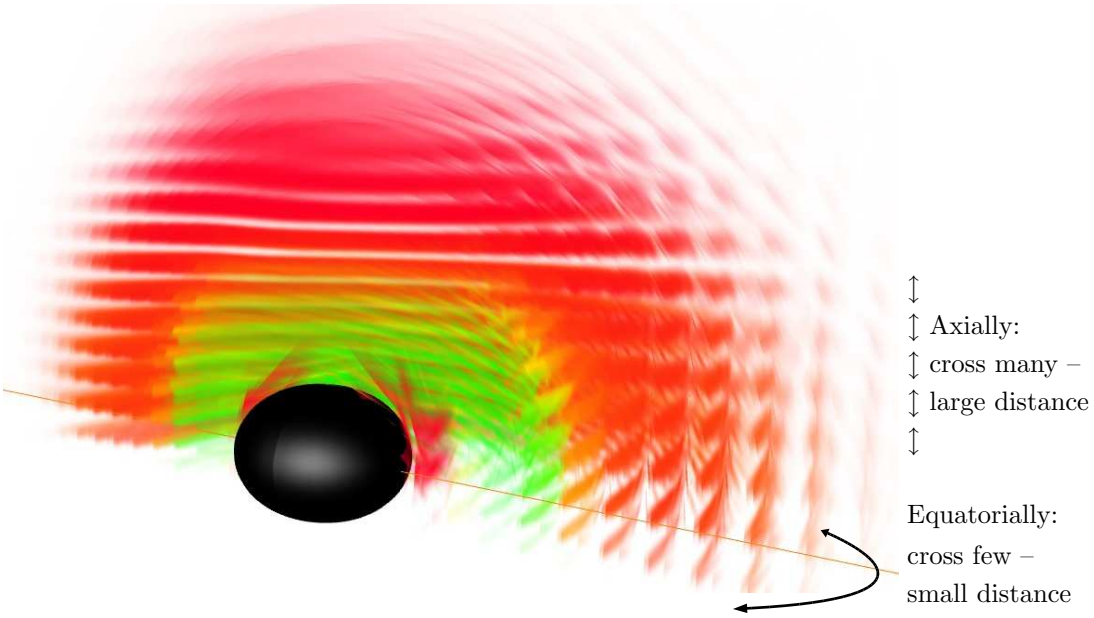


Figure 4.58: Quadrant with axial slice. At the rotation axis, the grid is stretched only radially, whereas in the equatorial plane grid stretching also occurs in the axial direction.

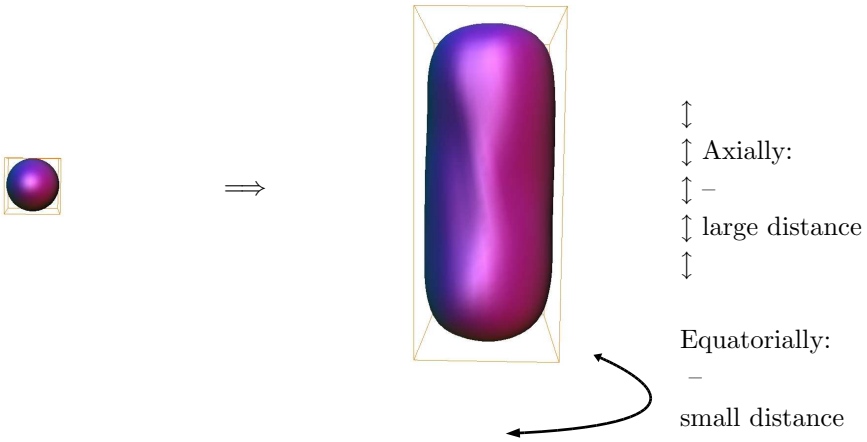


Figure 4.59: The true shape of a coordinate sphere in the numerical data, demonstrated by the apparent horizon from another numerical black hole simulation where an embedding has been computed (see A.1).

only radial planar grid stretching. At the pole itself, all tangential directions are equally stretched (no linearity), an expected symmetry around the rotation axis.

Our final interpretation tells us that equatorial circumferences are shorter than polar circumferences, i.e. the true shape of a coordinate sphere is like a cigar along the axis of rotation. Note that computing the actual shape would require determining the embedding[BAS02, Hot02], but such an embedding does not necessarily exist and could provide insight only for a surface, whereas the tensor splat technique allows to read off geometrical properties of an entire volume (provided its structure is not too complex).

Spacetime of three colliding Black Holes

The tensor splats technique has been applied to a numerical spacetime of three colliding black holes with no a-priori experience of its metric properties and no symmetries at all. The event horizon of the three black holes merging into one had been computed by a newly developed event horizon finder (Peter Diener, 2003). It was found that in a single slice there were up to four regions of high linearity in the metric tensor field which at first glimpse where not related to the event horizons. Isosurfaces of c_l depicted these regions as three connected loops enclosing the horizons (fig. 4.60, left column). Drawing tensor splats in the entire volume is not useful in this spacetime. because the volume is hardly isotropic anywhere such that transparently rendered regions are rare and the entire volume becomes overflooded with tensor splat icons. However, if we restrict the tensor splat rendering to regions of a certain linearity range (draw tensor splat only where $c_{l,min} \leq c_l \leq c_{l,max}$), we imitate an isosurface with a three-dimensional "skin", which now also indicates the metric tensor field on this skin. In fig. 4.60, regions of linear stretching, i.e.

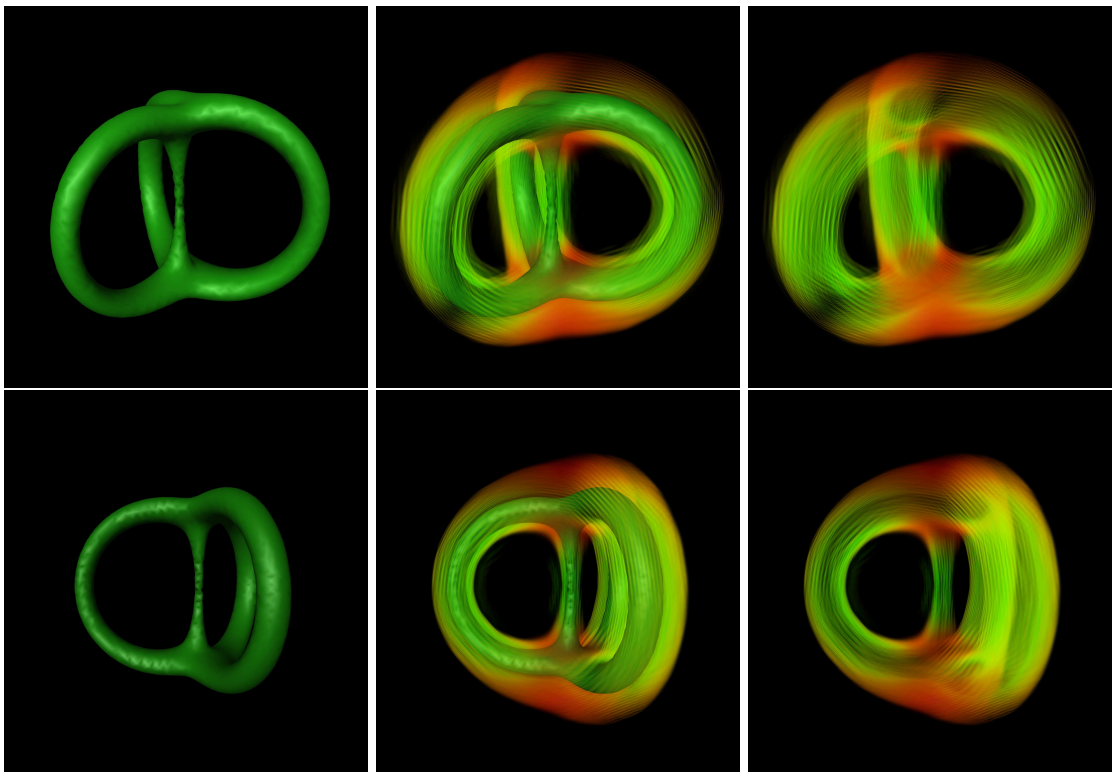


Figure 4.60: Left: isosurface around region with high linearity. Center: isosurface overlayed with tensor splats. Right: tensor splats without isosurface.

the region where light travels mostly in one direction, are depicted by as green stripes. Red areas depict areas of planar stretching, i.e. light may travel within a plane of two directions. When two regions of high linearity merge, both directions are equivalently possible, resulting in red, planar regions there.

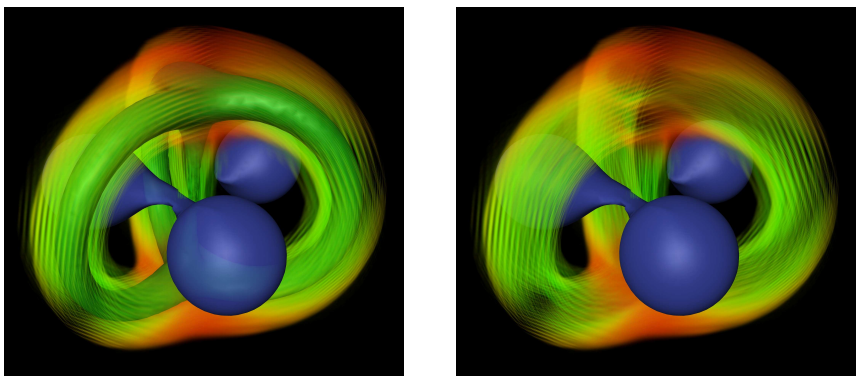


Figure 4.61: Event horizon (blue) of three colliding black holes and tensor splats.

Application to Extrinsic Curvature

Although the Tensor Splats technique was primarily designed with having positive definite tensor fields in mind, it can also be applied to other tensor fields, whereby negative definite areas are just ignored, i.e. rendered transparently.

The "extrinsic curvature" K, a by-product from numerical relativistic calculations, describes the embedding of the three-dimensional spatial slices within the four-dimensional spacetime. Its signature may vary at each point. fig. 4.62 demonstrates the results of tensor splats on selected time slices of a numerical simulation of colliding black holes. The extrinsic curvature tensor field may be interpreted as the rate of expansion of a volume element at the numerical grid.

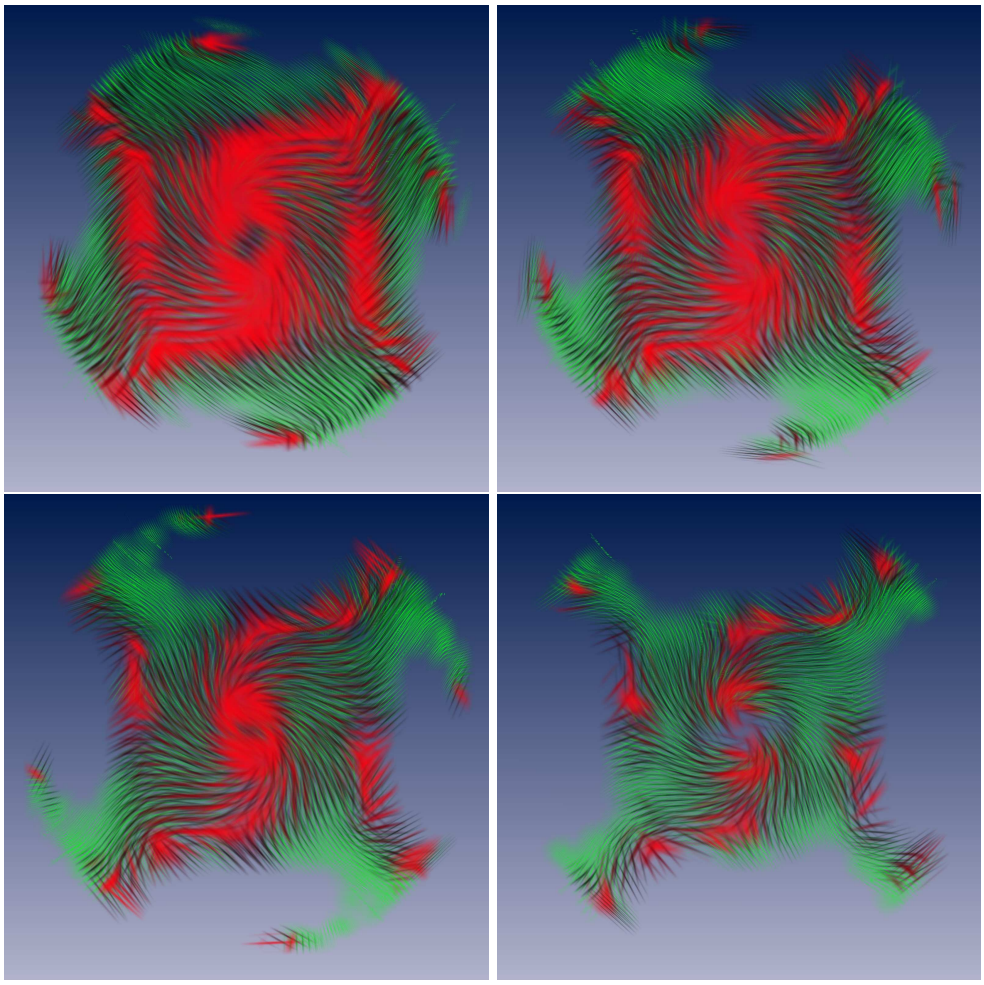


Figure 4.62: Tensor Splats Technique applied to the "extrinsic curvature" tensor field from a numerical simulation of two colliding black holes. Large portions are negative definite; these regions are not displayed; only positive definite regions are rendered with smoothly overlapping tensor splats. Color gives the rate of linear vs. planar expansion, while the texturing pattern reveals the expansion direction.

Geodesic Emission

At each event in a spacetime one can find a coordinate system such that the metric appears locally as a Minkowski metric, see def. 39. This is the coordinate system of a freely falling object. It does not experience any acceleration (def. 46), which is equivalent with the fact that the first derivatives of the metric tensor vanish in this coordinate system; consequently the Christoffel symbols, def. 2.34, are all zero as well. However, the second derivatives of the metric tensor cannot be transformed to zero, which means that even a freely falling observer may experience tidal forces due to a strong gravitational field. Their strength is proportional to the size of the observer. Tidal forces become arbitrarily small and vanish for objects of zero extend; so we may treat the coordinate frame of a freely falling observer as Minkowskian in first order. These coordinates are called Riemannian normal coordinates. An infinite number of such Riemannian normal coordinate frames may co-exist, because observers may be rotated relative to another one, and they may pass a certain point with different velocities. All these coordinate frames are related via (local) Lorentz transformations.

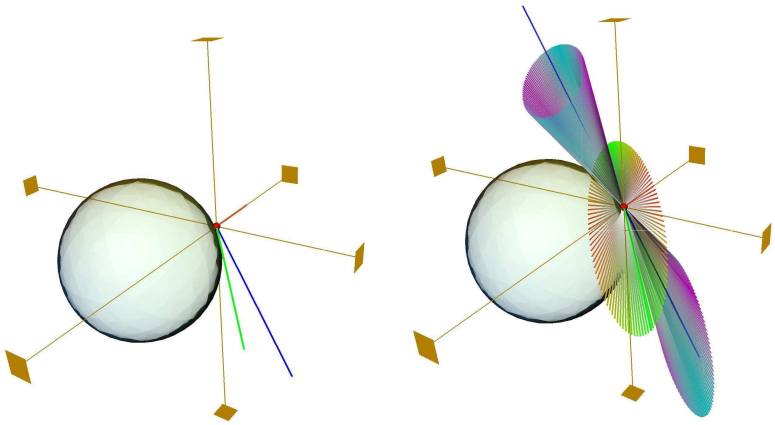


Figure 4.63: Visualizing the locally Minkowskian coordinates in the Schwarzschild metric: Orthogonal coordinate axes appear tilted (left), the opening angle of light cones - physically 45^o in their proper coordinate - appears changed (right).

Given a spacetime metric in some arbitrary coordinate system, we may compute a locally Minkowskian chart using the algorithm of Gram-Schmidt orthonormalization. For the purpose of a visualization tool we restrict ourself to the spatial 3-metric. At first, one needs to specify two linearly independent tangential vectors $\vec{v}_1$ and $\vec{v}_2$. They uniquely define three coordinate axis in the given coordinate system as the third vector is easily known as the cartesian cross-product $\vec{v}_3 = \vec{v}_2 \times \vec{v}_2$. We now want to compute a set of basis vectors $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$ which are orthonormal, i.e.

$$g(\vec{v}_i, \vec{v}_j\} = \delta_{i,j} \qquad . \tag{4.23}$$

In other words, the physical spacetime metric g becomes the Minkowski metric in the basis $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$. As precondition we require that $\vec{v}_1$ points into the same direction as $\vec{v}_1$ (same direction of view of a free-falling observer and observer at rest). Also, the orientation of the view should coincidence for a free-falling and resting observer, i.e. if both make an image along their line of sight, these images must not be rotated against each other.

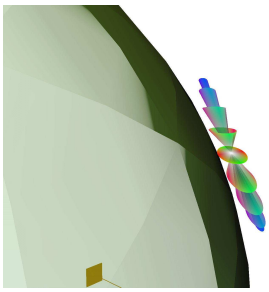


Figure 4.64: The Geodesic Emission glyphs are only a local indicator; they do not display the path of light rays, but just their initial conditions. They actually reside in the tangential space $T_p(M)$.

The algorithm of Gram-Schmidt orthonormalization works as an alternating operation of computing the projection of tangential vectors (using the physical angle according to def. 2.25) and normalization of the result (using the physical length according to def. 2.24) with the spacetime metric $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\} \longrightarrow \{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$:

$$\vec{v}_1 = \frac{\vec{v}_1}{\sqrt{|g(\vec{v}_1, \vec{v}_1)}} \tag{4.24}$$

$$\vec{v}_2 = \vec{v}_2 - g(\vec{v}_1, \vec{v}_2)\vec{v}_1$$

$$\vec{v}_2 = \frac{\vec{v}_2}{\sqrt{|g(\vec{v}_2, \vec{v}_2)}} \tag{4.25}$$

$$\vec{v}_3 = \vec{v}_3 - g(\vec{v}_1, \vec{v}_3)\vec{v}_1 - g(\vec{v}_2, \vec{v}_3)\vec{v}_2$$

$$\vec{v}_3 = \frac{\vec{v}_3}{\sqrt{|g(\vec{v}_3, \vec{v}_3)}} \tag{4.26}$$

The vectors $\vec{v}_2$ and $\vec{v}_3$ are intermediate variables with no physical significance (they are orthogonal, but not yet normalized). Once we have found a locally Minkowskian chart, others can be constructed via Lorentz transformation in any direction with any speed in the range that is allowed by special relativity.

A freely falling observer measures angles in Riemannian normal coordinates. So if we want to compute the view of an observer along the 2D surface of his view sphere as parameterized by altitude angle ϑ and azimuth angle φ, we need to interpret these angles as given in his locally Minkowskian coordinate system. A view direction vector $\vec{v}(\vartheta, \varphi)$ is then simply given as a polar coordinate expression by these basis vectors:

$$\vec{v}(\vartheta, \varphi) = \vec{v}_1 \sin\vartheta \cos\varphi + \vec{v}_2 \sin\vartheta \sin\varphi + \vec{v}_3 \cos\vartheta \quad .$$

By the means of the coordinate transformation (4.24), (4.25) and (4.26) the view direction can be expressed in the outer coordinate system and used as initial conditions when computing a geodesic path in the given spacetime coordinates. The surface built from the set of vectors of all (ϑ, φ) (i.e. the view sphere as seen in outer coordinates) is identical to a tensor ellipsoid like in 4.3.2; this is clear from (4.23).

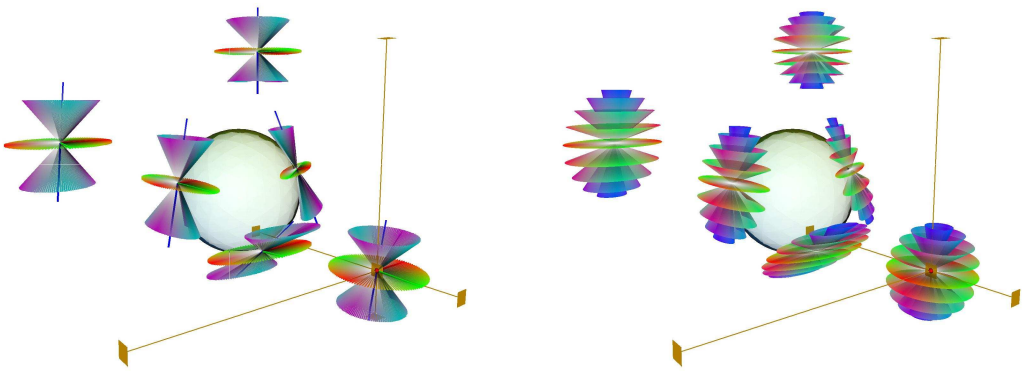


Figure 4.65: Many Geodesic Emission glyphs may be used together to a get a feeling for the properties of the underlying spacetime. Rendering single glyphs is fast enough to allow interactive exploration.

The view of an accelerated observer at each point is identical to the view of an freely falling observer (see e.g. N.Dragon's essay "Geometry of the Theory of General Relativity" [Dra01]) if their velocities coincidence at their meeting point. Thus the view as computed for a freely falling observer is also applicable to accelerated observers, which also includes observers at rest in the given spacetime coordinates, e.g. an observer hovering just above the event horizon of Schwarzschild static black hole.

Coordinate Transformation as Projection Matrix In a visualization environment using matrix transformations for image generation like OpenGL, the (local) appearance of the coordinate space to an observer residing at a certain point can be simulated by setting the transformation matrix (2.66) from Riemannian normal coordinates into coordinate space as the projection matrix used for rendering.

In the special case of axial symmetry like the Kerr or the Schwarzschild metric, where the eigenvectors are given by $\partial_r, \partial_\vartheta, \partial_\varphi$ with respective eigenvalues $E_r, E_\vartheta, E_\varphi$, we may compute the OpenGL projection

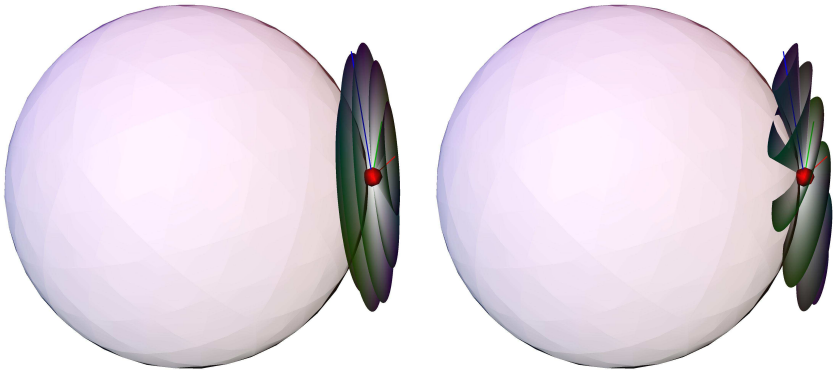


Figure 4.66: Appearance of the universe for an observer hovering close to the event horizon of a static black hole: We inspect the change of a view cone which has identical proper solid angle (i.e. same opening angle as experienced by an observer at a certain point). Its coordinate angle is smaller than looking tangentially, so the radially directed view cone can catch up more light. Consequently, most of the incident light from the universe will be concentrated in a very small view angle radially out- and inwards as seen from the observer. The actual view is demonstrated in fig. 4.69.

matrix (actually the spatial 3×3 matrix modifying an existing projection matrix) by first changing coordinates into a polar chart, applying scale factors according to the eigenvalues and transforming the result back into cartesian coordinates. This matrix is given by $A^{-1}SA$, whereby A^{-1} is the transformation matrix $\alpha^{\bar{\mu}}_{\mu}$ from p.16 and A is the transformation matrix $\alpha^{\mu}_{\bar{\mu}}$. The matrix S is a diagonal matrix built from the eigenvalues. The resulting camera projection transformation matrix is then

$$\begin{pmatrix} \frac{1}{r^2}\left(E_r x^2 + E_\vartheta y^2 + E_\varphi z^2\right) & \frac{z}{r\varrho}\left(E_r x^2 + E_\vartheta y^2 - E_\varphi \varrho^2\right) & \frac{xy}{r}\left(-E_r + E_\vartheta\right) \\ \frac{z}{r^3\varrho^2}\left(E_r x^2 + E_\vartheta y^2 - E_\varphi \varrho^2\right) & \frac{1}{r^2\varrho^2}\left(E_r x^2 z^2 + E_\vartheta y^2 z^2 - E_\varphi \varrho^4\right) & \frac{xyz}{r^2\varrho}\left(-E_r + E_\vartheta\right) \\ \frac{xy}{r\varrho^2}\left(-E_r + E_\vartheta\right) & \frac{xyz}{r\varrho^3}\left(-E_r + E_\vartheta\right) & \frac{1}{\varrho^2}\left(E_r y^2 + E_\vartheta x^2\right) \end{pmatrix} \tag{4.27}$$

whereby $\varrho := \sqrt{x^2 + y^2}$ and $r := \sqrt{x^2 + y^2 + z^2}$. Note that this matrix is the unit matrix for $E_r = E_\vartheta = E_\varphi = 1$. For a metric that is diagonal in polar coordinates like the Schwarzschild metric, the eigenvalues are simply its diagonal components, i.e. $E_r = g_{rr}$, $E_\vartheta = g_{\vartheta\vartheta}$ and $E_\varphi = g_{\varphi\varphi}$.

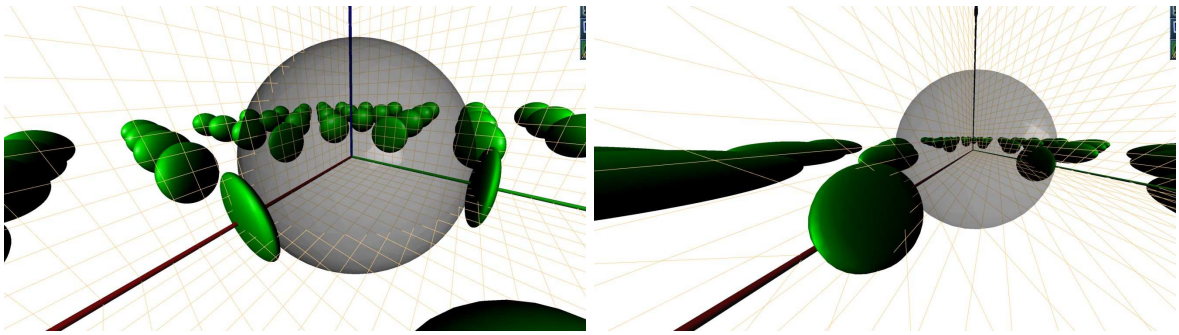


Figure 4.67: The camera projection matrix is such that the metric ellipsoids (same as on p.128) are "unstretched" to become spheres again. Left image shows metric ellipsoids in a closeup view, while the right image demonstrates the effect of the projection matrix switched on. Note that the apparent size of the BH virtually shrinks – this effect is discussed more detailed in fig. 4.69.

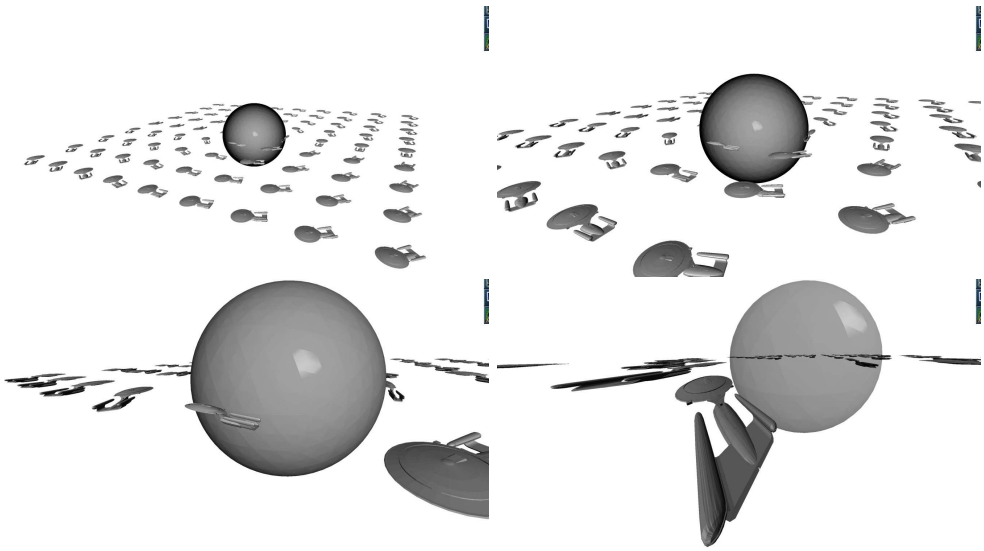


Figure 4.68: (Small) geometrical objects are transformed by the metric tensor field identically to light spheres. The apparently radially crunched spaceship appears undistorted once we move to its location. Then, all other spaceships which are more distant to the black hole appear to be radially stretched.

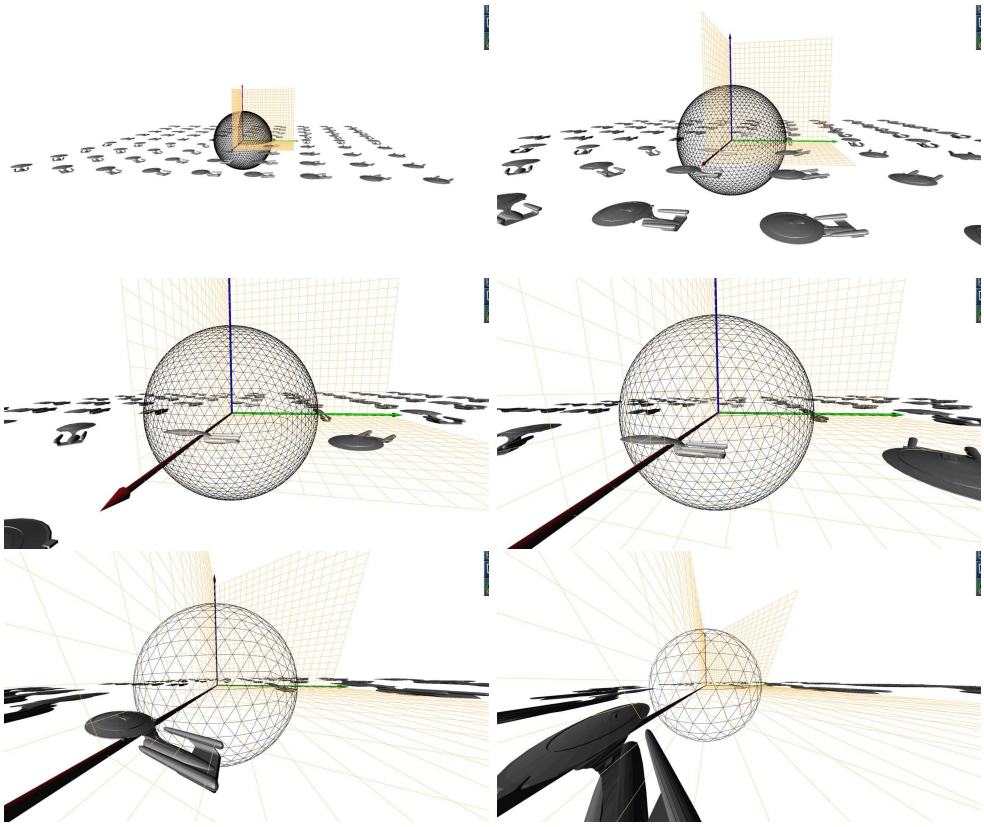


Figure 4.69: When approaching the black hole, its apparent size first increases, similar to approaching a planet from a spaceship. However, as the enter the region of high space curvature, at a certain distance the shape of the BH shrinks again. This indicates that we now "feel" the radially stretched distance - which actually becomes infinite up to the event horizon. The closer we try to come, the further away it appears. This effect is similar to the view of an accelerated observer in special relativity, who might keep the shape of an approached object at constant apparent size by adjusting his acceleration such that the effect of increasing light aberration and decreasing distance just cancel.

Generalization to Four Dimensions

Up to now, we discussed techniques for tensor fields on a three-dimensional manifold. However, as mentioned before, the interpretation of a tensor by inspecting close points as expressed by eq. (4.14) is perfectly suitable for four-dimensional spacetimes with pseudo-Riemannian metric tensor fields as well. Here, we preferably set the constant $\mathcal{C}$ to zero, then $\varepsilon(p_0)$ consists of all lightlike events around the point p_0. In a four-dimensional spacetime, this set of lightlike points is a volume. It is the set of all points which are causally connected to a specific point. Since causal actions travel with the speed of light, it is reasonable to ask for all points which reside the same amount of time in the past. This set of points will be a sphere, or a circle when inspecting a slice of the spacetime. A nice example is given in N.Straumann's book [Str91] in fig. 7.2, where these so-called "navigation circles" demonstrate the causality relationships of the ergosphere around a rotating black hole.

We may compute such local navigation spheres at some point by looking at the causal distance along some spatial direction $\vec{u}$, where $\vec{u}$ is a unit vector in our coordinate system. Light travels along the four-vector $u = (t, \lambda\vec{u})$ and fulfills the condition $g(u,u) = 0$, thus yielding a quadratic expression in λ. It tells us the radial extent of the navigation sphere in the direction $\vec{u}$:

$$0 = g(u,u) = g_{tt}t^2 + 2\lambda t g_{ta}u^a + \lambda^2 \underbrace{g_{ab}u^a u^b}_{-\gamma(\vec{u},\vec{u})} \quad . \tag{4.28}$$

This equation describing the shape of a physical sphere of light after some time t corresponds to an ellipsoid with its center shifted from the origin of the light sphere. We can see this by setting $\lambda\vec{u} := \vec{v} + \vec{w}(t)$ such that only the vector $\vec{w}(t)$ depends on t. Inserting into (4.28) yields

$$0 = g_{tt}t^2 + 2tg_{ta}v^a + 2tg_{ta}w^a - \gamma(\vec{v},\vec{v}) - 2\gamma(\vec{v},\vec{w}) - \gamma(\vec{w},\vec{w})$$

We can bring this equation into the pure quadratic form of an ellipsoid around the origin

$$R^2 = \gamma(\vec{w},\vec{w})$$

with

$$R^2 := g_{tt}t^2 + 2tg_{ta}v^a - \gamma(\vec{v},\vec{v}) \tag{4.29}$$

by setting the term

$$2tg_{ta}w^a + 2g_{ab}v^b w^a = 2tg_{ta}w^a - 2\gamma(\vec{v},\vec{w}) = 0$$

to zero. The vector equation

$$tg_{ta} + g_{ab}v^b = 0$$

can be solved by multiplication with the inverse 3-metric g^{ab} such that

$$v^b = -tg_{ta}g^{ab} =: -t\beta^a$$

where β is (the spatial component of) the shift vector field from 2.2.3. Inserting into (4.29) gives

$$R^2 := g_{tt}t^2 - 2t^2 g_{ta}\beta^a + t^2 g_{ab}\beta^a\beta^b = t^2\left[g_{tt} - g_{ta}\beta^a\right] = t^2\left[g_{tt} + \gamma(\beta,\beta)\right] \equiv t^2\alpha^2$$

with α the lapse function from 2.2.3. Thus,

> *vertex fiber visualization methods for three-dimensional spatial tensor fields can be generalized to a four-dimensional spacetime by drawing a sequence of vertex glyphs shifted by a vector $t\vec{\beta}$ and scaled by $t\alpha$, where t is a parameter of the sequence.*

Of course, not all visualization methods for 3-metrices are equally applicable. Tensor Ellipsoids 4.3.2, Tensor Glow 4.3.2 and Tensor Splats 4.3.2 are good candidates. Multiple icons according to a set of t parameters are to be drawn, whereby the transparency of the icon fades with t^2 - this yields a direct physical interpretation as an "object of light being emitted at a vertex point".

4.3.3 Indirect Visualization

Scalar Properties of Tensor Fields

More experience exists for visualizing scalar fields than for tensor fields - e.g. scalar fields can be rendered directly with graphics hardware and can be interpreted easily. Inspecting the components of a tensor field is straightforward, and can be useful for early detection of outstanding features, e.g. error prone data generation. Some tensor components might also have a direct physical meaning.

However, for a physical understanding of the properties of the underlying tensor field looking at the components is not of much use. fig. 4.70 shows two-dimensional slices from a numerical metric of two colliding black holes, arranged like the matrix representation in cartesian coordinates. Some features

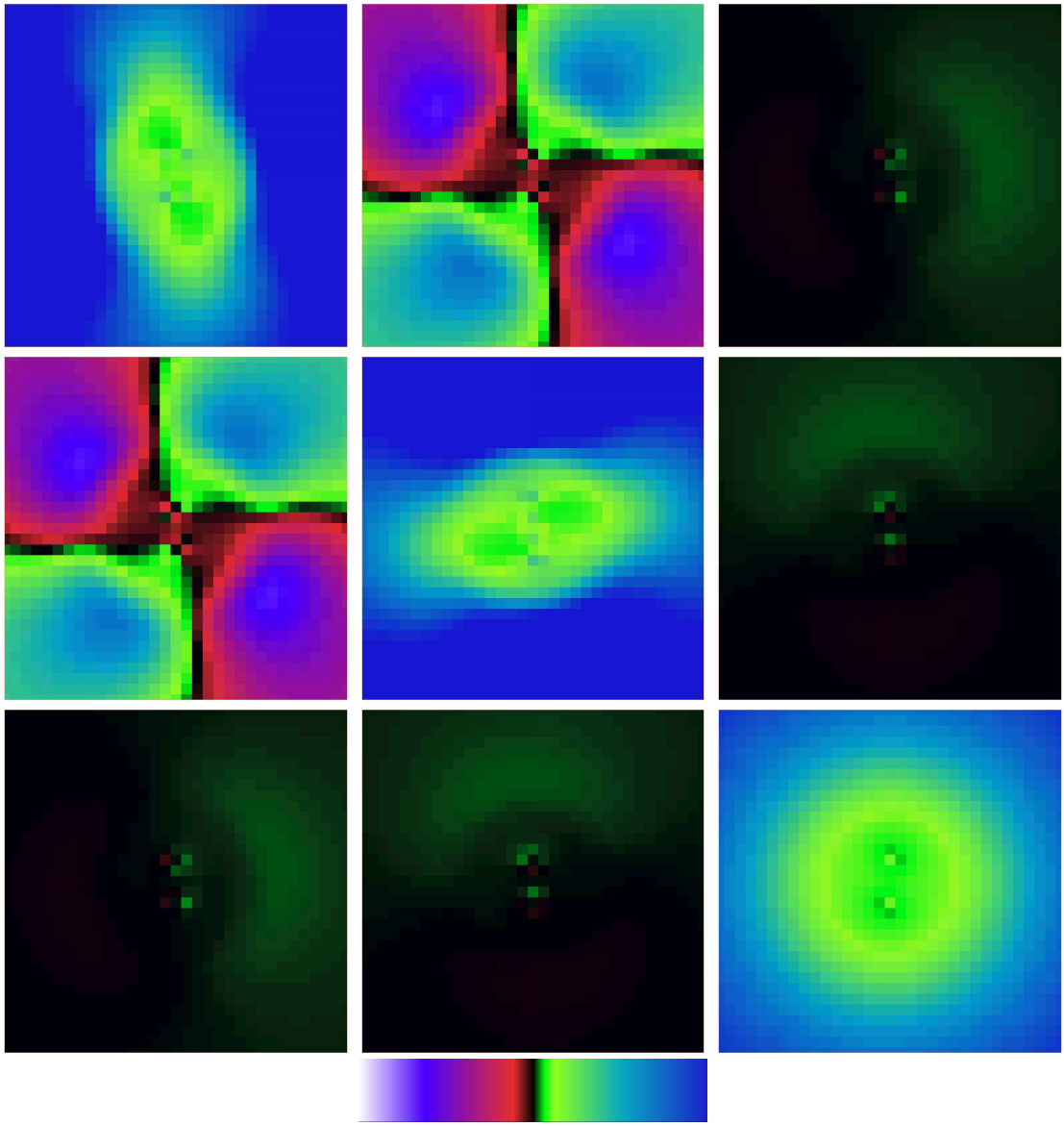


Figure 4.70: The nine components of the (symmetric) metric tensor field of a numerical simulation of two colliding black holes, colors ranging from $[-10, 10]$.

appear to be outstanding, but their interpretation isn't straightforward. A transformation into another coordinate system might help, in particular as we might note some spherical structure. However, whatever coordinate transformation we might employ, the tensor components will depend on this choice and we can never be sure whether we relevant features of the data set or "artifacts" due to the chosen coordinate system.

Inspecting eigenvalues fig. 4.71 of the tensor field provides a coordinate-independent view. For three-dimensional tensor fields we are done with avoiding coordinate-artifacts, but for tensor fields originating from numerical relativity one must also keep in mind that the three-dimensional metric is just a timelike

projection of a four-dimensional metric, so there is one more coordinate choice involved as well. Only a

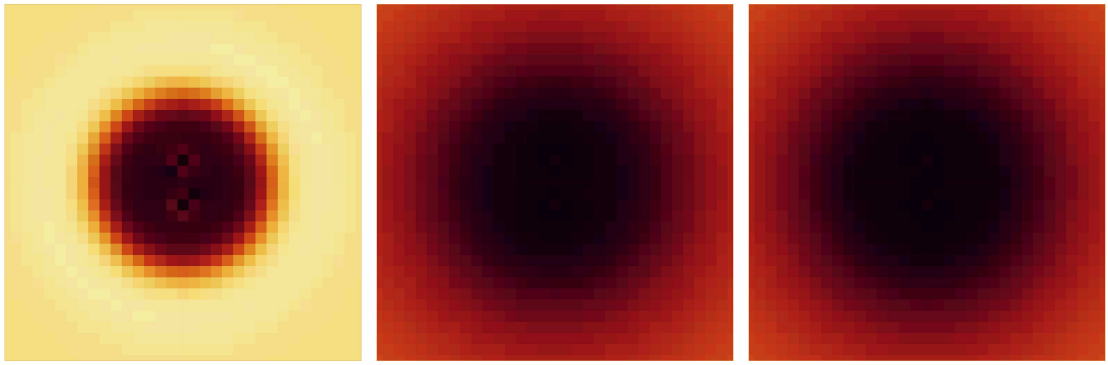
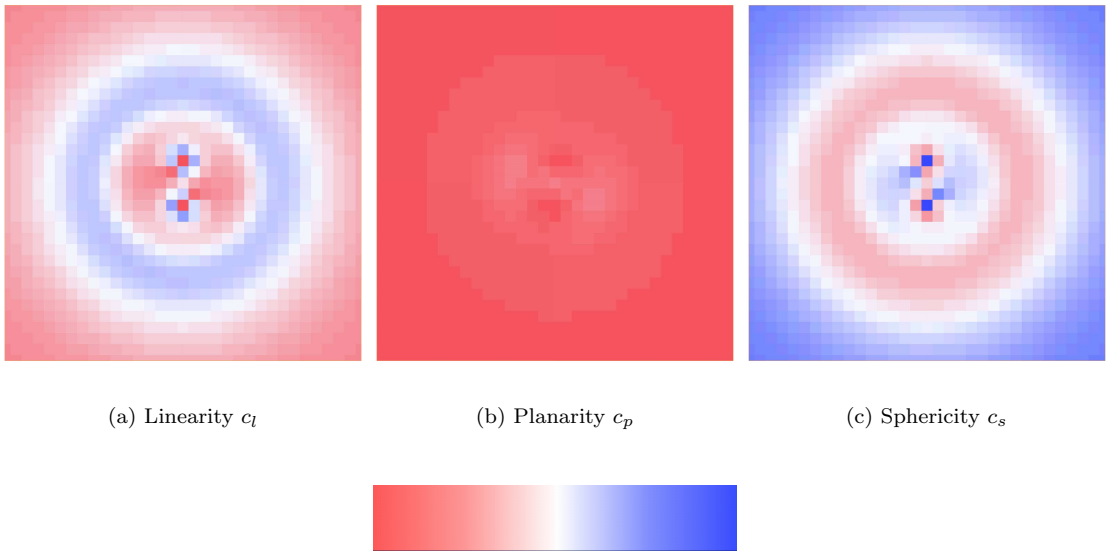


Figure 4.71: Eigenvalue fields derived from the metric field. We see that one eigenvalue is dominant, at least in the outer regions. Due to the absolute scale, we cannot tell if the same is true in the inner region.

full four-dimensional tensor field visualization method were able to depict a complete coordinate-artifact free view. Common three-dimensional tensor field visualization methods, the mere interest from other application areas than general relativity, can only provide a limited insight here, but is still useful to inspect numerical evolution schemes, where the effects of timelike slicing on the projected metric field is of central importance.

Eigenvalue fields still need some more interpretation. More insight might be given by looking at the shape factor fields as described in 2.3.2. The shape factors fig. 4.72 provide information about the number



(a) Linearity c_l (b) Planarity c_p (c) Sphericity c_s

(d) Colormap range $(0, 1)$

Figure 4.72: Tensor shape fields derived from the metric field, displaying the relationships between the eigenvalues. We immediately see that the tensor field is mainly isotropic (spherical) at the borders but contains a ring-like region of highly linear stretching.

of preferred directions in the tensor field and immediately tells if some grid stretching e.g. occurs on one or two directions. Isotropic regions are easily found. What we don't know yet from these images is the actual direction of the stretching and its absolute value.

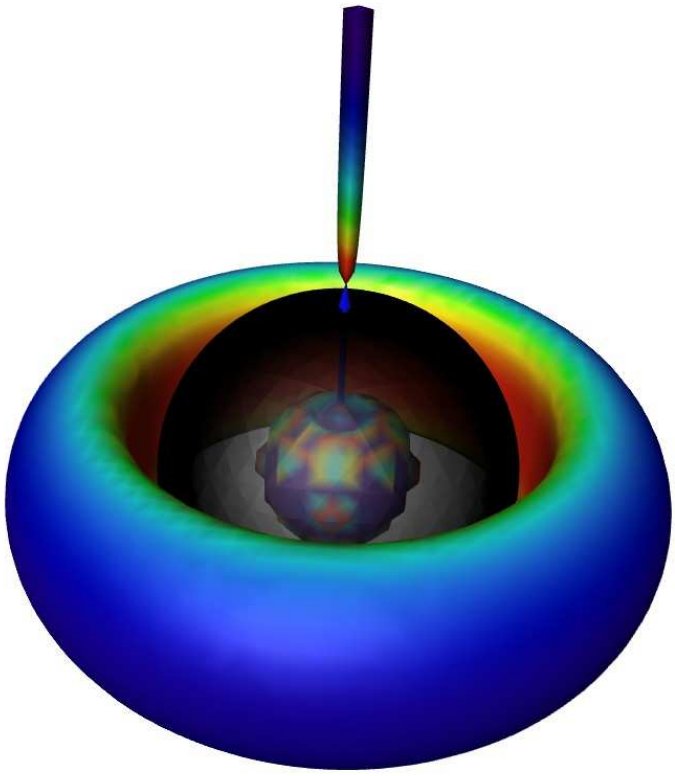


Figure 4.73: Shape Factors in the Kerr metric: Isosurface of the Planarity with Linearity as color encoding. Interestingly, the Planarity is highest in a torus around the Kerr black hole, but neither at the horizon nor far away.

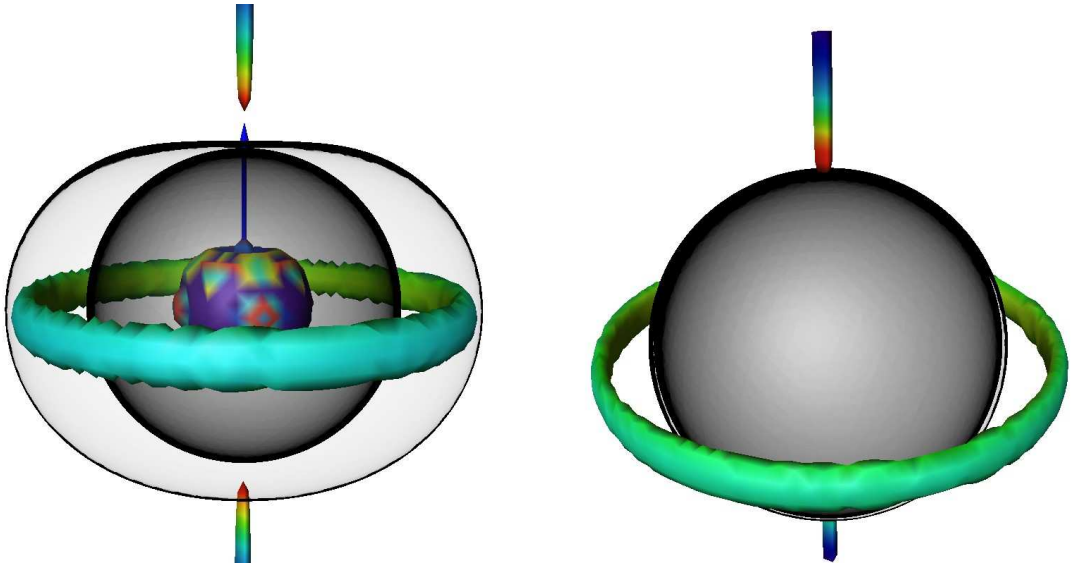


Figure 4.74: Shape Factors in the Kerr metric: The maximal planarity occurs in a toroidal region at the equator outside of the event horizon, but is not related to the Ergosphere. Left: 93% spin, right 30% spin.

Radiation Scalars

In the context of general relativity, the prediction of observable quantities describing gravitational radiation (which can be seen as fluctuations in the metric tensor field) are of special interest for the experimental detection of gravitational waves. However, gravitational energy cannot be "localized" [Str91], it is not possible to define some "density of gravitational energy". At each point in space, the influence of gravitational energy ("acceleration", "gravitational force") on an observer's movement is expressed by the Christoffel symbols (2.34), which can be transformed away by a proper choice of coordinates, see 2.1.2. So when some definition of gravitational energy were based on the Christoffel symbols, some observers would see gravitational energy, others don't. Thus in an arbitrary spacetime one cannot be sure if the resulting quantity is due to gravitational radiation, or if we are just looking at flat, radiation-free space in a weird coordinate system. Coordinate-invariant definitions are thus essential.

This search has only been carried out within the context of general relativity, but it might be of interest to apply the ideas also to other tensor fields from other scientific domains. In general relativity these fields are used to inspect properties of the 4-metric of signature $(+,-,-,-)$, similarly adjusted definitions may be useful to inspect positive definite 3-tensor fields like the diffusion tensor field in medicine or the stress tensor field in computational fluid dynamics.

Curvature Invariants

While the first derivatives of the metric tensor field – and with it the Christoffel symbols – can be transformed away, its second derivatives cannot be brought to zero by a proper choice of coordinates. I.e. the derivatives of the Christoffel symbols, which constitute the Riemann tensor $R^{\delta}_{\alpha\beta\gamma}$, cannot be transformed away. Therefore scalar expressions which are derived from the Riemann tensor provide coordinate invariant descriptions of the underlying metric tensor field. They at least involve the second derivative of the tensor field.

From the 256 components of the rank-4 Riemann tensor, only 20 are independent due to its symmetry properties. Contraction of the upper and lower index of the Riemann-tensor yields the rank-2 Ricci tensor $R_{\alpha\beta} = R^{\gamma}_{\alpha\beta\gamma}$. Due to the symmetry of the Riemann tensor $R^{\delta}_{\alpha\beta\gamma} = R^{\delta}_{\beta\alpha\gamma}$, the Ricci tensor is symmetric as well and contains 10 independent components. The other 10 components of the Riemann tensor construct the trace-free rank-4 Weyl tensor $C^{\gamma\delta}_{\alpha\beta}$. The trace of the Ricci tensor $\mathfrak{R} = R^{\alpha}_{\alpha}$ is called the curvature scalar. It is part of the Einstein Field equations of the gravitational field and is zero in the absence of matter. Thus, it is a suitable indicator for matter density, but not for pure gravitational radiation which propagates in vacuum spacetimes.

Other scalar invariants can be constructed from the Riemann tensor. In a general spacetime possibly containing some matter distribution there are fourteen independent scalars [Pet69]. In vacuum the Ricci tensor is zero, so any indicator of gravitational radiation must be based on the Weyl tensor. By proper choice of coordinates, any 4 of the 10 independents components of the Weyl tensor can be brought to zero, leaving 6 fundamental real invariants known as the eigenvalues of the Weyl tensor. Since the trace of the Weyl tensor and its dual are zero, just four scalar values suffice to reconstruct all six values. These 4 so-called Riemann invariants [Wit59] can be expressed as the two complex quantities:

$$I := \frac{1}{16}\left(C^{\gamma\delta}_{\alpha\beta}C^{\alpha\beta}_{\gamma\delta} - iC^{\gamma\delta}_{\alpha\beta}\frac{1}{2}\varepsilon^{\mu\nu}_{\gamma\delta}C^{\alpha\beta}_{\mu\nu}\right) \tag{4.30}$$

and

$$J := \frac{1}{96}\left(C^{\gamma\delta}_{\alpha\beta}C^{\mu\nu}_{\gamma\delta}C^{\alpha\beta}_{\mu\nu} - iC^{\gamma\delta}_{\alpha\beta}C^{\varrho\sigma}_{\gamma\delta}\frac{1}{2}\varepsilon^{\mu\nu}_{\varrho\sigma}C^{\alpha\beta}_{\mu\nu}\right) \quad . \tag{4.31}$$

They are the only independent, algebraic curvature scalars one can construct in a vacuum spacetime. However, they are not yet appropriate as a measure of gravitational radiation, since e.g. in the radiation-free Kerr-metric they are still non-zero [BB02].

Newman-Penrose Formalism

The Newman-Penrose Formalism [NP62, Ste91] encodes the ten independent components of the Weyl tensor into five complex quantities denoted by Ψ_i, called the curvature components. They are independent from the choice of coordinates, but allow a free choice "in which direction" the curvature components shall be read off. This choice determines the appropriate physical interpretation of the resulting numbers, like "radially outgoing" and "radially ingoing" gravitational energy (as depicted by fig. 4.75). However,

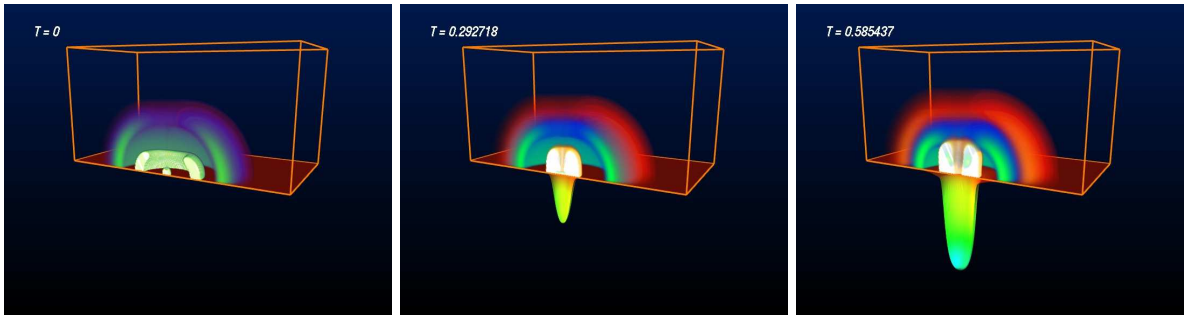


Figure 4.75: Outgoing and ingoing gravitational radiation for a subcritical "Brill wave" (2.3.1). Initially, both components are overlayed at the same geometrical location, thus appearing violet. During evolution, the incident blue Ψ_0 and outgoing red Ψ_4 component separate. The lapse function α, indicating strong gravitation, is displayed as height field below. Images based on these quantities have been used for many book and magazine publications [Sch03, Ben01a, Smo99, Ben03, Ben02b, Eng02, Ben00, Gre99].

apparently straightforward interpretations like "radially" are problematic if there is no radial symmetry at all in a spacetime. Thus, such interpretations are meaningful only for spacetimes that are asymptotically flat at infinity and contain a reasonably defined "radial" direction.

The choice of "interpretation directions" is specified via one time-like (t^μ) and three space-like (x^μ,y^μ, z^μ) vector fields that are orthonormal with respect to the given metric $g_{\mu\nu}$. I.e. the vectors $\{v^\mu_{(i)}\} = \{t^\mu, x^\mu, y^\mu, z^\mu\}$ at each point obey the relationship $v^\mu_{(i)} g_{\mu\nu} v^\nu_{(j)} = \eta_{ij}$. Note that $\{x, y, z\}$ do not imply cartesian coordinates, they may as well denote vector fields oriented along polar coordinates $\{r, \vartheta, \varphi\}$. See fig. 4.76 for demonstration of the Newman-Penrose quantities evaluated using a cartesian and a polar tetrad.

The metric can be expressed via the corresponding covector fields $v_{(i)\,\mu} = g_{\mu\nu} v^\nu_{(j)}$ as $g_{\mu\nu} = t_\mu t_\nu - x_\mu x_\nu - y_\mu y_\nu - z_\mu z_\nu$. A complex null tetrad is constructed from these via

$$k_\mu = (t_\mu - z_\mu)/\sqrt{2}, \tag{4.32}$$
$$l_\mu = (t_\mu + z_\mu)/\sqrt{2}, \tag{4.33}$$
$$m_\mu = (x_\mu - i\, y_\mu)/\sqrt{2}, \tag{4.34}$$
$$\bar{m}_\mu = (x_\mu + i\, y_\mu)/\sqrt{2}, \tag{4.35}$$

where all vectors $\{l, k, m, \bar{m}\}$ are null, l, k are real and $\bar{m}, m$ are complex, whereby $\bar{m}$ is the conjugate of m. This null tetrad is normalized such that $g(l, k) = -1$ and $g(m, \bar{m}) = 1$, whereas all other products vanish. The Newman-Penrose quantities are then constructed from this complex null tetrad $\{k, l, m, \bar{m}\}$ via contraction with the Weyl tensor:

$$\Psi_0 = C_{\mu\nu\lambda\xi} m^\mu k^\nu m^\lambda k^\xi \tag{4.36}$$
$$\Psi_1 = C_{\mu\nu\lambda\xi} k^\mu l^\nu k^\lambda m^\xi \tag{4.37}$$
$$\Psi_2 = \frac{1}{2} C_{\mu\nu\lambda\xi} k^\mu l^\nu \left(k^\lambda l^\xi - m^\lambda \bar{m}^\xi\right) \tag{4.38}$$
$$\Psi_3 = C_{\mu\nu\lambda\xi} l^\mu k^\nu l^\lambda \bar{m}^\xi \tag{4.39}$$
$$\Psi_4 = C_{\mu\nu\lambda\xi} l^\mu \bar{m}^\nu l^\lambda \bar{m}^\xi \quad . \tag{4.40}$$

The Riemann invariants I (4.30) and J (4.31) are related to the these quantities via

$$I = \Psi_0\Psi_4 - 4\,\Psi_1\Psi_3 + 3\,{\Psi_2}^2 \quad \text{and} \quad J = \begin{vmatrix} \Psi_0 & \Psi_1 & \Psi_2 \\ \Psi_1 & \Psi_2 & \Psi_3 \\ \Psi_2 & \Psi_3 & \Psi_4 \end{vmatrix} .$$

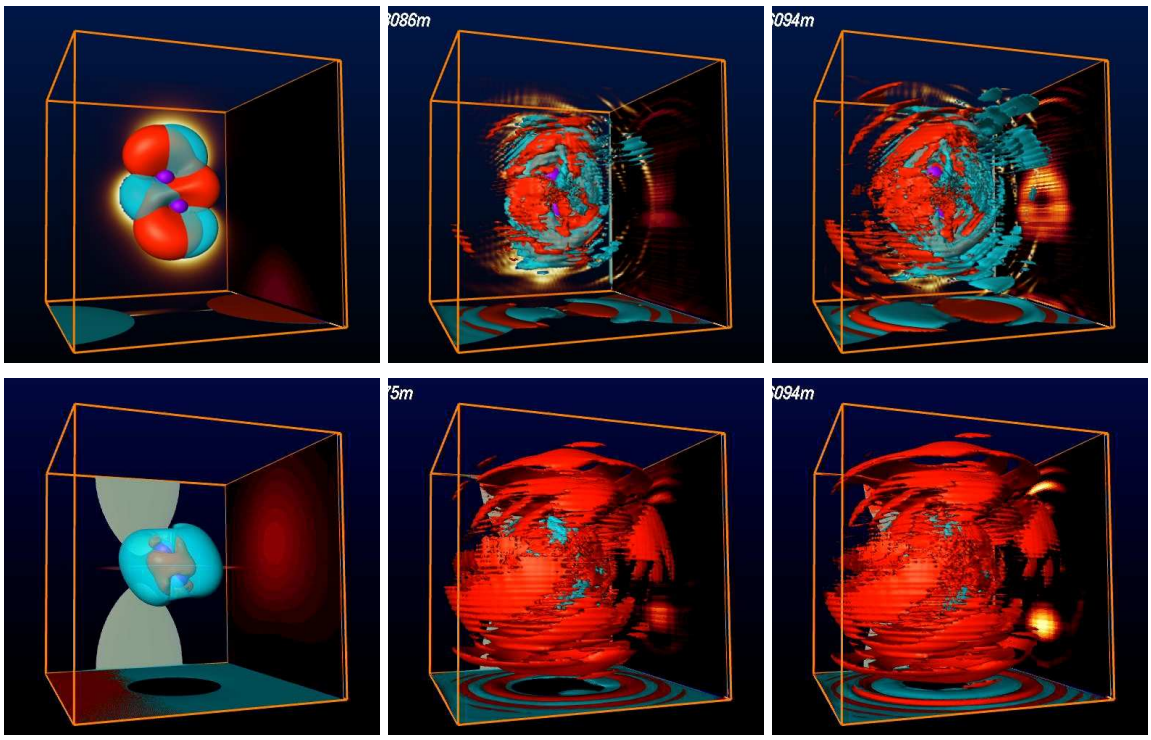


Figure 4.76: "Noisy" numerical dataset of gravitational energy released from a neutron star merger. Isosurfaces of $\Re\Psi_4$ displayed in red, $\Re\Psi_0$ displayed in blue. Upper row: cartesian tetrad, $\Re\Psi_4$ depicting "leftwards", $\Re\Psi_0$ "rightwards" moving structures. Lower row: polar tetrad, $\Re\Psi_4$ depicting radially outwards, $\Re\Psi_0$ radially inwards moving structures.

The Newman-Penrose quantities are used as an indicator of gravitational waves as they have a certain physical interpretation in the case of an asymptotically flat spacetime with radial symmetry. The Newman-Penrose formalism allows to describes the Weyl tensor in a spinor basis with spin 2. The *peeling theorem* (see e.g. Stewart [Ste91], p.130, for a general prove on arbitrary fields) states that $\Psi_i = \mathcal{O}(r^{-(2s+1-i)})$, thus

	Radial falloff behavior	Interpretation using $\Psi_i r^3$
Ψ_0	$\propto 1/r^5$	Ingoing radiation, $\propto 1/r^2$
Ψ_1	$\propto 1/r^4$	
Ψ_2	$\propto 1/r^3$	Equivalent of a rest mass, constant
Ψ_3	$\propto 1/r^2$	
Ψ_4	$\propto 1/r$	Outgoing radiation, $\propto r^2$

The real and imaginary part of the Newman-Penrose quantities corresponds to the even and odd parity of a gravitational wave radiation term as written in a Taylor series. The Newman-Penrose quantity Ψ_4 is the preferred scalar field when inspecting data from numerical relativity to visualize the "outflowing gravitational energy", see fig. 4.77. For an axisymmetric collision of black holes the imaginary part $\Im(\Psi_4)$ vanishes, so this one is an indicator of non-axisymmetric properties of a gravitational wave.

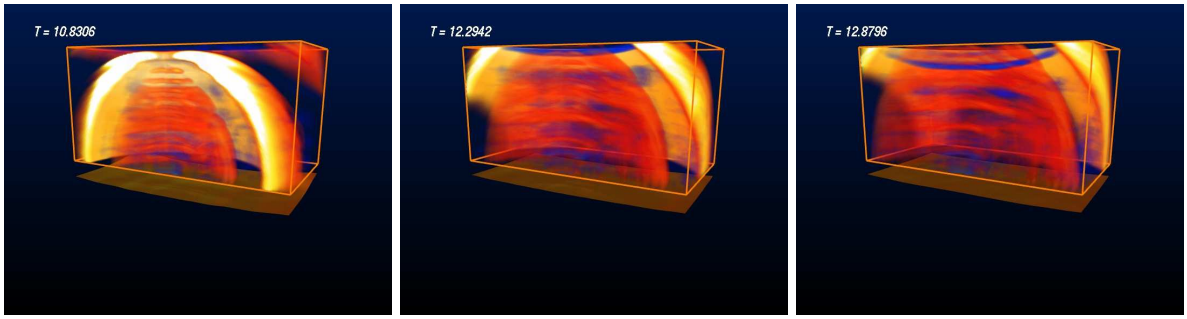


Figure 4.77: An outflowing intensity peak of gravitational radiation, rendered in red (Ψ_4), is undesirably reflected at the grid boundary, shown as blue (Ψ_0) ingoing bow. Visualization of Ψ_0 and Ψ_4 helps to detect such numerical artifacts. (Late time of fig. 4.75.)

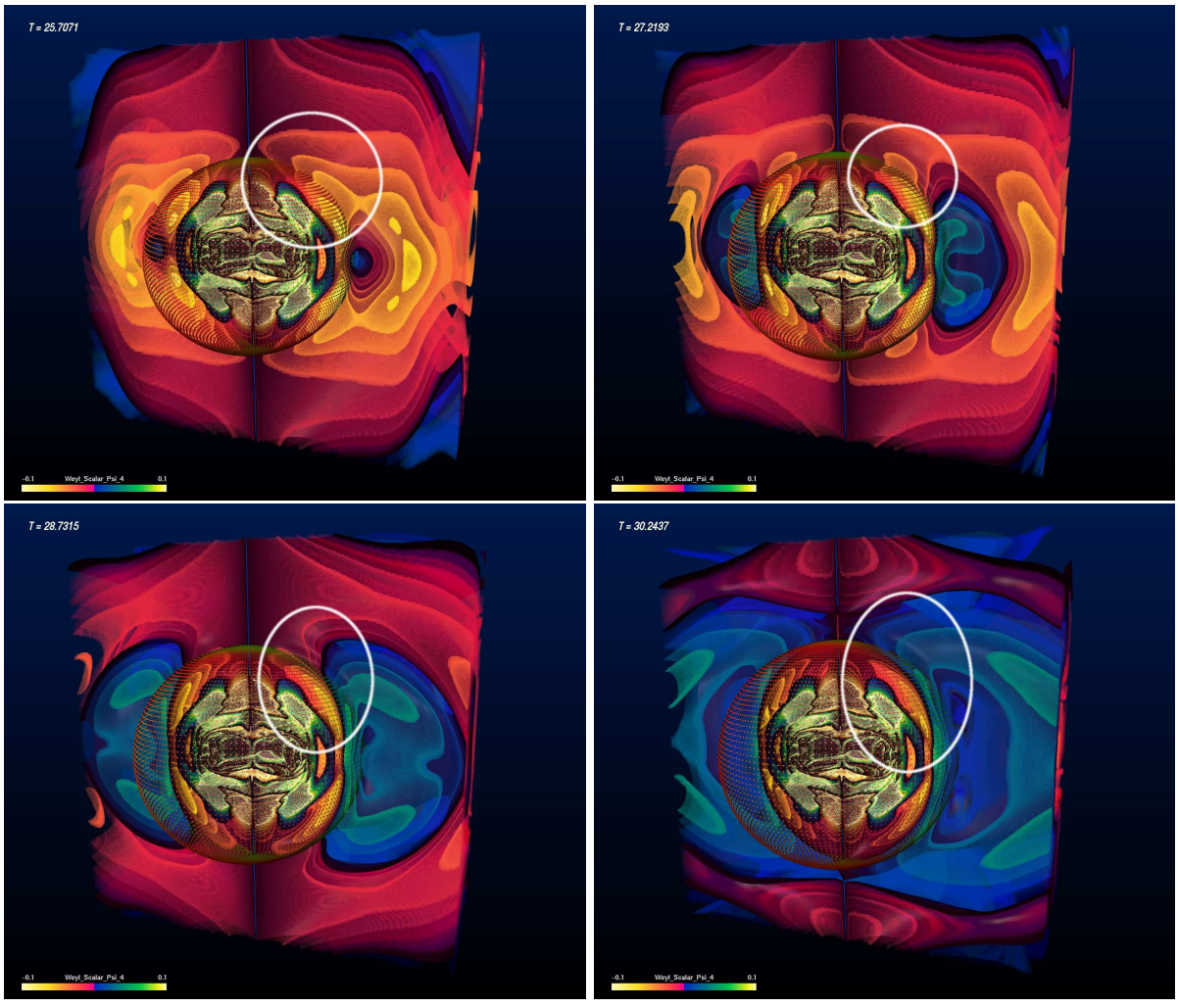


Figure 4.78: An example where using Ψ_4 gives an hint about the location of the **event horizon**, as compared to the apparent horizon: certain structures visible in the Ψ_4 field appear to fall inwards, even though they originally are born outside of the apparent horizon. Watch the red-yellow intense structure in the marked region falling in.

Volume Rendering Lazarus Data

The Lazarus project, as it has been called by AEI researchers C.Lousto, J.Baker and M.Campanelli [JB02], is an effort to merge the methods from analytic perturbation theory with full numerical methods. For strong gravitational fields, like the merger of black holes, which are beyond the capabilities of perturbative approaches, full numerical simulations are performed. When this evolutions fades down into some weaker regime where the full numerical treatment becomes troublesome (even a static situation can be numerically instable) but perturbative approaches are valid again, the perturbative analysis takes over and may continue the computation of the numerically given spacetime.

These Lazarus data provide the complex Weyl Scalar Ψ_4, as an indicator of the outgoing gravitational radiation. They are given in polar coordinates as modes of a decomposition of the three-dimensional spacetime around the rotation axis of orbiting black holes ($\vartheta = \pi/2$ corresponds to the orbit plane):

$$\Psi_4(t, \bar{r}, \vartheta, \varphi) = \sum_{m=0}^{\infty} \Psi_4^m(t, \bar{r}, \vartheta) e^{im\varphi}$$

The selected radial coordinate $\bar{r}$ is similar to the radial coordinate in the Eddington-Finkelstein coordinates of a static black hole:

$$\bar{r} = R + 2m \ln\left(\frac{R}{2m} - 1\right)$$

where r is the radial coordinate of the numerical cartesian x, y, z grid and m is the mass of the spacetime as computed from the initial numerical spacetime (which actually by itself is the result of the numerical evolution). In this coordinate system negative values of $\bar{r}$ are possibleDue to the logarithmic scale, a large domain far outside the original numerical spacetime can be covered, which is essential to read off the gravitational wave forms generated by some gravitational strong event.

For visualization only the $m = 2$ mode $\Psi_4^2(t, \bar{r}, \vartheta)$ is used, because the other modes are mostly drowned out by numerical noise. The full three-dimensional domain has to be reconstructed by adding the φ-term, and the resulting location in polar coordinates has to be transformed into cartesian coordinates for final display. For employing standard visualization algorithms like volume rendering, interpolation of the data onto a uniform cartesian grid is required.

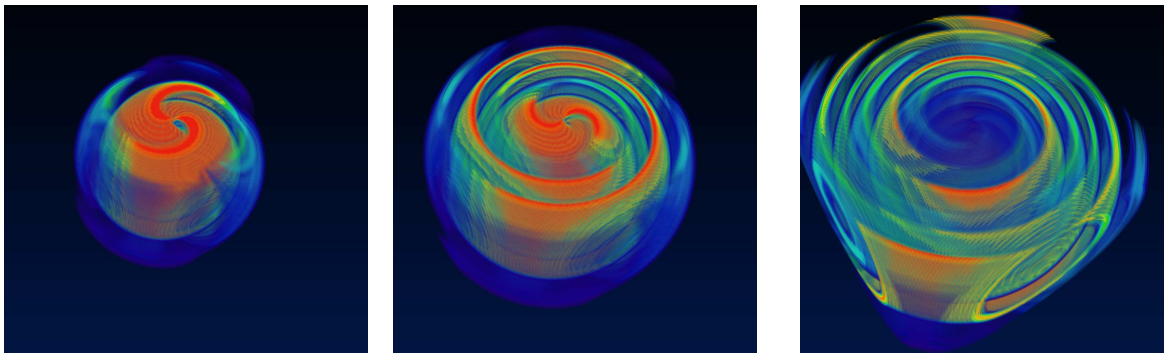


Figure 4.79: An outspiraling gravitational wave, as computed by the methods of the Lazarus project [JB02] and visualized using fiber bundle operations.

The procedure to perform the required visualization implemented upon the fiber bundle data model as follows:
Task: Data from the Lazarus project are given as a complex scalar field on 2D axial coordinates $\{r, \vartheta\}$, where ϑ is the usual altitude polar angle and r is the Eddington-Finkelstein radial coordinate. A three-dimensional field may be constructed from these two-dimensional data by adding a $e^{-i2\varphi}$ term to the data.These reconstructed data are thus uniform in polar 3D $\{r, \vartheta, \varphi\}$ coordinates. We require these data on a uniform cartesian grid.

Input: The uniform $\{r, \vartheta\}$ axial coordinate positions and a complex value at each point.

Output: Data uniformly distributed in cartesian coordinates..

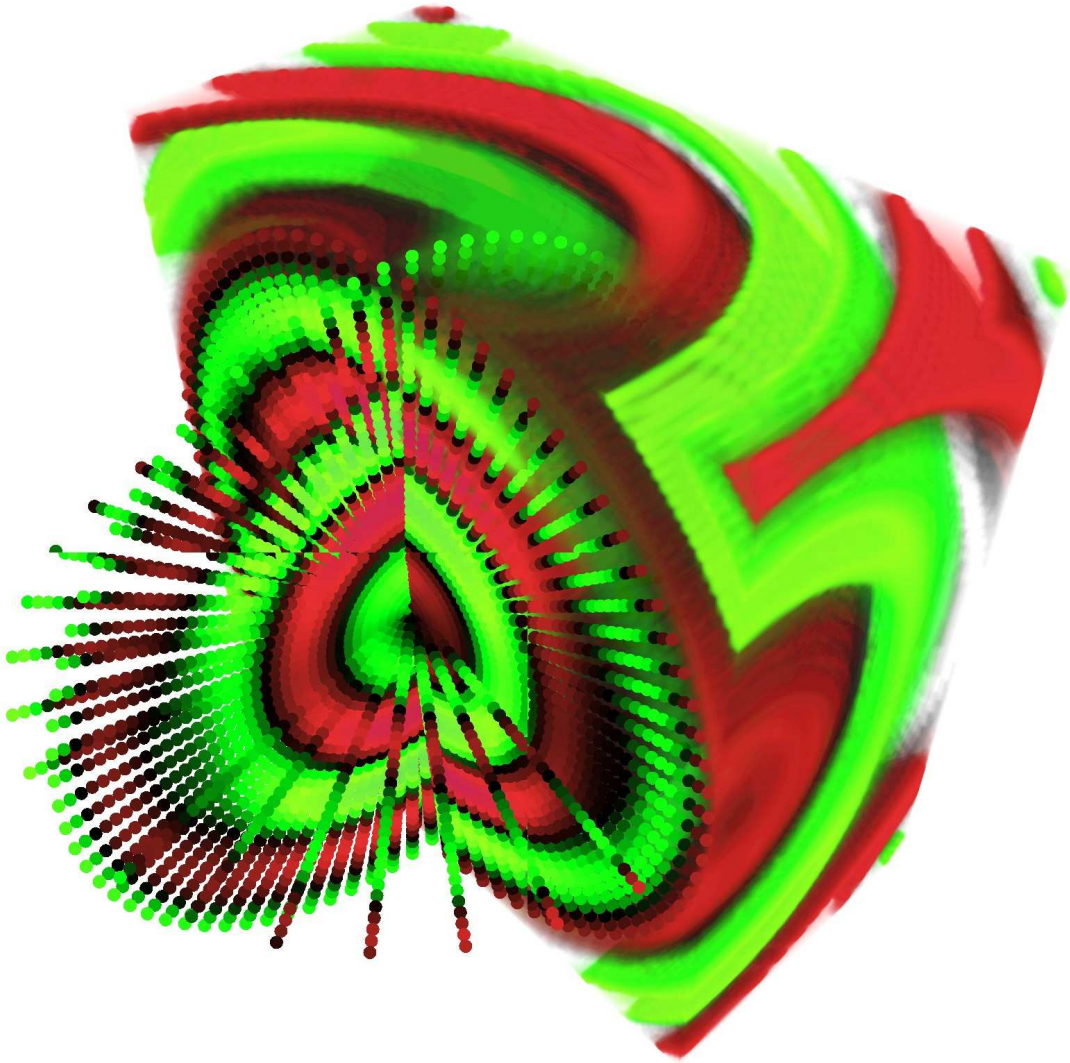


Figure 4.80: Volume rendering data from the "Lazarus" project: data are given uniformly in a two-dimensional (r, ϑ) coordinates, procedurally extended to cover a polar grid (illustrated by the points) and need to be interpolated onto a uniform cartesian grid.

Procedure:

1. Construct complex polar field $\{r, \vartheta, \varphi\}$ data field via $\Psi(r, \vartheta) \cdot e^{-im\varphi}$
2. Transform coordinates of polar points into cartesian coordinates
3. Compute interpolation weights from the target grid (uniform cartesian coordinates) to the data source grid (transformed cartesian positions)
4. Evaluate the data field on the source grid on the target grid via the interpolation weights.

$$\begin{array}{ccc} \mathbf{R_{dst}}\{\mathbf{x}, \mathbf{y}, \mathbf{z}\} & \xleftarrow{\text{eval}} & R_{src}\{x, y, z\} \\ & & \text{coordinate} \uparrow \text{transformation} \\ & & R_{src}\{r, \vartheta, \varphi\} \xleftarrow[e^{im\varphi}]{\text{explicit coding}} \mathbf{R_{src}}\{\mathbf{r}, \vartheta\} \end{array}$$

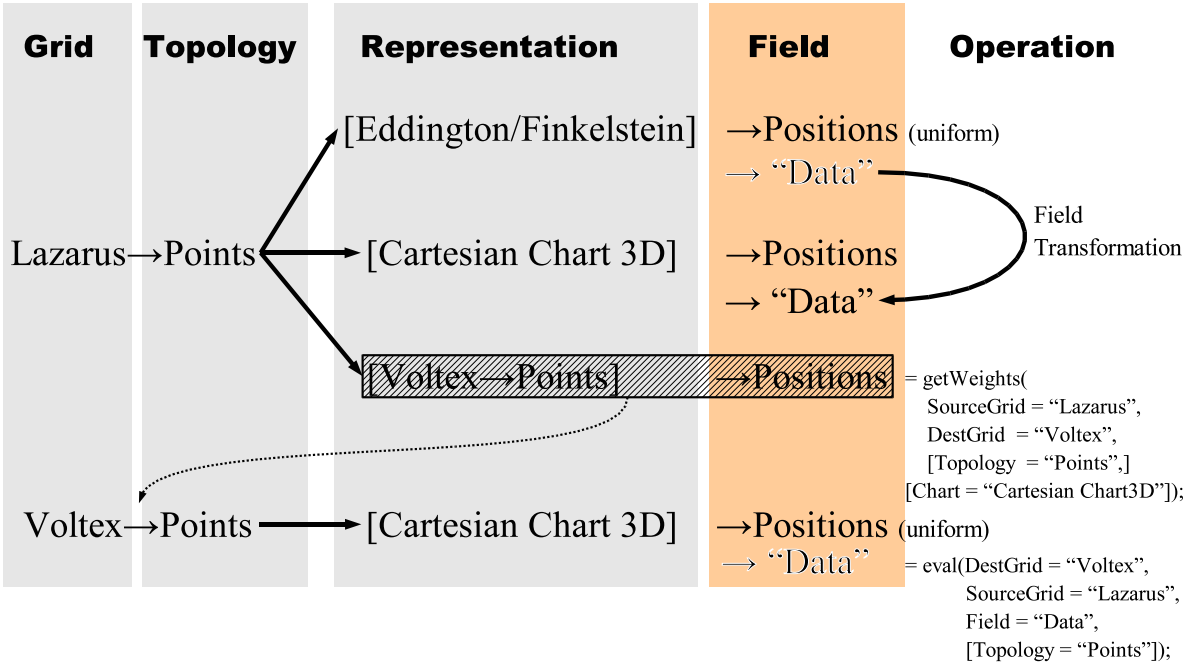


Figure 4.81: Fiber bundle objects and operations involved in providing data stemming from the Lazarus project for a volume renderer. Images created by this method can be found in [Ben01b, Geo01, Ben02a].

Appendix A

Data Description and Algorithms

A.1 Examples of Fiber Bundle Data Model Descriptions

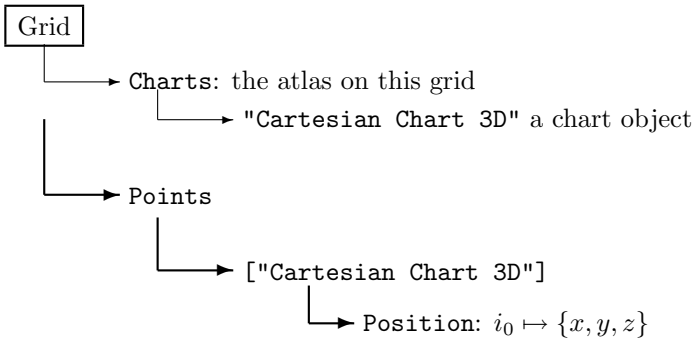
In this section a conceptual overview of how to describe various grid types is given. A specific grid type is shown as a graph below the Grid group (section 3.1.1). As the data model is constructed by hierarchically nested associative maps (3.1.2), at each leaf an identifier is mapped to the next subgroup object. The following notations are used within the graphical layout describing a certain grid type:

- `Text` The identifier is a predefined, intrinsically known value. It must be exactly this value.
- `"Text"` The identifier is a freely selectable text. Restrictions are that predefined values must not be used here. If the same identifier occurs in the same graph, then all occurrences must be identical.
- `[Text]` The identifier is a reference to an object named "Text". If there is no such entry in the graph, it refers to an external object, mostly chart objects. This notation is usual for representation identifiers.
- `T=23.4` A floating point number identifier; it is used for slices within a bundle group.
- `(Text)` Parenthesis around an identifier mean that this subgroup is a procedural one, ie. the data within here can be computed on-the-fly from some parameters which are stored in this group instead of a set of elements
- $i_0, j_0, k_0, \ldots$ indices from the Index Space of depth 0, ie. point-related data
- $i_1, j_1, k_1, \ldots$ indices from the Index Space of depth 1, ie. cell-related data
- $\{a, b, c\}$ a set of exactly three elements
- $\{a, b, c, \ldots\}$ a set of a varying number of elements
- hierarchy entries within the atlas of a grid are rendered with thin lines, whereas the topology subgroups of a grid have thick lines in the graph.

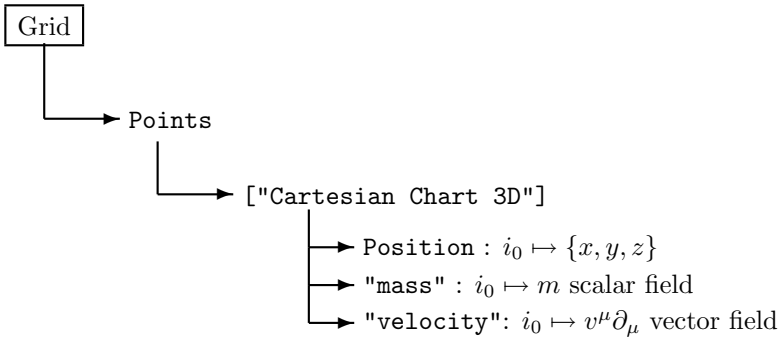
A.1.1 Particle Sets

Point Sets

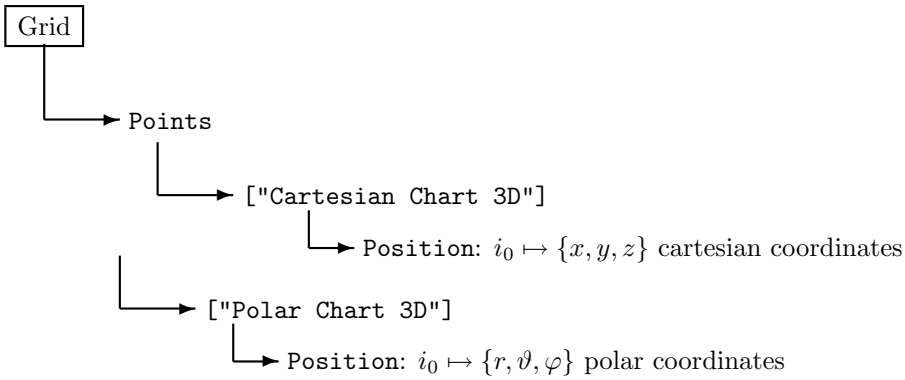
A set of points in a cartesian chart:



The grid atlas will not be shown in subsequent graphs, because it is mostly trivial information. The grid atlas only is of relevance if multiple coordinate systems are used with known transformation rules. With data fields defined on the point set, the grid atlas is omitted:

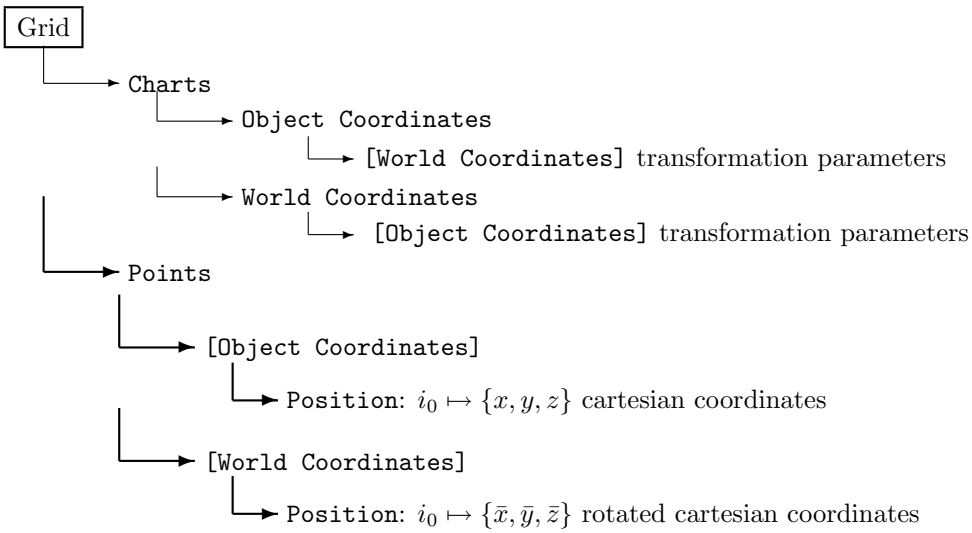


Point set with coordinates also known in a polar chart:



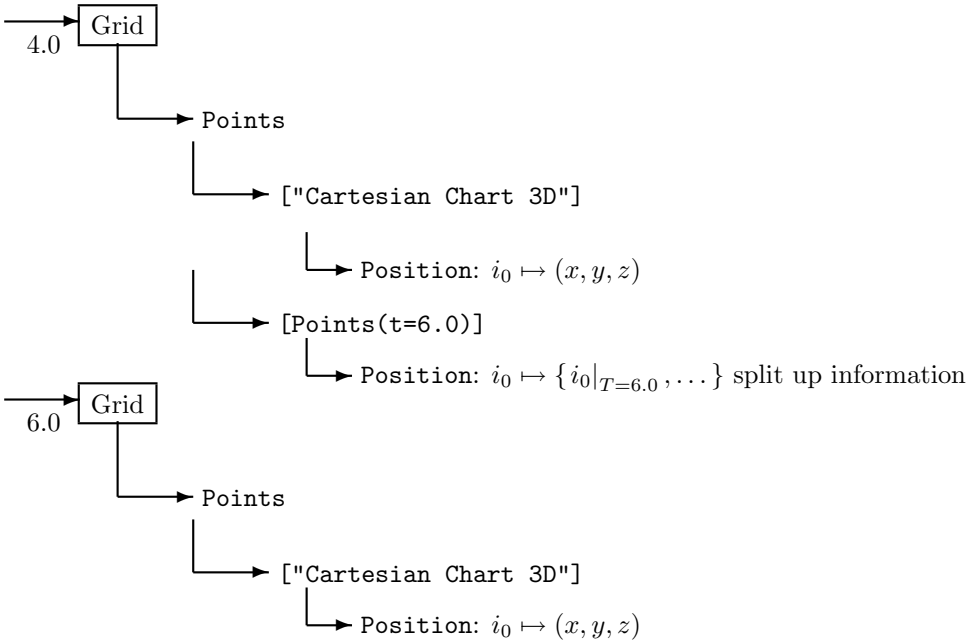
Scalar fields are independent from the coordinate systems and could thus be put into the parent Topology group instead of a coordinate-specific representation. However, it is simpler to store all data fields in the representation groups, such that some code only needs to inspect this group when seeking for a data field. An implementation is assumed to be able to share the actual data, such that there is no significant memory overhead.

The coordinate transitions from the `Polar Chart 3D` to the `Cartesian Chart 3D` may be done via a usual standard transformation rule. However in general a transformation rule may involve some parameters, eg. when relating two cartesian charts via some rotation matrix. This rotation matrix (or the transformation parameters in general) may be time-dependent, so it needs to be stored at least in some slice group. Some coordinate system might be fixed for a specific grid object, so it makes sense to store transformation parameters with each grid (eg. object-coordinates, world-coordinates). An example of a grid atlas containing some transformation rules, dealing with multiple coordinate systems and transformation rules:



Particle Trajectories and Geodesics

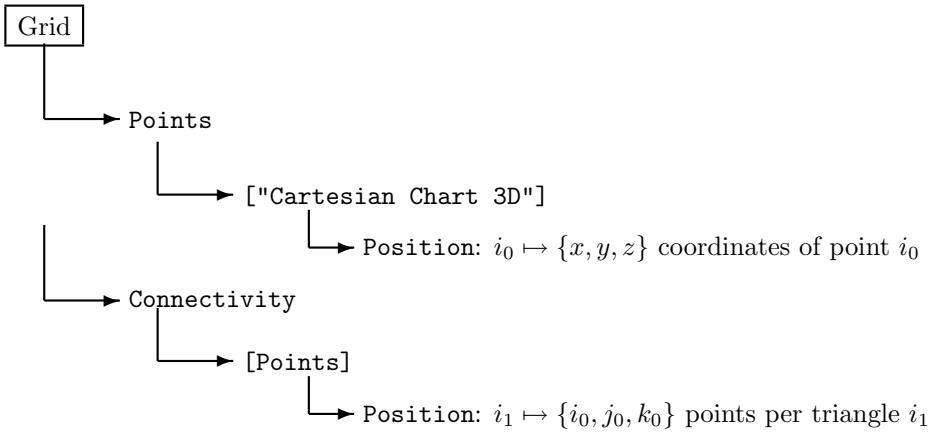
When a set of particles is evolving in time, some of these particles might leave the grid, others merge, split up or are newly introduced. It is required to track this behavior. This is done by explicitly specifying a representation of the successor's Point Topology in a Point Topology. The positional data set of this representation maps points from one time step to the next (or previous) ones.



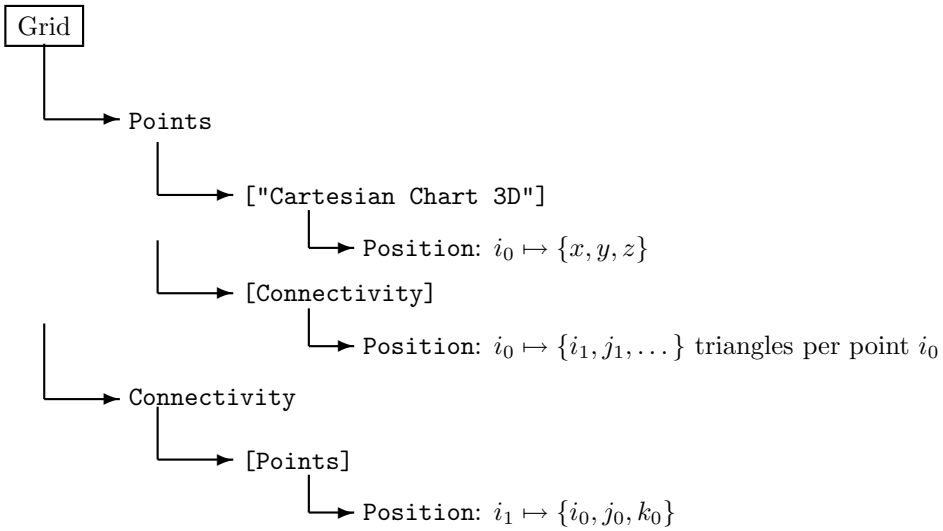
A.1.2 Two and Three-Dimensional Geometries

Triangular Surfaces

A surface constructed by triangles:

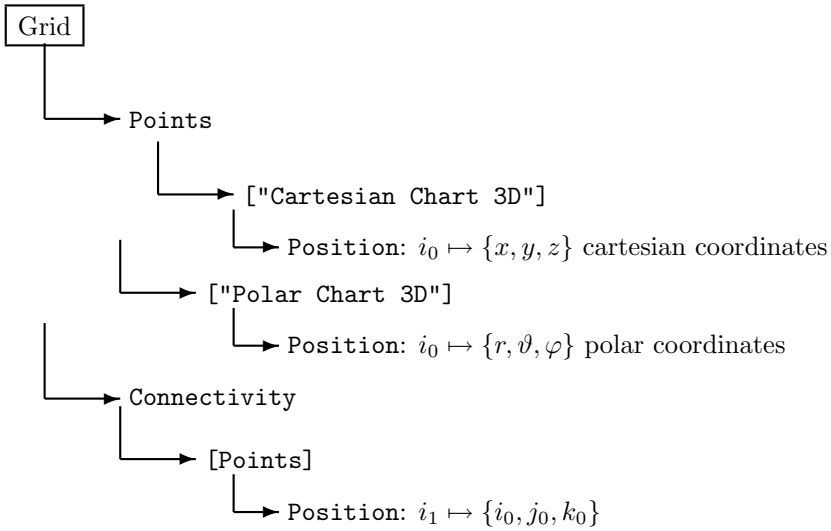


A surface with additional reverse information about the triangles around each point:

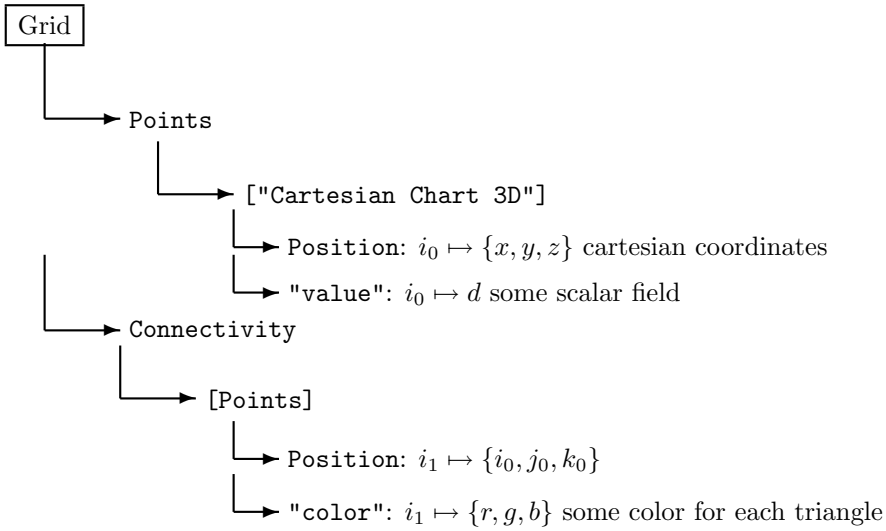


The reverse triangles-per-point information can be computed from the points-per-triangle array by an inversion operations. Note that the representer of the triangles-per-point representation is the "Connectivity" Topology object on the same grid.

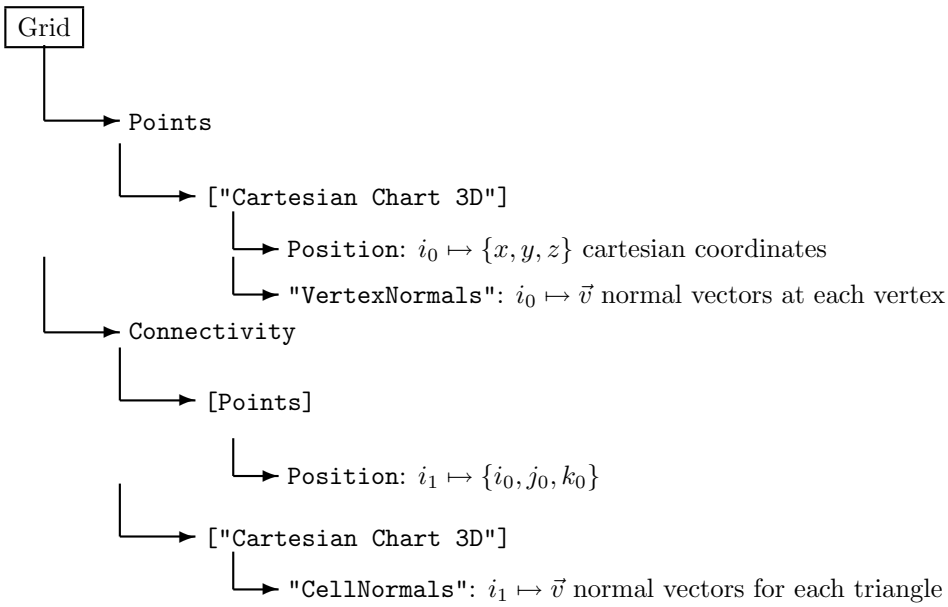
A triangular surface with points given in polar and cartesian coordinates:



A triangular surface with a data field given on the points and another one on the cells:



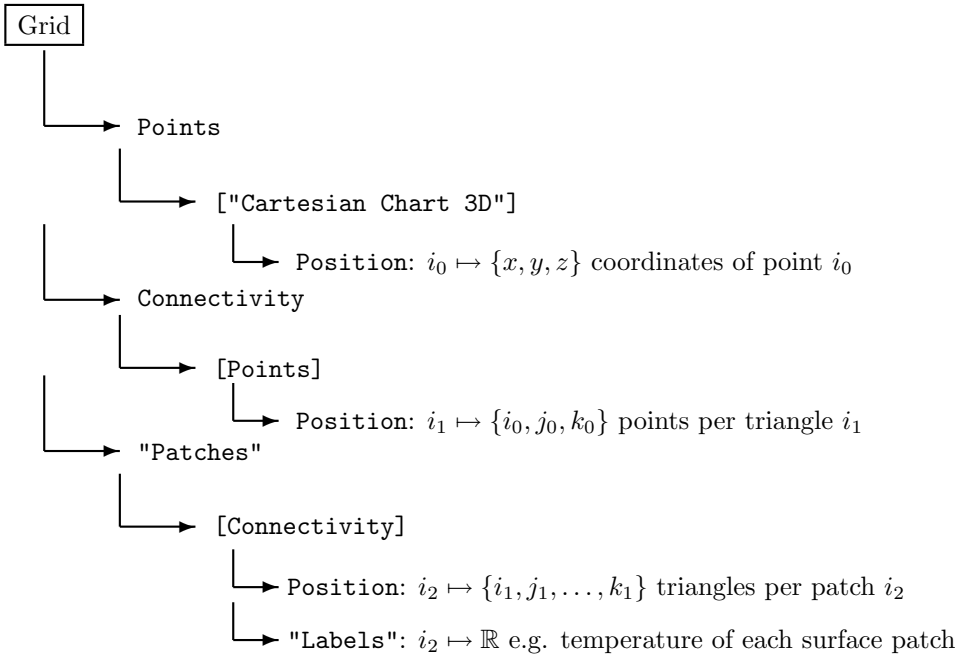
It makes sense to store cell fields, which are related to a chart, in the representation of this cell in this chart. An example are normal vectors of the cells:



The semantics of the "Positions" of a cell representation in a chart might be the barycenter of each triangle.

Multipatch Triangular Surfaces

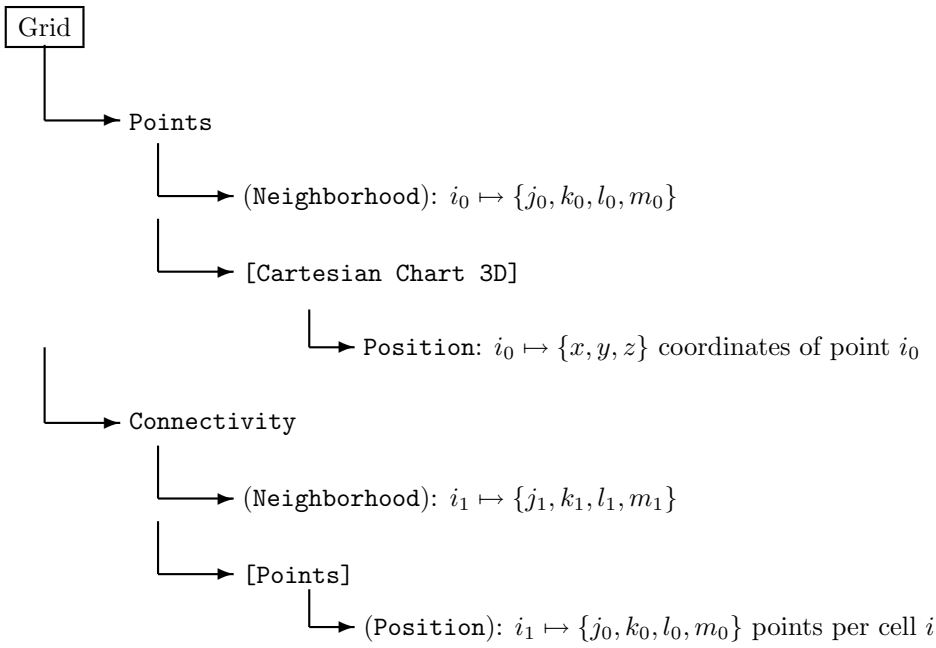
A triangular surface with some cell complexes that contain certain data, e.g. some labeling of surface regions, called multiple "patches":



The patches on the surface are described via a set of triangles, that is casted into a "Patches" Topology object of index depth 2. Here, the surface patch is given by triangles, but another representation could also refer directly to the related vertices. In this case the index depth 2 indicates that the representation of a surface patch via vertices is incomplete, i.e. there needs to be additional structural (topological) information of index depth 1. Otherwise, the patch would be indistinguishable from a polygon. As a consequence, a polygon algorithm can be applied to the vertex representation of a patch, but patch-specific algorithms cannot be applied to polygons, because these required intermediate structure within the polygon.

Regular Surfaces

A surface constructed by a regular mesh, each cell is constructed by exactly four vertices. The neighborhood information is known intrinsically (details given in A.1.2) on the vertices and on the cells.



The only data which need to be given explicitly are the point coordinates. The neighborhood information arrays and the points per cell are known from the dimensional information of the surface.

Images and Movies

A special case of quad-based surfaces are flat surfaces which are uniform in some chart. In this case the positional component in this chart can be procedural as well and the entire data structure does not contain any data sets at all. However, additional data fields can be added and provide the canonical layout to store images. Free parameters of the image storage are the grid identifier, the field identifier and the coordinate system under which it is stored. A sequence of consecutive images (animation/movie) is naturally defined via the slice group. In this context a slice has the meaning of an animation frame. The coordinate system selects among various representations of the same object; conceptually, all data sets within a representation group have the same number of elements, ie. all images must have the same number of pixels. It is appropriate to store viewpoint information in the corresponding chart object, so the "representations of an image" refer to eg. the left and right bitmap of a stereo image, or to various views of the same object from different locations. Multiple `Topology` object the ideal location to store different resolutions of the same image (e.g. mipmaps), similar to refinement levels of an AMR grid. Note that image data are associated with cells, not vertices of a grid. The field identifier is useful to select different channels of an image, eg. RGB, CMYK, alpha channel etc. The `Grid` object identifier is left free to allow arbitrary user-definable names to identify the image source and context. The following table summarizes the mapping between fiber bundle data model groups and image properties:

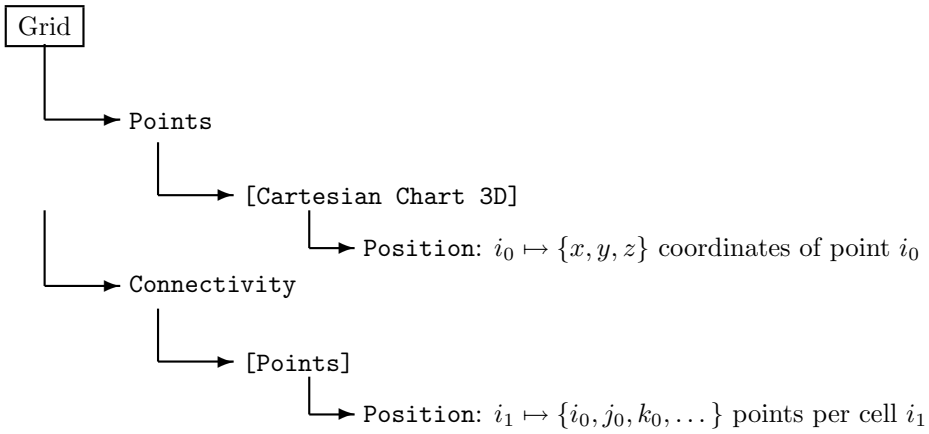
Slice	$\longleftrightarrow$	frames of an animation
Grid	$\longleftrightarrow$	scenery description
Topology	$\longleftrightarrow$	different resolutions
Representation	$\longleftrightarrow$	different viewpoints
Fields	$\longleftrightarrow$	color channels, transparency, ...

By storing images and movies in this data model, the following features are available in a consistent and natural way:

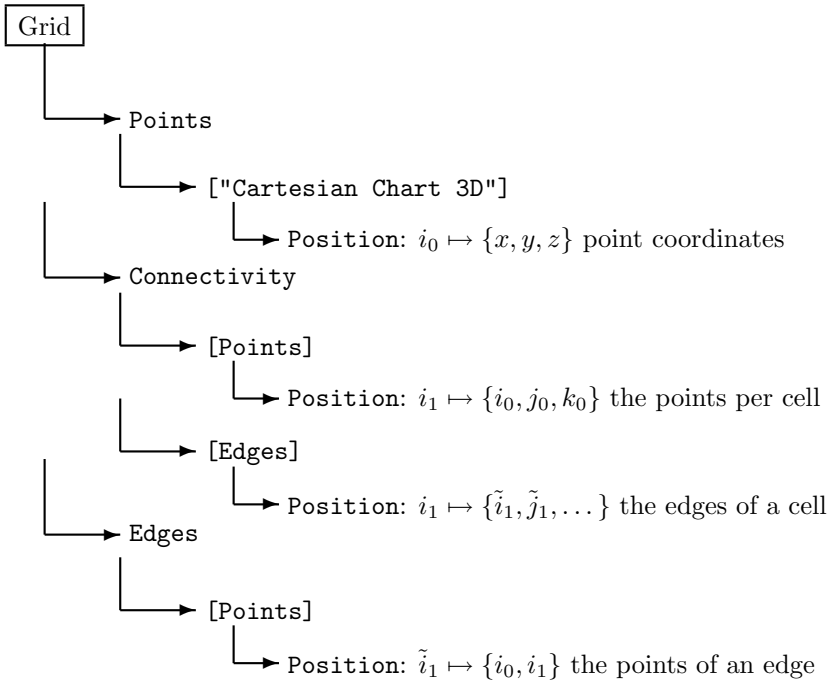
- multiple images per file (via multiple `Grid` identifiers)
- multi-channel images (via multiple `Field` identifiers)
- image sequences, e.g. animations (via multiple `Slice` instances)
- multi-viewpoint images, e.g. stereo images, tilings,... (via multiple `Representation`s)
- multi-resolution images (via refinement `Topology` objects)

Irregular Surfaces

An irregular surface is constructed by polygons, which have a varying number of points for each cell. The only difference to a triangular surface is that the positional element of the connectivity is no longer of fixed size:

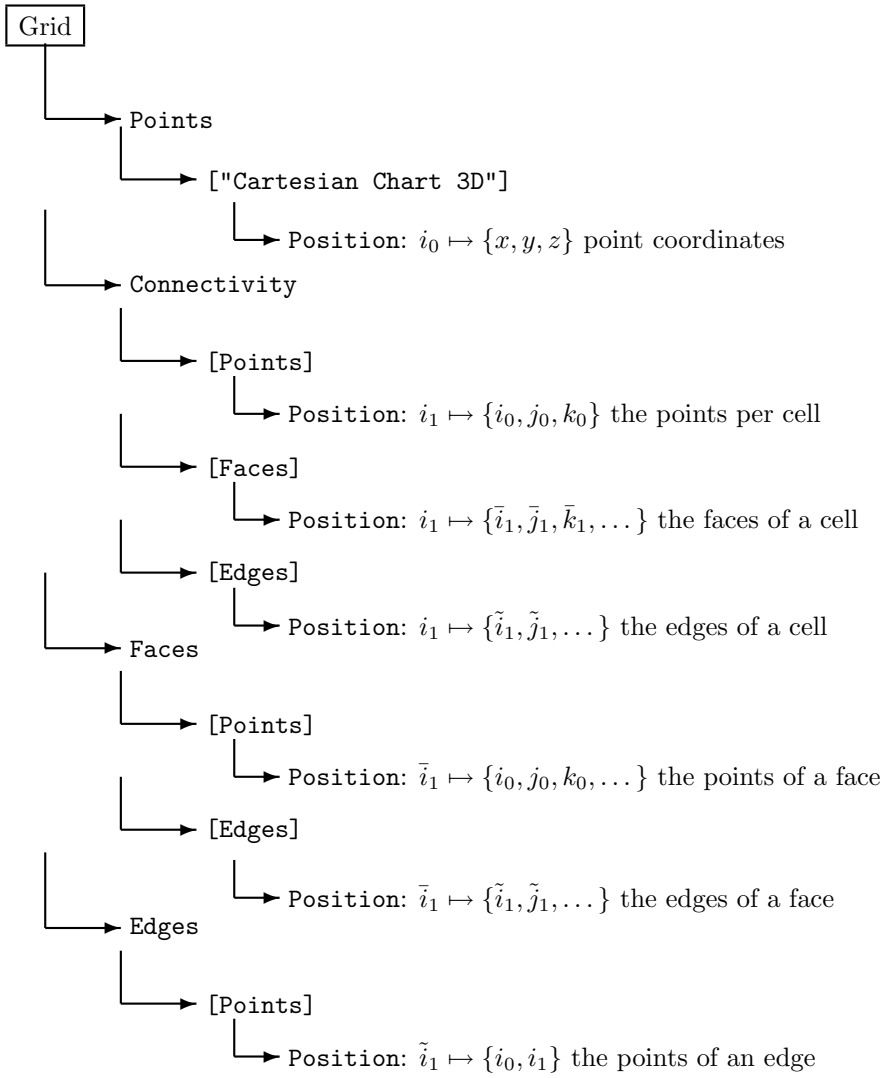


Each cell is constructed by a set of edges, whereby an edge is a set of two points. An additional Topology object can be added to the grid to store this edge-wise information:



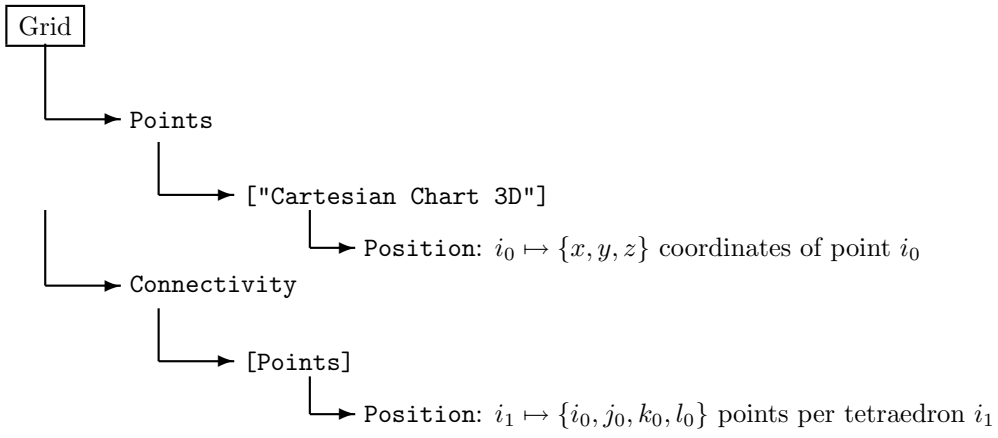
Unstructured Cell Data

Unstructured Cell Data (UCD) are like irregular surfaces, but each cell also consists of a number of faces.



Tetrahedral Grids

Tetrahedral grids are simpler than UCD, because the cell size of the connectivity is fixed. Formally, the same topologies exist on a grid as with the UCD grid, but the storage mechanism for the positional field of the connectivity in the points representation is simpler. By omitting edge and face related information, the description of a tetrahedral grid is very similar to a triangular surface:



Regular Grids

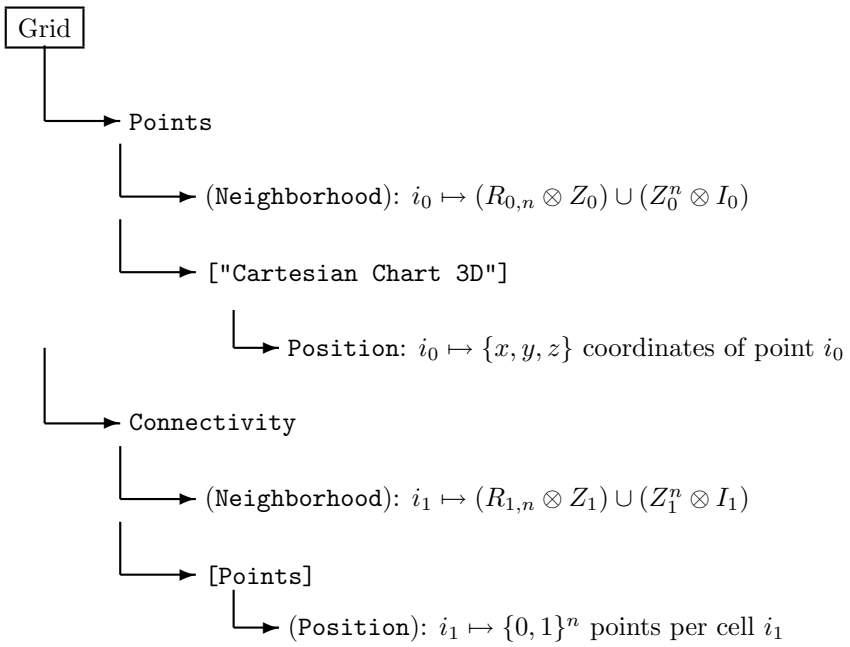
Regular grids are the higher-dimensional case of regular surfaces A.1.2. The neighborhood information on points and cells as well as the points per cell information are given intrinsically here by n numbers, which are called the size of the grid in each dimension. n is the dimension of the grid.

A regular neighborhood of dimension n can be defined recursively by the tensor product of one-dimensional neighborhoods. In one dimension, the ith point p_i has neighbors p_{i-1} and p_{i+1}. Without loss of generality, let us set i to 0 and inspect the indices of the neighborhood points. The point itself is the image of the set $Z := \{0\}$ and its neighbors are the images of the set $I := \{-1, +1\}$. The neighborhood R_n of a point in n dimensions is described by the iterative formula:

$$\begin{aligned} R_{n+1} &:= (R_n \otimes Z) \cup (Z^n \otimes I) \\ R_1 &:= I \end{aligned}$$

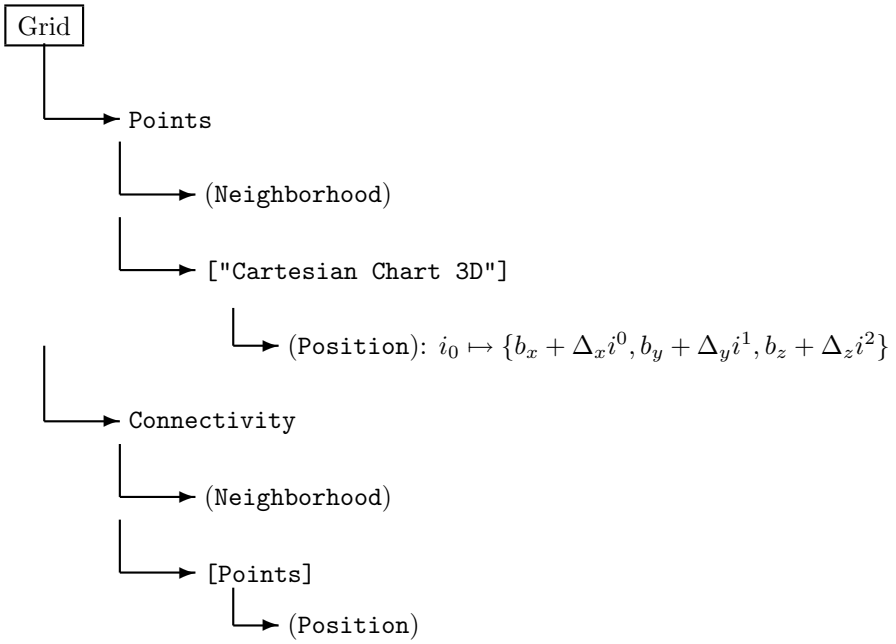
The (relative) regular neighborhoods therefore are:
$n = 1$ $\{-1, +1\}$
$n = 2$ $(\{-1, +1\} \times \{0\}) \cup (\{0\} \times \{-1, +1\}) \equiv \{(-1, 0), (+1, 0), (0, -1), (0, +1)\}$
$n = 3$ $\{(-1, 0, 0), (+1, 0, 0), (0, -1, 0), (0, +1, 0), (0, 0, -1), (0, 0, +1)\}$
...
The regular connectivity is given by the nth tensor product of of the set $\{0, 1\}^n$. Eg. in two dimensions the (relative) connectivity of a cell is the set of points with indices $\{0, 1\} \otimes \{0, 1\} \equiv \{(0, 0), (0, 1), (1, 0), (1, 1)\}$. In the context of the fiber bundle data model, these n-tuples of indices are only used when neighborhood or connectivity information is explicitly required by some algorithm. Otherwise, only the one-dimensional indexing of point numbers is used.

A symbolic layout of a regular grid using procedural tensor product expressions for the neighborhood and connectivity informations looks like this:

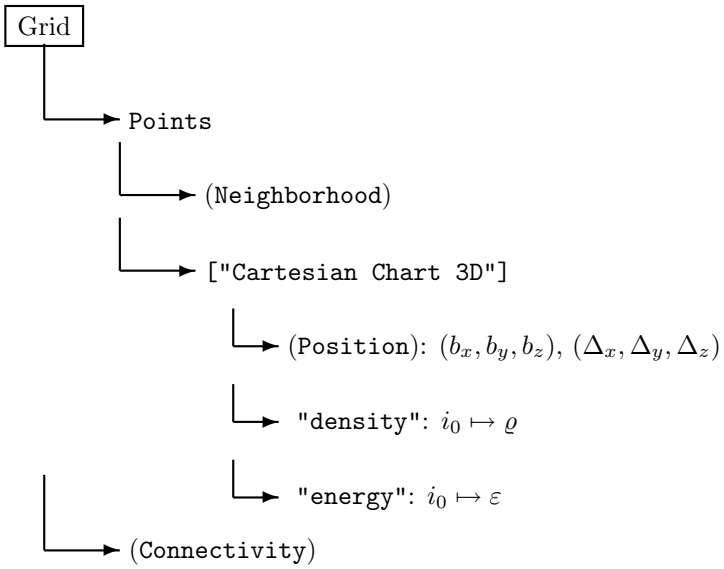


Uniform Grids

Uniform grids are special cases of regular grids, where the positional component is a first-order polynomial expression of the multidimensional index (i^0, i^1, i^2) in some chart, i.e. a linear map of the form $f(x) = kx + d$ separately for each coordinate:



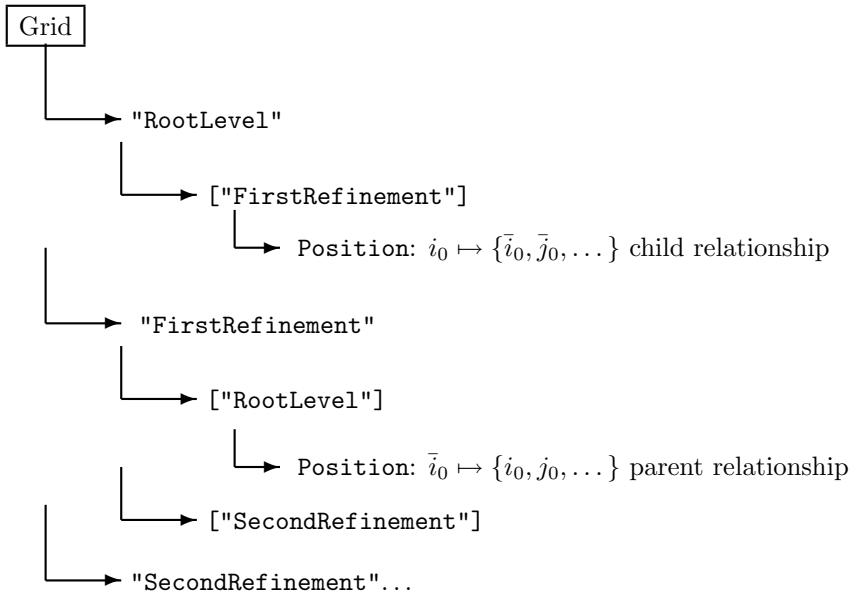
The only data to be stored are the coordinates of the base point (b_x, b_y, b_z) and their coordinate spacing $(\Delta_x, \Delta_y, \Delta_z)$. Data on a uniform grid are stored in the appropriate representation groups, eg.:



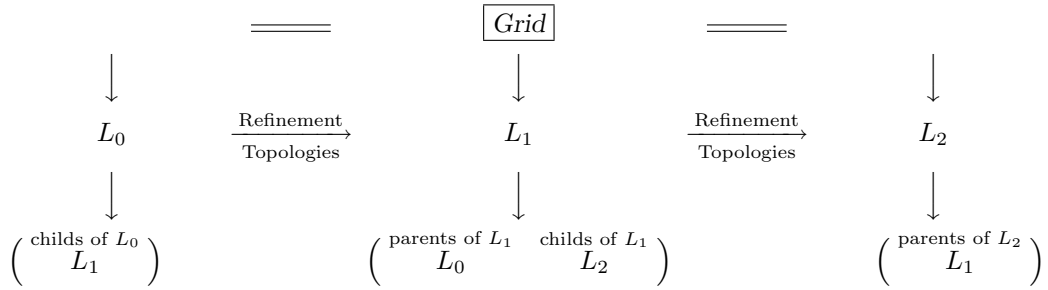
Structural information which is common for all fields is stored only once. Additional other charts and data representations can be added to a uniform grid, but are not necessarily uniform. While the regularity property is sustained under coordinate transformations, the uniformity property is not.

Hierarchical Grids

Hierarchical relationships among elements of a grid can be formulated via additional Topology groups, each of them describing a certain level of the hierarchy. The relative representation of the Topology groups is the right place to store relationships among these levels:



Another view is given by the succeeding diagram (A.1), which displays a three-level hierarchical grid with its Topology and Representations objects. The "Positions" field in the four Representation objects provide sets of indices for each level's element to its parents or childs (as future/past cones, see fig. 3.5). These are integer coordinates of each element in another discretized topological space, like in fig. 3.6. An implementation might store them explicitly (e.g. for refined unstructured grids) or extract the relevant information procedurally from a set of bounding boxes.



(A.1)

In addition to the relative level representations, each Topology level may also carry a representation of data and physical coordinates for a certain chart (not shown in (A.1)).
Topology groups may be of arbitrary index space. A Topology of index space 0 refers to vertices in the hierarchy, a Topology of index space 1 refers to cells. They have the same meaning as the `Point` and `Connectivity` Topologies and can be used in place of them to allow operations on a specific level only.

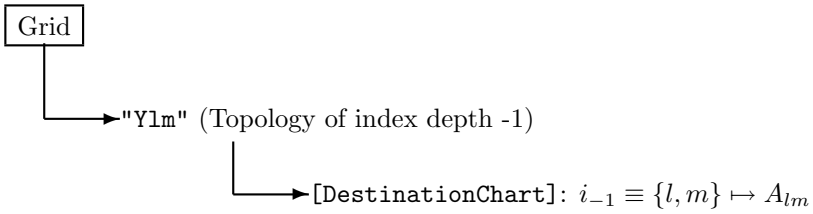
The various Topology objects may exist on different times, cross-references among Topology objects from distinct Slices (but not different Bundles) are possible. A typical case is when e.g. data on the coarsest level is only defined for $T = 4.0$ and $T = 6.0$, while the first refinement level is available for $T = 4.0, 5.0$ and 6.0.

A.1.3 Non-spatial data

Spherical Harmonic Coefficients

Within the framework of numerical relativity apparent horizons play a key role (see e.g. Max-Planck-News [Ben99a, Ben99b]). They are a primary indicator of the location and extend of black holes. An algorithm implemented by M. Alcubierre computes the apparent horizon by minimizing the expansion coefficients of spherical harmonics. The resulting surfaces are then described by non-spatial data but instead as a series of spherical harmonics corresponding to the multipole moments. The apparent horizon is an evolving ray surface with time-dependent coefficients, which constitute the radial distances according to (3.11). In general, the A_{lm} are complex coefficients, but in practice just a subset is required to describe the apparent horizon surface due to symmetry relationships.

Using the concept of a Coefficient Topology (3.2.3), the only data which needs to be stored e.g. in a file is the Coefficient Grid with the respective $A_{lm}(t)$ data and a coordinate transformation object that tells that the resulting field shall be the radial coordinate of the surface in a three-dimensional polar chart. This information is sufficient to compute the surface of an apparent horizon with arbitrary user-defined precision.



Multispectral Expansions

Embedding Diagrams as they are computed in a new method developed at the AEI by Bondarescu, Alcubierre and Seidel [BAS02] are an extension of the apparent horizon finding algorithm. Similar to finding the apparent horizon by minimizing the coefficients of a spherical harmonics expansion of the surface, a set of coefficients can also be minimized for a flat space surface, such that the metric on top of this 2D-surface in 3D flat space yields the same metric as the 2D surface of the apparent horizon in 3D curved space. However, just minimizing for the radial coefficients is not sufficient. In addition, one needs also to introduce new coordinates $\{\bar{\vartheta}, \bar{\varphi}\}$ within the embedded surface. These new coordinates can then be related to the original $\{\tilde{\vartheta}, \tilde{\varphi}\}$ surface coordinates along the surface via a multispectral expansion. The coefficients for a certain coordinate are denoted by the same letter, grouped by the family of basis functions:

$$\bar{\vartheta}(\tilde{\vartheta}, \tilde{\varphi}) = B^{(0)}\tilde{\vartheta} + B^{(1)}\tilde{\varphi} + \sum_i B_i^{(2)} \sin(i\tilde{\vartheta}) + \sum_i B_i^{(3)} \sin(i\tilde{\varphi}) + \sum_i B_{i,j}^{(4)} \sin(i\tilde{\varphi}) \sin(j\tilde{\vartheta})$$

$$\bar{\varphi}(\tilde{\vartheta}, \tilde{\varphi}) = C^{(0)}\tilde{\vartheta} + C^{(1)}\tilde{\varphi} + \sum_i C_i^{(2)} \sin(i\tilde{\vartheta}) + \sum_i C_i^{(3)} \sin(i\tilde{\varphi}) + \sum_i C_{i,j}^{(4)} \sin(i\tilde{\varphi}) \sin(j\tilde{\vartheta})$$

The embedding algorithm then also minimizes for the B and C coefficients. The radial component is to be computed in the flatspace embedding coordinates $\{\bar{\vartheta}, \bar{\varphi}\}$:

$$\bar{R}(\bar{\vartheta}, \bar{\varphi}) = \sum_{l,m} A_{lm} Y_{lm} \left(\tilde{\vartheta}(\bar{\vartheta}, \bar{\varphi}), \tilde{\varphi}(\bar{\vartheta}, \bar{\varphi}) \right)$$

All these coefficients, A, B, C are time-dependent; this dependency has been dropped in the above expressions for clarity, but is of importance because the grid operations of coefficient evaluation and grid interpolation are no longer commutative (in contrast to the simpler apparent horizon surfaces described in the previous section A.1.3). Thus it is essential to make use of the concept of local and global charts (ref. 3.2.3).

To formulate these data in the context of the fiber bundle data model, they can be put into coefficient topologies, each of them corresponding to a specific (procedurally defined) set of basis functions. Hereby $C_{i,j}^{(4)}$ and $B_{i,j}^{(4)}$ are two-dimensional coefficients, while the others are one-dimensional coefficients or just a constant. Also higher dimensional sets of coefficients are thinkable in general, and it is obvious that the formulation via a topological space makes sense for coefficients as well.

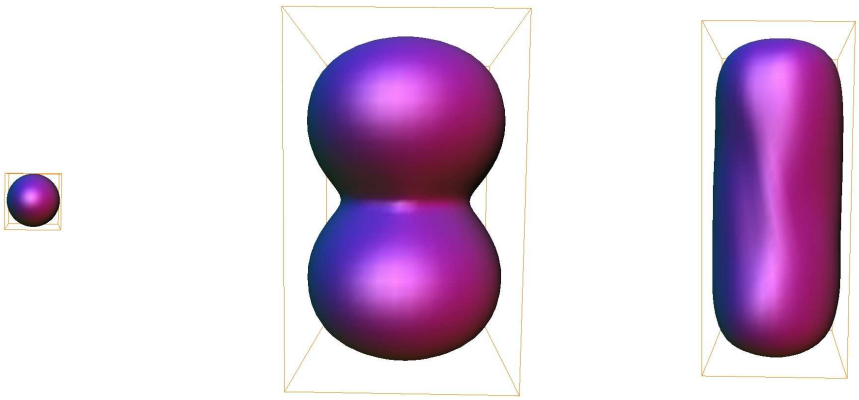


Figure A.1: The apparent horizon of a numerical black hole simulation, shown at the same scale in coordinate space (coordinates chosen for the numerical computation), surface coordinates (showing just the radial expansion) and flat space (including also a coordinate transformation on the surface itself before the radial expansion is shown).

Technically, the coefficient evaluation operation will just compute separate fields which are the sum of the coordinate expressions. It is left to the functionality of a transformation object to combine these separate fields to new coordinates. Fig. A.1 shows an example of an embedding surface in three different coordinate systems according to the included transformation rules.

A.2 Summation of Spherical Harmonics

The summation of spherical harmonic functions from a set of coefficients

$$A = \sum_{k} \sum_{l=-k}^{k} A_{lk} Y_{lk}(\vartheta, \varphi) \quad , \tag{A.2}$$

is a frequent task. E.g., it was used to compute the surface of an apparent horizon as determined by an expansion in spherical harmonics. Often, it it is done in the most straightforward way by directly casting the mathematical expression into a numerical algorithm. This is a bad choice, once for performance reasons and, more critical, for numerical stability. An algorithm by Deuflhard presented in [Deu76] and [DH03] tackles both issues and provides a fast and numerically stable recipe for the summation of coefficients of a spherical harmonic expansion series. In the original publication, the algorithm was shown only for a limited range of ϑ, φ, i.e. the first quadrant. Here we show its extension to the full range on a sphere and review its derivation. The two publications differ by their normalizations of the underlying Legendre polynomials. [Deu76] uses

$$P_k^l(x) = \frac{(-1)^l}{2^k k!} \sqrt{1-x^2} \, \frac{d^{k+l}}{dx^{k+l}} \, (x^2-1)^k \quad ,$$

whereas [DH03] implements the other convention found in literature as well,

$$\hat{P}_k^l(x) = \frac{(-1)^l}{(k+l)! \; 2^k k!} \sqrt{1-x^2} \, \frac{d^{k+l}}{dx^{k+l}} \, (x^2-1)^k \quad \equiv \frac{1}{(k+l)!} P_k^l(x) \quad .$$

Here, we use the first normalization convention.

A critical source of numerical instabilities is the computation of differences. While addition is numerically harmless, subtractions may be troublesome. For instance, the expression $(10^{-10}+1)-1$ will yield exactly zero when the computation is performed with a precision of nine digits, because $10^{-10}+1$ will be rounded to 1. However, reordering of the expression as $10^{-10}+(1-1)$ will exactly yield 10^{-10}, even when computed with nine digits of precision. Alternating sums including a factor -1^l like those of the Legendre polynomials are very sensitive to such errors, but may be cured by reordering of the summation algorithm. Similar cases like the term $\cos\vartheta - 1$ lead to troubles as well and become numerically instable for small ϑ where $\cos\vartheta \to 1$. By the means of trigonometric relationships we may substitute the cosine by a more suitable expression, as will be demonstrated below.

Here, the spherical harmonic functions are defined as

$$Y_k^l(\vartheta,\varphi) := \quad P_k^l(\cos\vartheta) \quad e^{il\varphi} \tag{A.3}$$

whereby the associated Legendre polynomials $P_{l,k}$ obey the recurrence relationship

$$(l-k)P_k^l(\mu) = \mu(2l-1)P_{k-1}^l - (l+k-1)P_{k-2}^l \tag{A.4}$$

with $\mu = \cos\vartheta$. The algorithm described in [Deu76] computes the real and imaginary part of a summation $A = C + iS$ over real-valued coefficients A_{lm}. For complex coefficients the algorithm can be called again with another parameter set for the imaginary coefficients. At first, the sum (A.2) can be written by reordering the summations and performing the sum over k as the inner sum:

$$C = \sum_{l=0}^{L}\sum_{k=l}^{K} A_{lk}\ \Re Y_{lk}(\vartheta,\varphi) \tag{A.5}$$

$$S = \sum_{l=1}^{L}\sum_{k=l}^{K} A_{lk}\ \Im Y_{lk}(\vartheta,\varphi) \tag{A.6}$$

The resulting algorithm is numerically stable and much faster than a straightforward implementation of (A.2), but is still simple, such that an implementation of the algorithm fits into a single page, as will be demonstrated in A.2. In this section the derivation of the algorithm is continued at the place where the original paper handled the first quadrant only. For a complete introduction, the reader is referred to [Deu76].

Summation basics

When computing the sum

$$S_N(x) = \sum_{k=0}^{N} A_k T_k(x)$$

of special functions $T_k(x)$ given by a difference equations of the form

$$T_k - a_k(x)T_{k-1} - b_k(x)T_{k-2} = 0 \qquad b_k \neq 0 \quad k = 2,\ldots,N \quad ,$$

we can compute the sum forward (known as Forsythe's algorithm), starting from $k = 2$ increasing to N:

$$\begin{array}{lcl} S_1 & = & A_0T_0 + A_1T_1 \\ k & = & 2,3,\ldots,N \\ & & \quad T_k = a_kT_{k-1} + b_kT_{k-2} \\ & & \quad S_k = S_{k-1} + A_kT_k \end{array}$$

Alternatively we may employ Clenshaw's algorithm to perform the sum backward, starting with $k = N$ decreasing to $k = 1$, introducing auxiliary variables U:

$$\begin{array}{lcl} U_{N+1} & = & U_{N+2} = 0 \\ k & = & N, N-1,\ldots,1 \\ & & \quad U_k = a_{k+1}U_{k+1} + b_{k+2}U_{k+2} + A_k \\ S_N & = & (A_0 + b_2U_2)T_0 + U_1T_1 \end{array}$$

The mathematical equivalence of both algorithms is e.g. shown in [Deu76]. Numerically, the second version is faster because there obviously are less operations in the k-loop, saving N multiplications all together. Moreover, numerical stability of the summation process is related to the stability of the related recurrence scheme. For instance, for the Bessel functions $J_k(x)$ the recurrence scheme $a_k(x) = 2(k-1)/x$, $b_k(x) = -1$ is known to be stable backward but unstable forward, so Clenshaw's algorithm is suitable and numerically preferable here.

Summation of spherical harmonics

Introducing auxiliary variables U, V, one can derive the following recursion scheme for the spherical harmonics [Deu76]:

$$
\begin{array}{lcl}
V_{L+1} & = & 0 \\
V_{L+2} & = & 0 \\
l & = & L, L-1, \ldots, 0 \\
\\
\quad U^l_{K+1} & = & 0 \\
\quad U^l_{K+2} & = & 0 \\
\quad k & = & K, K-1, \ldots, l+1 \\
\quad\quad U^l_k & = & A^l_k + \frac{2k+1}{k-l+1} U^l_{k+1} \cos\vartheta - \frac{k+l+1}{k-l+2} U^l_{k+1+2} \\
\\
\quad \text{if} & (l>0) & \\
\quad\quad V_l & = & U^l_l - (2l+1)\sin\vartheta\,(2\cos\varphi V_{l+1} + (2l+3)\sin\vartheta V_{l+2}) \\
\\
C & = & U^0_0 - \sin\vartheta\,[\cos\varphi V_1 + 3\sin\vartheta V_2] \\
S & = & -\sin\vartheta\sin\varphi V_1
\end{array}
$$

This algorithm is unstable for $\varphi \to 0$ or $\vartheta \to 0$.

Stabilization of the inner sum (loop over k)

The summation variables of the k loop are denoted by U. The summation starts with $k = K$, thus

$$U_{K+1} = U_{K+2} = 0 \quad . \tag{A.7}$$

The recursion formula for $k = K, K-1, \ldots, l$ is known as

$$U^l_k = A^l_k + \frac{2k+1}{k-l+1} U^l_{k+1} \cos\vartheta - \frac{k+l+1}{k-l+2} U^l_{k+2} \quad . \tag{A.8}$$

We make an ansatz to stabilize the recursion of U:

$$
\begin{aligned}
U^l_k &= q^l_k \bar{U}^l_k && \text{(A.9)} \\
\bar{U}^l_k &= r^l_k \bar{U}^l_{k+1} + dU^l_k && \text{(A.10)}
\end{aligned}
$$

Here, $\bar{U}^l_k$ and dU^l_k shall be used as new computational variables in the loop. The coefficients q^l_k and r^l_k have been introducted to gain some freedom in scaling the loop variables. We now show how to determine these coefficients. Reformulation of this ansatz yields

$$
\begin{aligned}
U^l_k &= q^l_k \left(r^l_k \bar{U}^l_{k+1} + dU^l_k \right) && \text{(A.11)} \\
\bar{U}^l_{k+1} &= \frac{1}{r^l_k} \left(\bar{U}^l_k - dU^l_k \right) && \text{(A.12)}
\end{aligned}
$$

We now insert (A.11) and (A.9) into the recursion formula (A.8):

$$\underbrace{q^l_k \left(r^l_k \bar{U}^l_{k+1} + dU^l_k \right)}_{U^l_k} = A^l_k + \frac{2k+1}{k-l+1} \underbrace{q^l_{k+1} \bar{U}^l_{k+1}}_{U^l_{k+1}} \cos\vartheta - \frac{k+l+1}{k-l+2} \underbrace{q^l_{k+2} \bar{U}^l_{k+2}}_{U^l_{k+2}} \tag{A.13}$$

By means of (A.12) we may express $\bar{U}^l_{k+2}$ by terms of $k+1$:

$$\bar{U}^l_{k+2} = \frac{1}{r^l_{k+1}} \left(\bar{U}^l_{k+1} - dU^l_{k+1} \right) \quad . \tag{A.14}$$

Inserting (A.14) into (A.13) results in

$$
\begin{aligned}
& q^l_k \left(r^l_k \bar{U}^l_{k+1} + dU^l_k \right) = \\
& \qquad A^l_k + \frac{2k+1}{k-l+1} q^l_{k+1} \bar{U}^l_{k+1} \cos\vartheta - \frac{k+l+1}{k-l+2} q^l_{k+2} \underbrace{\frac{1}{r^l_{k+1}} \left(\bar{U}^l_{k+1} - dU^l_{k+1} \right)}_{\bar{U}^l_{k+2}}, \qquad \text{(A.15)}
\end{aligned}
$$

thus revealing an expression of dU_k^l as a function of $k+1$ terms:

$$\begin{aligned} q_k^l dU_k^l &= A_k^l + \frac{2k+1}{k-l+1} q_{k+1}^l \bar{U}_{k+1}^l \cos\vartheta - q_k^l r_k^l \bar{U}_{k+1}^l - \frac{k+l+1}{k-l+2} \frac{q_{k+2}^l}{r_{k+1}^l} \left(\bar{U}_{k+1}^l - dU_{k+1}^l\right) \\ &= A_k^l + \frac{k+l+1}{k-l+2} \frac{q_{k+2}^l}{r_{k+1}^l} dU_{k+1}^l + \bar{U}_{k+1}^l \underbrace{\left[\frac{2k+1}{k-l+1} q_{k+1}^l \cos\vartheta - q_k^l r_k^l - \frac{k+l+1}{k-l+2} \frac{q_{k+2}^l}{r_{k+1}^l}\right]}_{=:F(\cos\vartheta)} \end{aligned} \tag{A.16}$$

Since the dependency of $\cos\vartheta$ on ϑ is low for small angles $\vartheta \to 0$, the term $F(\cos\vartheta)$ shall not contribute to the recursion at $\vartheta = 0$ (stability condition), i.e.:

$$F(\cos\vartheta)|_{\vartheta=0} = 0 \tag{A.17}$$

Therefore we require:

$$F(\cos(\vartheta))|_{\vartheta=0} = \frac{2k+1}{k-l+1} q_{k+1}^l \cdot \underbrace{1}_{\cos 0} - q_k^l r_k^l - \frac{k+l+1}{k-l+2} \frac{q_{k+2}^l}{r_{k+1}^l} \equiv 0 \tag{A.18}$$

This condition can be achieved by setting

$$q_k^l := (k-l) \tag{A.19}$$
$$r_k^l := 1 \tag{A.20}$$

Analogously, for $\cos\vartheta < 0$ we need to stabilize the case $\vartheta \to \pi$:

$$F(\cos(\vartheta))|_{\vartheta=\pi} = \frac{2k+1}{k-l+1} q_{k+1}^l \cdot \underbrace{-1}_{\cos\pi} - q_k^l r_k^l - \frac{k+l+1}{k-l+2} \frac{q_{k+2}^l}{r_{k+1}^l} \equiv 0 \tag{A.21}$$

which can be achieved by

$$q_k^l := (k-l) \tag{A.22}$$
$$r_k^l := -1 \tag{A.23}$$

Thus we set

$$r_k^l \equiv r_\vartheta := \begin{cases} \cos\vartheta \geq 0 & +1 \\ \cos\vartheta < 0 & -1 \end{cases} \tag{A.24}$$

and get for F:

$$\begin{aligned} F(\cos(\vartheta)) =& \frac{2k+1}{k-l+1} \underbrace{(k+1-l)}_{q_{k+1}^l} \cos\vartheta - \underbrace{(k-l)}_{q_k^l} r_\vartheta - \frac{k+l+1}{k-l+2} \frac{\overbrace{k+2-l}^{q_{k+2}^l}}{r_\vartheta} \\ =& (2k+1)\cos\vartheta - (k-l) r_\vartheta - \frac{k+l+1}{r_\vartheta} \end{aligned} \tag{A.25}$$

By recognizing that $r_\vartheta \equiv \frac{1}{r_\vartheta}$ due to A.24 we may write A.25 as

$$F(\cos(\vartheta)) = (2k+1)\cos\vartheta - \underbrace{(k-l+k+l+1)}_{2k+1} r_\vartheta = (2k+1)(\cos\vartheta - r_\vartheta) \tag{A.26}$$

As the expression $\cos\vartheta - 1$ is numerically instable for $\cos\vartheta \to 1$, we employ trigonometric relationships and use the replacement expression $-2\sin^2\frac{\vartheta}{2}$, which is numerically well behaved; analogously at $\cos\vartheta \leq 0$. We define:

$$c_\vartheta := \cos\vartheta - r_\vartheta = \begin{cases} \cos\vartheta \geq 0 & \cos\vartheta - 1 \equiv -2\sin^2\frac{\vartheta}{2} \\ \cos\vartheta < 0 & \cos\vartheta + 1 \equiv 2\cos^2\frac{\vartheta}{2} \end{cases} \tag{A.27}$$

such that

$$F(\cos(\vartheta)) = (2k+1) c_\vartheta \tag{A.28}$$

is a numerically stable expression for all ϑ. Now we revisit the recursion formula (A.16) by inserting the results (A.19) and (A.22) as well as (A.28) and get:

$$(k-l)dU_k^l = A_k^l + \frac{k+l+1}{k-l+2}\frac{k+2-l}{r_\vartheta}dU_{k+1}^l + \bar{U}_{k+1}^l(2k+1)c_\vartheta \tag{A.29}$$

Thus the recursion takes the form:

$$(k-l)dU_k^l = A_k^l + (k+l+1)r_\vartheta dU_{k+1}^l + \bar{U}_{k+1}^l(2k+1)c_\vartheta \tag{A.30}$$

together with the other recursion term (A.10):

$$\bar{U}_k^l = r_\vartheta \bar{U}_{k+1}^l + dU_k^l \tag{A.31}$$

Terminating the Recursion over k

The case $k = l$ requires special treatment when looping over k. According to (A.8) we get for U_l^l:

$$U_l^l = A_l^l + (2l+1)U_{l+1}^l\cos\vartheta - \frac{2l+1}{2}U_{l+2}^l \tag{A.32}$$

where

$$U_{l+2}^l \overset{(A.9)}{=} \underbrace{q_{l+2}^l}_{(l+2)-l=2}\bar{U}_{l+2}^l \overset{(A.12)}{=} 2\frac{1}{r_\vartheta}\left(\bar{U}_{l+1}^l - dU_{l+1}^l\right) \tag{A.33}$$

$$U_{l+1}^l \overset{(A.9)}{=} \underbrace{q_{l+1}^l}_{1}\bar{U}_{l+1}^l \tag{A.34}$$

and therefore we may directly express U_l^l as

$$\begin{aligned} U_l^l &= A_l^l + (2l+1)\bar{U}_{l+1}^l\cos\vartheta - \frac{2l+1}{2}2\frac{1}{r_\vartheta}\left(\bar{U}_{l+1}^l - dU_{l+1}^l\right) \\ &= A_l^l + (2l+1)\bar{U}_{l+1}^l\underbrace{(\cos\vartheta - r_\vartheta)}_{c_\vartheta} + r_\vartheta(2l+1)dU_{l+1}^l \end{aligned} \tag{A.35}$$

Stabilization of the Summation over l

We start with considering eq. (3.10) in [Deu76]:

$$V_{L+1} = V_{L+2} = 0 \tag{A.36}$$

$$V_l = U_l^l - (2l+1)\sin\vartheta\left[2V_{l+1}\cos\varphi + (2l+3)V_{l+2}\sin\vartheta\right] \tag{A.37}$$

We make an ansatz for new loop variables $\bar{V}$ and dV:

$$V_l = q_l\bar{V}_l \tag{A.38}$$

$$\bar{V}_l = r_l\bar{V}_{l+1} + dV_l \tag{A.39}$$

and get

$$V_l = q_l\underbrace{\left(r_l\bar{V}_{l+1} + dV_l\right)}_{\bar{V}_l} \tag{A.40}$$

$$V_{l+1} = q_{l+1}\bar{V}_{l+1} \tag{A.41}$$

$$V_{l+2} = q_{l+2}\underbrace{\frac{1}{r_{l+1}}\left(\bar{V}_{l+1} - dV_{l+1}\right)}_{\bar{V}_{l+2}} \tag{A.42}$$

Insertion into (A.37) yields

$$q_l\left(r_l\bar{V}_{l+1} + dV_l\right) = U_l^l - (2l+1)\sin\vartheta\left(2q_{l+1}\bar{V}_{l+1}\cos\varphi + (2l+3)\frac{q_{l+2}}{r_{l+1}}\left(\bar{V}_{l+1} - dV_{l+1}\right)\sin\vartheta\right) \tag{A.43}$$

We may now express dV_l in terms of $l+1$:

$$\begin{aligned} q_l dV_l =& U_l^l + (2l+1)(2l+3)\frac{q_{l+2}}{r_{l+1}} dV_{l+1} \sin^2\vartheta - \\ & - \bar{V}_{l+1} \underbrace{\left[q_l r_l + 2(2l+1) q_{l+1} \sin\vartheta\cos\varphi + (2l+3)(2l+1)\frac{q_{l+2}}{r_{l+1}} \sin^2\vartheta \right]}_{=:G(\varphi)} \end{aligned} \tag{A.44}$$

We want $G(\varphi)$ to vanish when the dependency of $\cos\varphi$ on φ is small. The stability condition thus reads:

$$\cos\varphi \geq 0: \quad \varphi \to 0, \cos\varphi \to \quad 1 \quad G(\varphi) \to 0 \tag{A.45}$$

$$\cos\varphi < 0: \quad \varphi \to \pi, \cos\varphi \to -1 \quad G(\varphi) \to 0 \tag{A.46}$$

Condition (A.45), i.e. $G(0) = 0$, is achieved by

$$r_l(\cos\varphi \geq 0) = -(2l+1)\sin\vartheta \tag{A.47}$$

$$q_l = 1 \tag{A.48}$$

and condition (A.46), i.e. $G(\pi) = 0$, is achieved by

$$r_l(\cos\varphi < 0) = +(2l+1)\sin\vartheta \tag{A.49}$$

$$q_l = 1 \tag{A.50}$$

We may thus set

$$r_l = -r_\varphi (2l+1)\sin\vartheta \tag{A.51}$$

with

$$r_\varphi := \begin{cases} \cos\varphi \geq 0 & +1 \\ \cos\varphi < 0 & -1 \end{cases} \tag{A.52}$$

and get, considering numerical instabilities of the trigonometric functions as in (A.27),

$$G(\varphi) = (2l+1)2\sin\vartheta \qquad \underbrace{(\cos\varphi - r_\varphi)}_{c_\varphi := \begin{cases} \cos\varphi \geq 0: & -2\sin^2\frac{\varphi}{2} \\ \cos\varphi < 0: & +2\cos^2\frac{\varphi}{2} \end{cases}} . \tag{A.53}$$

Inserting this result into (A.44) yields

$$\begin{aligned} dV_l =& U_l^l + \underbrace{\frac{(2l+1)(2l+3)}{-r_\varphi(2l+3)\sin\vartheta}}_{-(2l+1)\sin\vartheta r_\varphi} \sin^2\vartheta\, dV_{l+1} - \bar{V}_{l+1} 2(2l+1)2\sin\vartheta c_\varphi \\ =& U_l^l - (2l+1)\sin\vartheta \left(r_\varphi dV_{l+1} + 2\bar{V}_{l+1} c_\varphi \right) \end{aligned} \tag{A.54}$$

This expression together with (A.39) defines the recursion:

$$\bar{V}_l = -r_\varphi (2l+1)\bar{V}_{l+1}\sin\vartheta + dV_l \tag{A.55}$$

Terminating the Recursion over l

We know from [Deu76]:

$$C = U_0^0 - \sin\vartheta\,(V_1\cos\varphi + 3V_2\sin\vartheta) \tag{A.56}$$

$$S = -\sin\vartheta\sin\varphi V_1 \tag{A.57}$$

We may write V_2 in the computational variables:

$$V_2 = \bar{V}_2 \overset{(A.39)}{=} \frac{1}{r_1}\left(\bar{V}_1 - dV_1\right) \overset{(A.51)}{=} \frac{1}{-3r_\varphi\sin\vartheta}\left(\bar{V}_1 - dV_1\right) = -\frac{1}{3} r_\varphi \frac{1}{\sin\vartheta}\left(\bar{V}_1 - dV_1\right) \tag{A.58}$$

which lets us express C and S via $\bar{V}_1$ und dV_1. Neither $\bar{V}_0$ nor dV_0 need to be computed. We finally get:

$$C = U_0^0 - \sin\vartheta\left[V_1\cos\varphi - r_\varphi\left(\bar{V}_1 - dV_1\right)\right] = U_0^0 - \sin\vartheta\left[c_\varphi\bar{V}_1 + r_\varphi dV_1\right] \tag{A.59}$$

$$S = -\sin\vartheta\sin\varphi\bar{V}_1 \tag{A.60}$$

Normalization factors

The definition (A.3) used here for deriving the summation algorithm does not include the normalization factors as often found in physics

$$\tilde{Y}_k^l(\vartheta,\varphi) := \sqrt{\frac{2l+1}{4\pi}\,\frac{(l-k)!}{(l+k)!}}\quad P_k^l(\cos\vartheta)\quad e^{il\varphi} \equiv \sqrt{\frac{2l+1}{4\pi}\,\frac{(l-k)!}{(l+k)!}}\quad Y_k^l(\vartheta,\varphi)\quad . \tag{A.61}$$

To yield the same results, the normalization factors can be added to the coefficients or into the respective location of the algorithm where the coefficients are evaluated.

Algorithm

Given a set of real-valued coefficients A_k^l with $l \le L$, the real part C (A.5) and the imaginary part S (A.6) of the spherical harmonic expansion up to the multipole moment of order L are computed via the following algorithm:

$$\begin{aligned}
r_\vartheta &\overset{(A.24)}{:=} \begin{cases} \cos\vartheta \ge 0 & +1 \\ \cos\vartheta < 0 & -1 \end{cases} \\
c_\vartheta &\overset{(A.27)}{:=} \begin{cases} \cos\vartheta \ge 0 & -2\sin^2\frac{\vartheta}{2} \\ \cos\vartheta < 0 & +2\cos^2\frac{\vartheta}{2} \end{cases} \\
r_\varphi &\overset{(A.52)}{:=} \begin{cases} \cos\varphi \ge 0 & +1 \\ \cos\varphi < 0 & -1 \end{cases} \\
c_\varphi &\overset{(A.53)}{:=} \begin{cases} \cos\varphi \ge 0: & -2\sin^2\frac{\varphi}{2} \\ \cos\varphi < 0: & +2\cos^2\frac{\varphi}{2} \end{cases}
\end{aligned}$$

$$\begin{aligned}
dV_{L+1} &= 0 \\
\bar{V}_{L+1} &= 0 \\
l &= L, L-1, \ldots, 0
\end{aligned}$$

$$\begin{aligned}
dU^l_{K+1} &= 0 \\
\bar{U}^l_{K+1} &= 0 \\
k &= K, K-1, \ldots, l+1 \\
dU^l_k &\overset{(A.30)}{=} \tfrac{1}{k-l}\left[A^l_k + (k+l+1) r_\vartheta dU^l_{k+1} + (2k+1)\bar{U}^l_{k+1} c_\vartheta\right] \\
\bar{U}^l_k &\overset{(A.31)}{=} r_\vartheta \bar{U}^l_{k+1} + dU^l_k
\end{aligned}$$

$$U^l_l \overset{(A.35)}{=} A^l_l + (2l+1)\left(\bar{U}^l_{l+1} c_\vartheta + r_\vartheta dU^l_{l+1}\right)$$

$$\begin{aligned}
&\text{if} \quad (l > 0) \\
&\quad dV_l \overset{(A.54)}{=} U^l_l - (2l+1)\sin\vartheta\left(r_\varphi dV_{l+1} + 2\bar{V}_{l+1} c_\varphi\right) \\
&\quad \bar{V}_l \overset{(A.55)}{=} -r_\varphi (2l+1)\bar{V}_{l+1}\sin\vartheta + dV_l
\end{aligned}$$

$$\begin{aligned}
C &\overset{(A.59)}{=} U^0_0 - \sin\vartheta\left[c_\varphi \bar{V}_1 + r_\varphi dV_1\right] \\
S &\overset{(A.60)}{=} -\sin\vartheta\sin\varphi\bar{V}_1
\end{aligned}$$

For complex coefficients $A_k^l = \Re A_k^l + i\Im A_k^l$ the iteration variables `dV, V, dU, U` and their corresponding expressions need to be duplicated. Alternatively, the sums involving $\Re A_k^l$ and $\Im A_k^l$ can be computed separately and combined as

$$\sum_l \sum_k A_k^l Y_{lk} = C(\Re A_k^l) - S(\Im A_k^l) + i\big(C(\Im A_k^l) + S(\Re A_k^l)\big)$$

whereby the pairs $C(\Re A_k^l), S(\Re A_k^l)$ and $C(\Im A_k^l), S(\Im A_k^l)$ are computed directly by the described algorithm on real-valued data arrays.

Example Implementation in C++

Here, a function

```
inline double sqr(double x) { return x*x; }
```

computes the square of its argument. The spherical harmonic coefficients A_k^l are available from a real-valued object A via

$$A_k^l = \texttt{A(l,k)}$$

The angular input parameters ϑ, φ may point to an arbitrary location on the sphere, i.e. $\vartheta = [0, \pi]$ and $\varphi = [0, 2\pi]$.

```
double r_theta, c_theta, sin_theta = sin(theta);
       if (theta > M_PI_2) // cos(theta)<0
       {
              r_theta = -1;
              c_theta =  2*sqr(cos(0.5*theta));
       }
       else
       {
              r_theta = +1;
              c_theta = -2*sqr(sin(0.5*theta));
       }
double r_phi, c_phi;
       if (phi > M_PI_2 && phi < 1.5*M_PI_2) // cos(phi)<0
       {
              r_phi = -1;
              c_phi =  2*sqr(cos(0.5*phi));
       }
       else
       {
              r_phi = +1;
              c_phi = -2*sqr(sin(0.5*phi));
       }

double dV = 0, V = 0,
       dU, U;
       for(int l = L; l>=0; l--)
       {
              dU = 0;
              U  = 0;
              for(int k=K; k>l; k--)
              {
                     dU = ( A(l,k) + (k+l+1)*r_theta*dU +
                            (2*k+1)*U*c_theta ) / (k-l);
                      U = r_theta*U + dU;
              }
              U = A(l,l) + (2*l+1) * (U*c_theta + r_theta*dU);
              if (l>0)
              {
                     dV = U - (2*l+1)*sin_theta*(r_phi*dV + 2*V*c_phi);
                      V = -r_phi * (2*l+1)*V*sin_theta + dV;
              }
       }
double C = U - sin_theta*( c_phi * V  + r_phi * dV ),
       S = - sin_theta*sin(phi)*V;
```

The result is available in the variables C and S.

A.3 Application to Non-relativistic Datasets

Tensor fields do not only occur in general relativity, but also in other scientific domains. While the visualization methods for rendering tensor fields have been discussed for their application on general relativistic tensor fields, some of these methods are also applicable for mathematical equivalent data sets from other sources.

Application to MRI Data

Diffusion weighted magnetic resonance imaging (DW MRI) is a technique that measures the diffusion properties of water molecules in tissues[ZMB+03].With the availability of such measured tensor field data for medical purposes, the interest of visualizing such data has grown rapidly in the last years [ZDD+01, ZDK+01, TRW+02, HTR+02, TWD+01]. Anisotropic diffusion is described by the equation

$$\frac{\partial C}{\partial t} = \vec{\nabla} \cdot (D\vec{\nabla}C) \quad \text{or} \quad C_{,t} = (D^{ij}C_{,i}\,)_{,j}$$

whereby the scalar function $C : M \times \mathbb{R} \to \mathbb{R}$ is the time-dependent concentration of water molecules and $D : \mathcal{T}^*(M) \to \mathcal{T}(M)$ is a symmetric, positive definite contravariant tensor field of rank two. The inspection of diffusion tensor data is relevant for the segmentation and classification of MRI data to detect the white matter tracts that form the "wiring" of the human brain [KWH00]. Inspection of the measured matrix elements by themselves is not reasonable, because they are not invariant under rotations, hence depending on the scan measurement's orientation. Invariant visualization methods are thus important here. An exemplary dataset was provided generously by Gordon Kindlmann and Andrew L. Alexander. It is of certain interest for visualization purposes because it provides a large variety of tensor shapes, while still containing clear structures without overwhelming symmetries (as is the case with the inspected relativistic data sets). We will thus compare various methods upon this data set, using exactly the same view parameters.

As a first approach, we may employ metric ellipsoids (p.128) with colors indicating the trace of the tensor. We find, fig. A.2, that this representation clearly depicts the properties of the tensors at each point, but we need to enlarge the image such that each ellipsoid becomes visible on its own. When inspecting the entire image as an overview, hardly anything can be seen at all because the structures of the ellipsoids fall below the resolution and we could equivalently use volume rendering of the trace as a scalar field. But even when zoomed onto an interesting region, the ellipsoids are hard to interpret because we only see their projected shape and reconstructing their three-dimensional orientation from a two-dimensional image is more a matter of imagination than visualization.

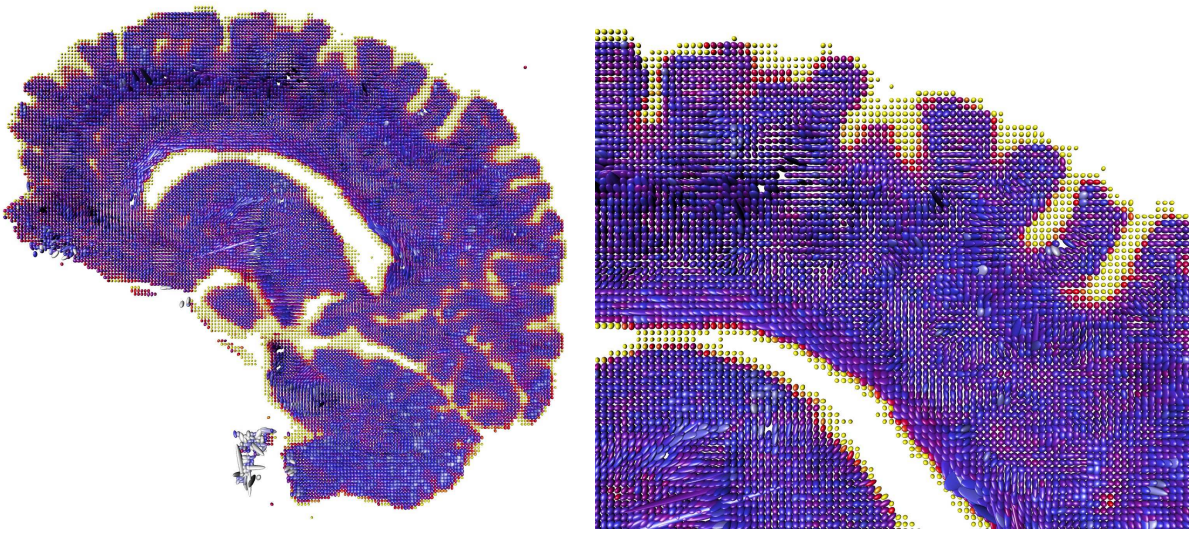


Figure A.2: Metric Ellipsoids applied to a slice of the Human Brain.

Employing the method of Tensor Glow (p.131) in fig. A.3 reduces the visual clutter. In the variant used here, the projected glow icon is normalized, but its transparency is set proportional to the trace of the tensor field. These settings enhance regions of high trace, i.e. the regions where water may flow rapidly. Such areas are depicted clearly, in an overview as well as in an enlargement. We also get an

glimpse of the orientation of the flow, but it is not too prominent as the anisotropy is rather small. The tensor glow method is thus applicable and helps to enhance certain features, but is not optimal for depicting anisotropy.

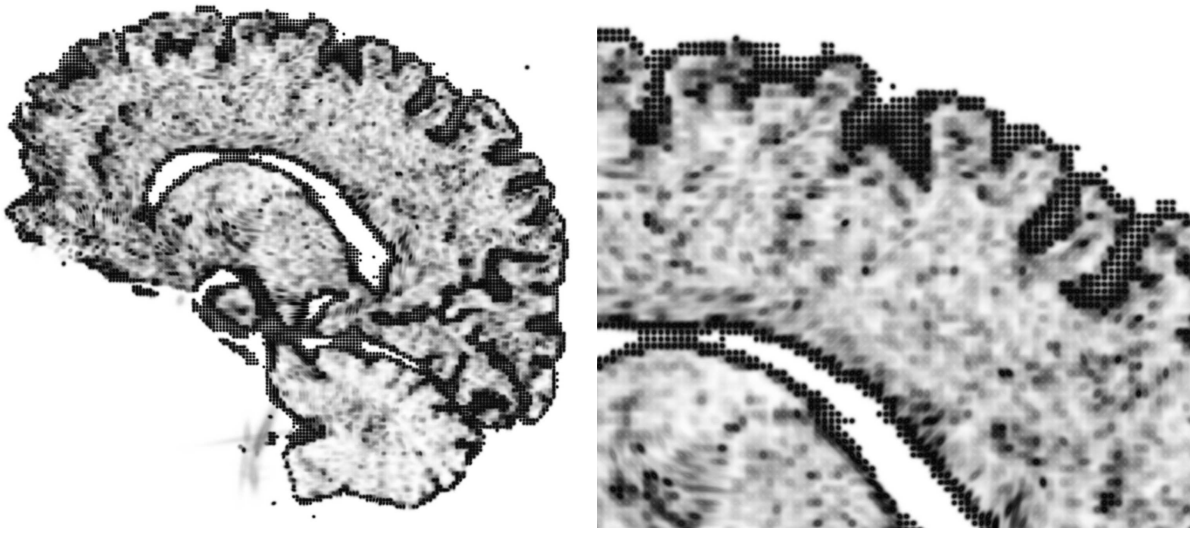


Figure A.3: Tensor Glow technique applied to a slice of the Human Brain.

Tensor Cones, p.133, are primarily inspired by the relativistic context. However, it may be applied to the MRI data set rather successful. In fig. A.4 we find global structures information more clearly than using "'metric ellipsoids"' or "'Tensor Glow"'. This is due to the larger sensitivity of the appearance of tensor cones to variations of the tensor field. As a consequence, we get a good overview of all structures contained in the data set. However, the interpretation is difficult because we have the vector field probe as an arbitrary input parameter. The structure of this user-chosen vector field is clearly recognizable, so we can study the tensor field properties by visual inspection, but it still requires some mental effort.

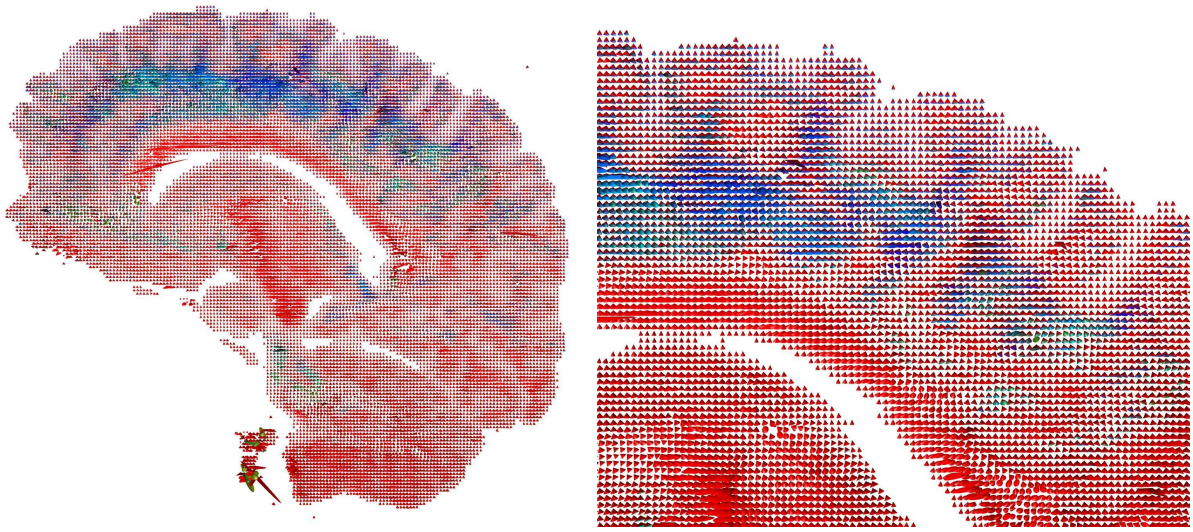


Figure A.4: Tensor Cones applied to a slice of the Human Brain.

Haber glyphs, p.135, have some history in computation fluid dynamics. They are very sensitive to anisotropy and are thus able to enhance global structures in an overview similar to the tensor cones, but without dependence on an user-chosen input vector field. The enlarged view also gives an hint of some large structures that incorporate a flow. However, to really recognize the details, we require an extreme enlargement such that all glyphs become resolved. As a drawback of Haber glyphs, see fig. A.5, they suffer from anisotropy artifacts, as the glyphs are randomly oriented in isotropic areas.

In contrast to Tensor Cones and Haber Glyphs, the technique of Tensor Schlieren, p.138, uses transparency as a fundamental part of the visualization algorithm. Thus, it is more suitable for large-scale

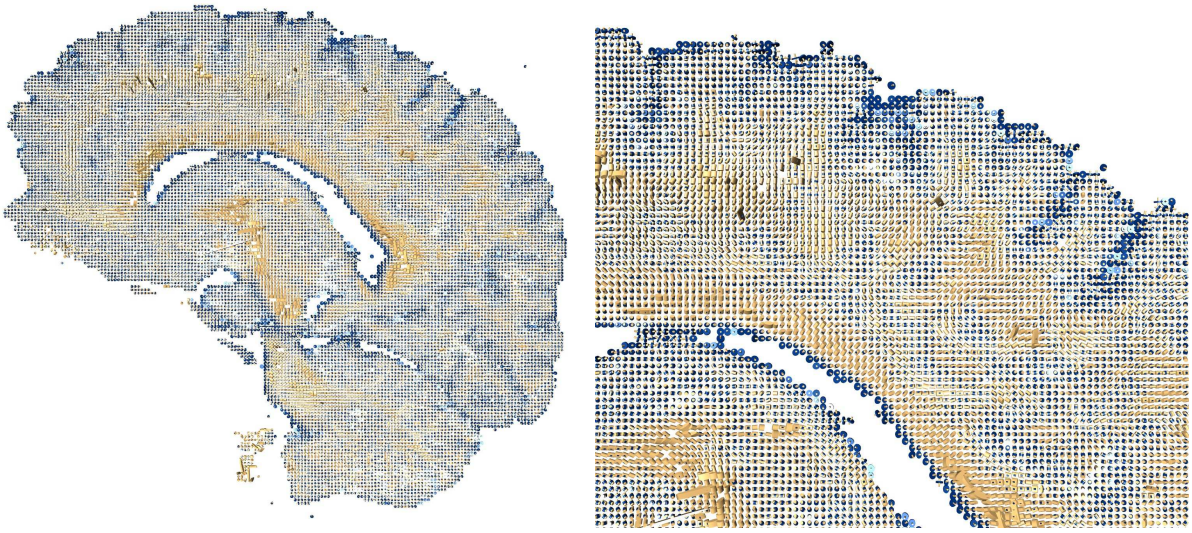


Figure A.5: Haber glyphs applied to a slice of the Human Brain.

overviews. However, transparency is not an invariant quantity here, but depends on the view direction, as the purpose of Tensor Schlieren is to enhance regions where the maximal eigenvector is perpendicular to the view direction. Tensor Schlieren are thus more suitable for an interactive environment than for static, two-dimensional images. However, even for static images it yields the best overview of the brain visualizations discussed so far: it reduces visual clutter by rendering large regions transparent (those where the maximal eigenvector is parallel to the view direction), while strongly displaying the orientation of the minor eigenvector in other regions. We thus get a good structural overview plus directional information in each area.

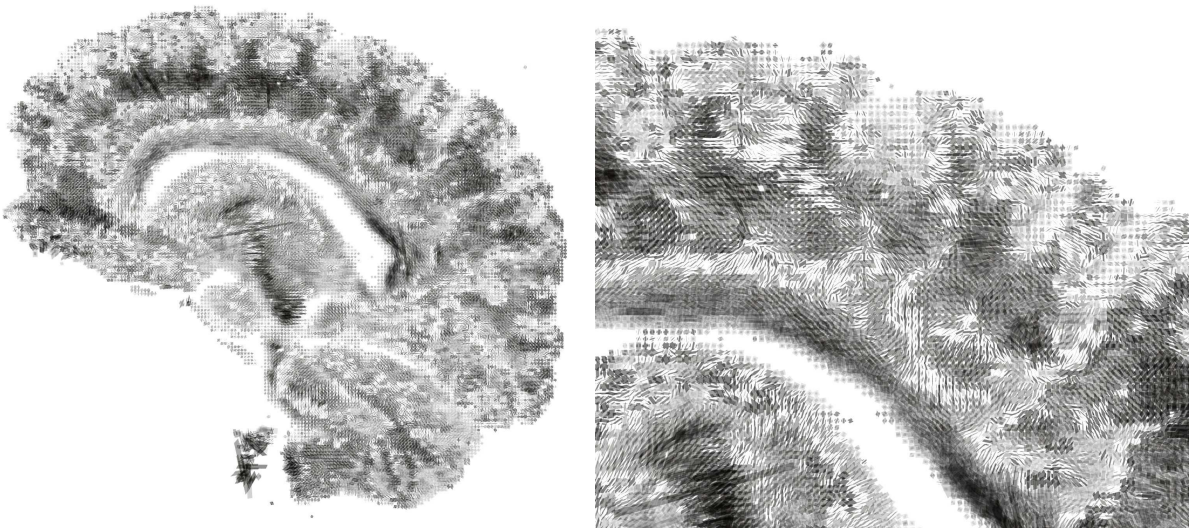


Figure A.6: Tensor Schlieren applied to a slice of the Human Brain.

While Tensor Schlieren produces view-dependent images encoding the orientation of the tensor field's eigenvectors by intensity, the technique of Tensor Splats p.140 uses colors for doing so and enhances the differences among the median and minor eigenvector by employing an additional one-dimensional high-frequency texturing. Transparency is used to encode the isotropy, i.e. isotropic regions are visually removed from the image. The result fig. A.7 is a strong enhancement for all anisotropic features with clear depiction of difference among minor and median eigenvectors as well. The Tensor Splat technique displays various features redundantly in different manners. In this respect, it is overdetermined and theoretically were able to display even more than the six quantities of a metric tensor field – in contrast to most former methods, that are designed to encode exactly six quantities or even less. However, this approach is not an overkill because for a certain view some glyph might not be able to display all of its quantities. For instance, although an ellipsoid encodes six parameters, we can only see three of them for a

certain view. The redundant display of parameters in the Tensor Splat technique avoids these problems. E.g. green indicates a linear region independent from its orientation and is thus clearly distinguishable from a red disk seen from aside. Tensor Splats thus appear to provide the best view of the discussed methods and due to their power also are appropriate for a full three-dimensional volume visualization. Their application to the human brain will thus be discussed in more detail.

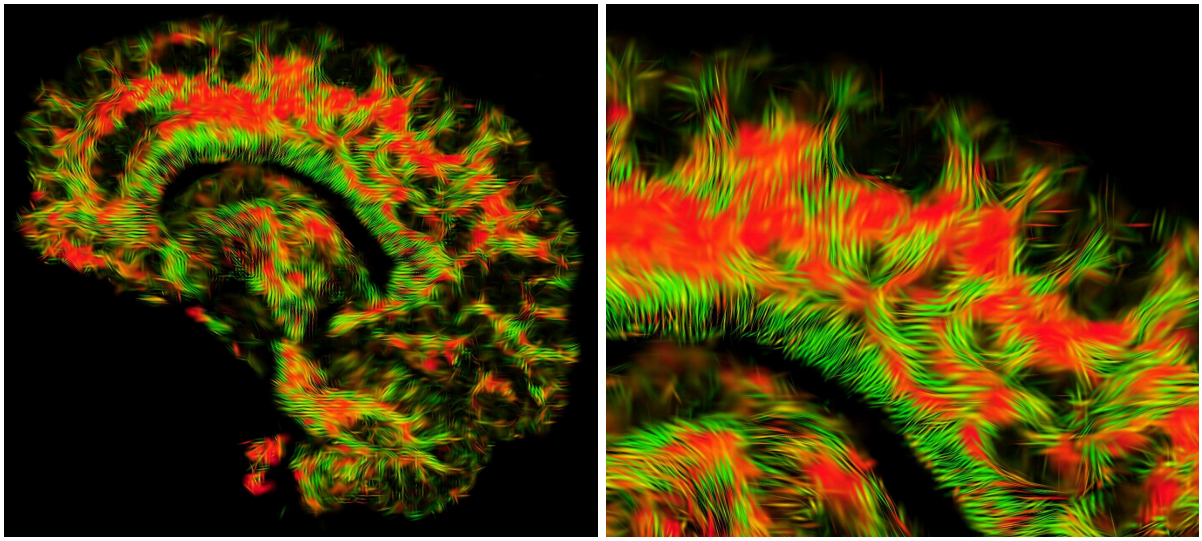


Figure A.7: Tensor Splats applied to a slice of the Human Brain.

Detailed Discussion of Tensor Splats Fig. A.8 demonstrates the tensor splats technique applied to a two-dimensional slice of a diffusion tensor scan of the human brain with a resolution of $148 \times 190 \times 160$ points. While the coloring provides a rough overview to distinguish between regions with highly linear (water diffusion happens mainly in one direction) and highly planar diffusion (water diffusion may occur in two directions), the finer detailed texture also shows the *direction* of the dominant diffusion. Interactive rotation of a 3D image on a compute screen or stereo viewing in a virtual reality environment provides a better perception of the actual orientation of planar structures. Alternatively, we can also inspect the inverse tensor field, right image in Fig. A.8, to map planar splats to linear ones and vice versa. The comparison of both views clearly displays the orientation of the planar diffusion. The high-contrast colors

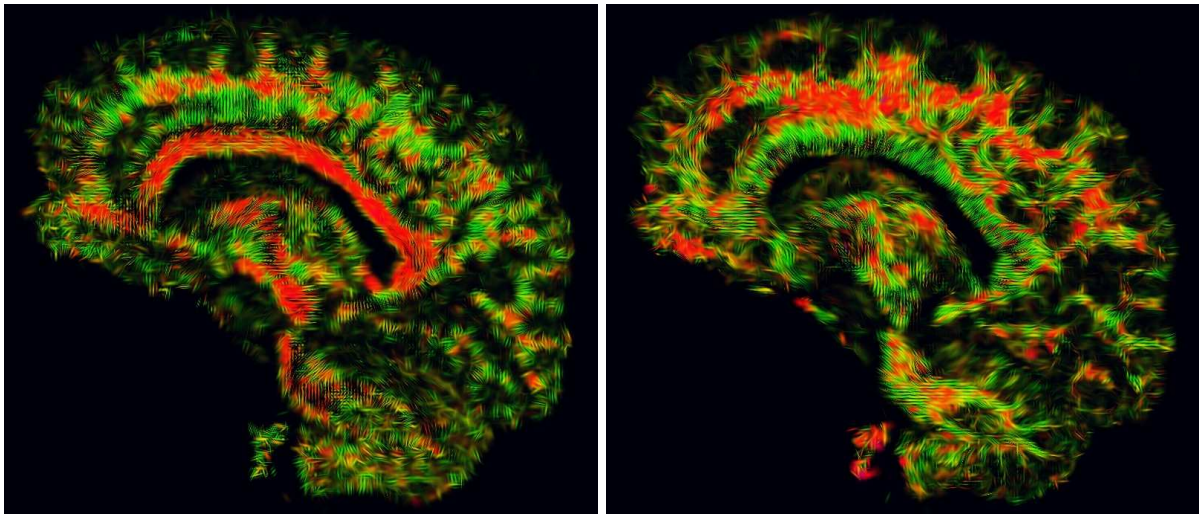


Figure A.8: A slice through the brain (left image), displaying diffusion "obstacles", and the inverse tensor (right image), displaying the possible diffusion velocities. Linear regions are easily detectable, planar regions appear smoother.

were chosen to emphasize the transition between linear and planar regions. However, the colors are just used as enhancement of information that is already contained in the texture. Using a black-white color

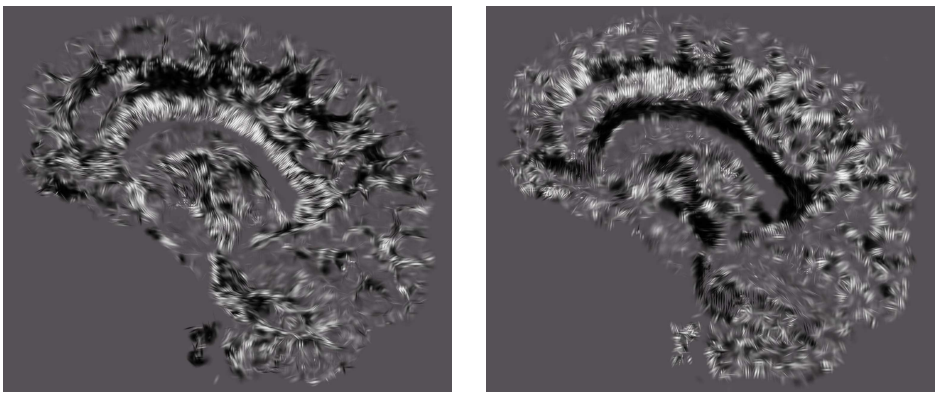


Figure A.9: Tensor Splats also work well in just gray-scale images. which is of interested when some image can only be printed in black and white.

transition as in fig. A.9 is sufficient as well, provided that high image resolution is possible to reveal the fine grass-like structures The Tensor Splats method extends easily to three-dimensional data sets, although it might require some fine-tuning of the visualization parameters to hide more of the isotropic regions than needed for two-dimensional slices. The full three-dimensional view of the brain data is shown in fig.A.10, whereas the visual impression is of course increased by high resolution images and direct user interaction with the three-dimensional geometry on a computer screen or VR projection device. The directional information as provided by the tensor splats technique also easily coexists with standard volume rendering. So beside color coding some scalar or vector field on the tensor splats themselves, a smooth volume rendering may be used in addition to display other scalar quantities. Useful examples are the trace of the tensor field, or the c_l shape factor field, as shown in Fig. A.11 to display the coincidence of highly linear regions and the directional information provided by the tensor splats.

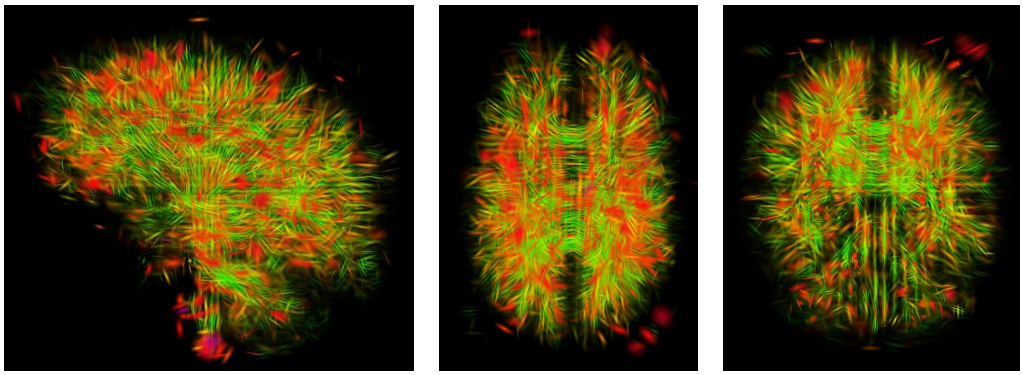


Figure A.10: Tensor Splats are capable to visualize an entire 3D volume of the Brain.

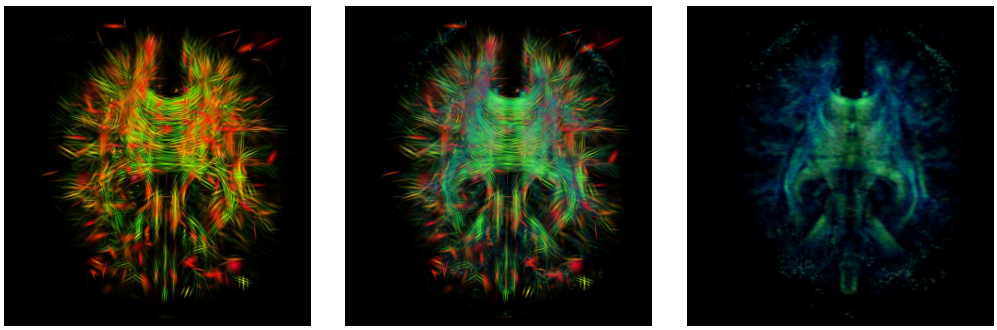


Figure A.11: While volume rendering of the anisotropy shape factor c_l (right) gives a notion *where* some highly linear regions occur, the Tensor Splats (left) also gives a notion in *which direction* such a linear diffusion takes place. The center image shows an overlay of volume rendering with Tensor Splats.

Eigenvector Streamlines The eigenvectors of a tensors are an essential and easily understandable property. Treating the eigenvectors of a tensor field as a vector field is not really correct because the sign of the eigenvectors is undetermined by the eigenvalue equation. Moreover, eigenvectors become undefined in isotropic regions. Still, we may try to employ vector field visualization methods like the illuminated stream line technique [ZSH96] upon eigenvector fields. To reduce or even avoid anisotropy artifacts, the transparency of the resulting stream lines is set proportional to the sphericity shape factor c_s. Then lines in the $c_s = 1$ regions of undefined maximal eigenvector become invisible, although a streamline of the maximal eigenvector field continues there. The seed points for the streamline integration, which determine the density and number of stream lines, are set dominantly in regions where the linearity factor c_l is close to one. Consequently, stream lines start in highly linear regions, may traverse through planar regions but are less dense there and vanish in isotropic parts of the volume data set. This approach is perfectly suitable

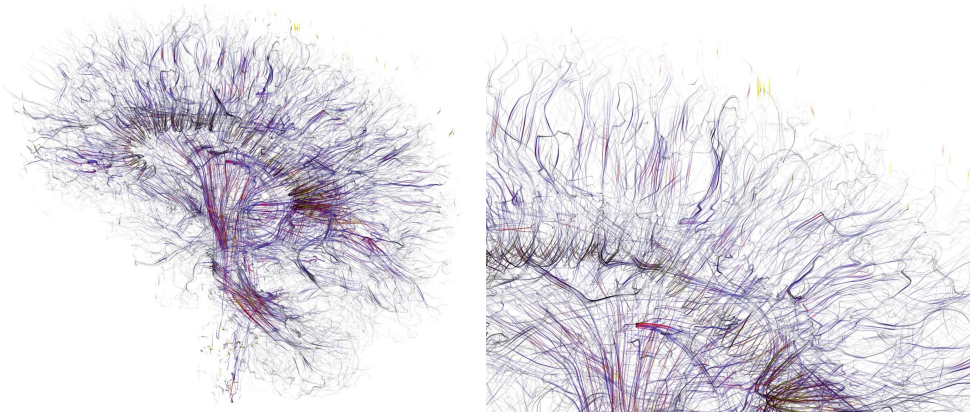


Figure A.12: Streamlines along the maximum eigenvector in linear regions.

for full three-dimensional visualization of a data set. Due to the three-dimensional nature of streamlines, it is not even applicable to two-dimensional slices. fig. A.12 displays a three-dimensional rendering with the same view parameters as used in the glyph comparison techniques before, fig. A.13 displays the full three-dimensional volume from different view positions. The ISL/transparency/line density based eigenvector streamline technique is able to display practically all of the tensor field features, including isotropic and linear regions in a clear way. However, it is not suitable for point-wise detailed inspection of a data set. Also, planar regions are not visualized correctly, since eigenvector stream lines visually suggest only one direction there.

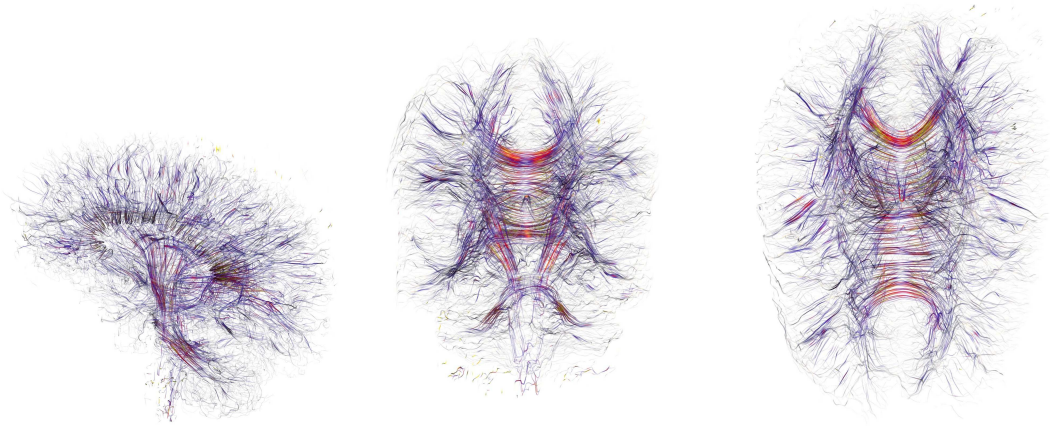


Figure A.13: Front, side and top view of streamlines along the maximum eigenvector in linear regions of the human brain data set.

Application to Distortion Vector Fields

If the underlying manifold is a vector space, then we may employ a vector field to compute an "elastic distortion" by remapping each point onto another point as given by the vector field:

$$v : \mathbb{R}^n \rightarrow \mathbb{R}^n \tag{A.62}$$

$$p \mapsto p + v(p) \tag{A.63}$$

A set of equidistant points will no longer remain equidistant under this mapping. If we assign the new point location with the same coordinates as the original points, we need to replace the euclidean metric with a non-euclidean one. This non-euclidean metric is directly related to the underlying distortion vector field. Visualization of this distortion tensor field is an alternative to displaying the vector field or the distorted point locations.

Derivation of the distortion tensor field Let x_0, x_1 be two points with unit distance, i.e. $\eta(x_1 - x_0, x_1 - x_0) = 1$ with η the euclidean metric. These points are mapped to $x_0 + v(x_0)$ and $x_1 + v(x_1)$. The distance between these mapped points is then

$$l := (x_1 + v(x_1)) - (x_0 + v(x_0)) \equiv (x_1 - x_0) + (v(x_1) - v(x_0)) =: \Delta x + \Delta v \quad .$$

We now are seeking a metric that fulfills $g(l, l) = 1$. It is the metric tensor field that is induced by the distortion vector field v. By insertion we find:

$$g(l,l) = g(\Delta x + \Delta v, \Delta x + \Delta v) = g(\Delta x, \Delta x) + g(\Delta x, \Delta v) + g(\Delta v, \Delta x) + g(\Delta v, \Delta v)$$

In a coordinate system, the distorted distance l is given component-wise:

$$l^i = \Delta x^i + \Delta v^i$$

To derive a metric tensor field in the tangential space, we are interested in the infinitesimal change $\Delta x \rightarrow 0$:

$$l^i{}_{,j} \equiv \frac{dl^i}{dx^j} = \lim_{\Delta x^j = 0} \frac{l^i}{\Delta x^j} = \lim_{\Delta x^j = 0} \frac{\Delta x^i}{\Delta x^j} + \frac{\Delta v^i}{\Delta x^j} = \frac{dx^i}{dx^j} + \frac{dv^i}{dx^j}$$

and thus may write the infinitesimal distortion vector as

$$l = l^i{}_{,j}\, dx^j \partial_i = (\delta^i_j + v^i{}_{,j}\,) dx^j \partial_i$$

Insertion yields

$$g_{ij} = \delta^n_i \eta_{mn} \delta^m_j + \delta^n_i \eta_{mn} v^m{}_{,j} + v^n_i \eta_{mn} \delta^m{}_{,j} + v^n_i \eta_{mn} v^m{}_{,j} = \delta_{ij} + v^i{}_{,j} + v^j{}_{,i} + \sum_k v^k{}_{,i}\, v^k{}_{,j} \tag{A.64}$$

This equation describes the induced "distortion tensor" of a vector field. It is sensitive to variations in the underlying vector field and becomes the Euclidean metric for a constant vector field. Based on this relationship, we may now use tensor field visualization methods to inspect arbitrary vector fields. The tensor splats visualization method is appropriate because it maps isotropic regions corresponding to unstretched grid regions to transparency. The visual appearance and interpretation is discussed at the following figures that have been created from a data set provided generously by Alexander Maye from ZIB. It describes the deviations of a certain individual bee brain from a standard bee brain.

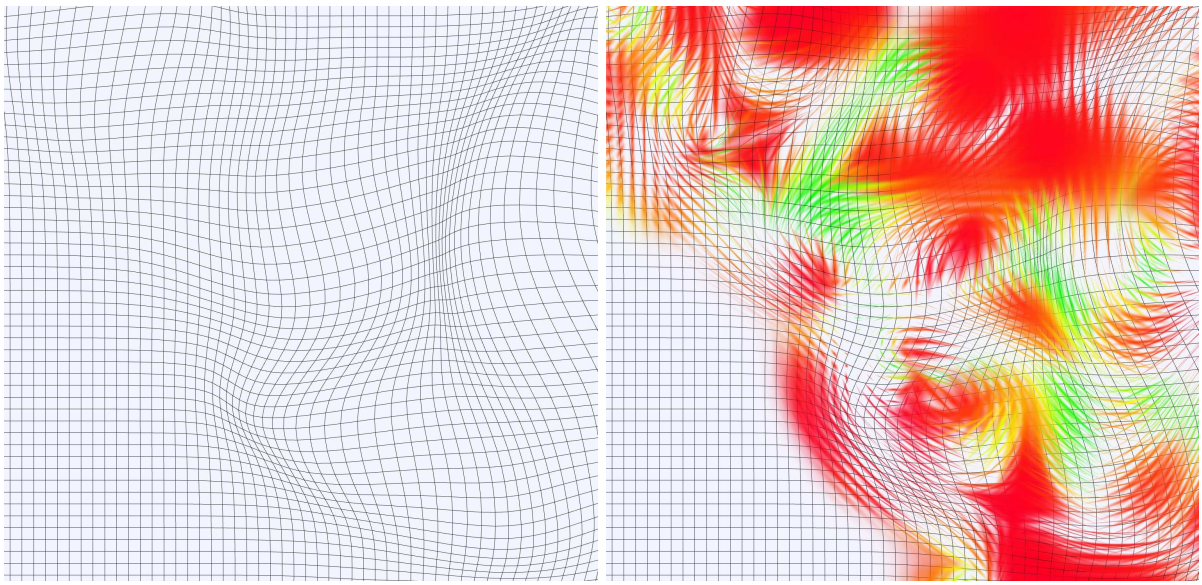


Figure A.14: Mapping of a uniform grid by a vector field: Left image displays the resulting non-uniform grid, the right image includes an overlay by Tensor Splats. Tensor Splats are highly sensitive to anisotropy and display even small deformations. Isotropic regions correspond to areas where the grid is constantly shifted and are displayed transparently. Note that the tensor field display reveals full three-dimensional information of the grid distortion, whereas the grid view only provides a two-dimensional slice, so some tensor splats are visible also in regions with apparently no grid distortion.

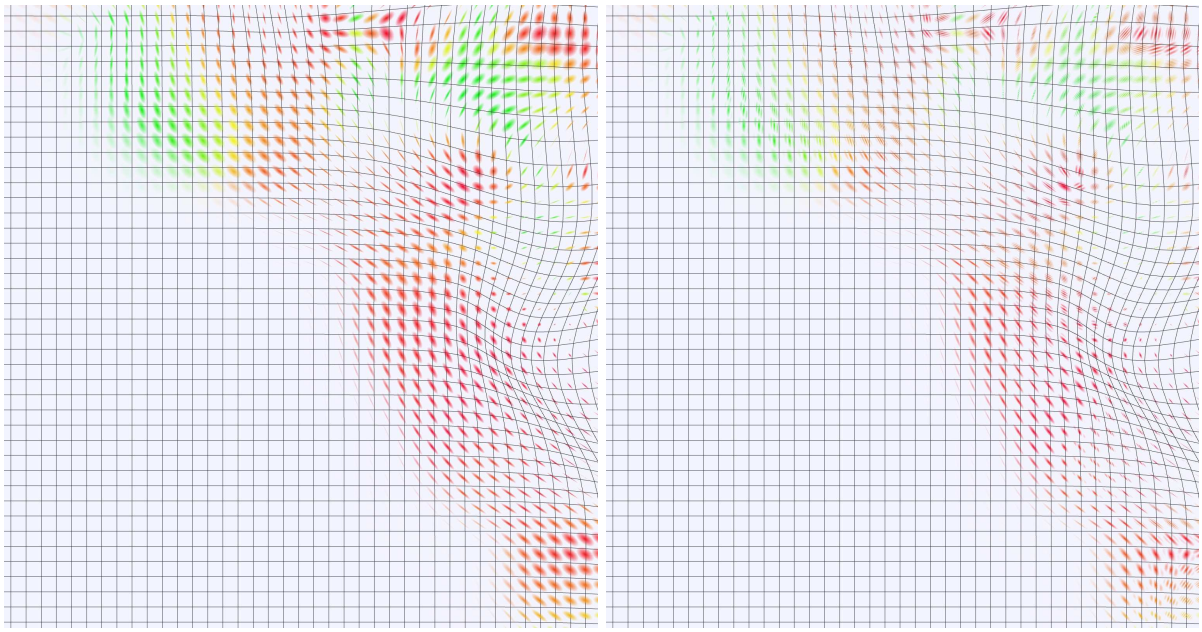


Figure A.15: Detail of the distortion tensor field. While we can clearly see the connection among grid compression and tensor field, we also find some visual ambiguities in the shape of the tensor splats within the left image: The shape of the green splats in the upper left area is just similar to the shape of the red splats in toward the lower right. However, grid distortion apparently only occurs in the domain of the red splats. Due to the color coding we know that the green splats indicate a linear region. Adding the tensor texturing – right image – also reveals the direction of the linearity: it is perpendicular to the projected major axis and thus shows that the linearity actually indicates grid stretching mostly perpendicular to the view plane. This interpretation is confirmed by a 3D zoom onto the region of interest as in fig. A.16.

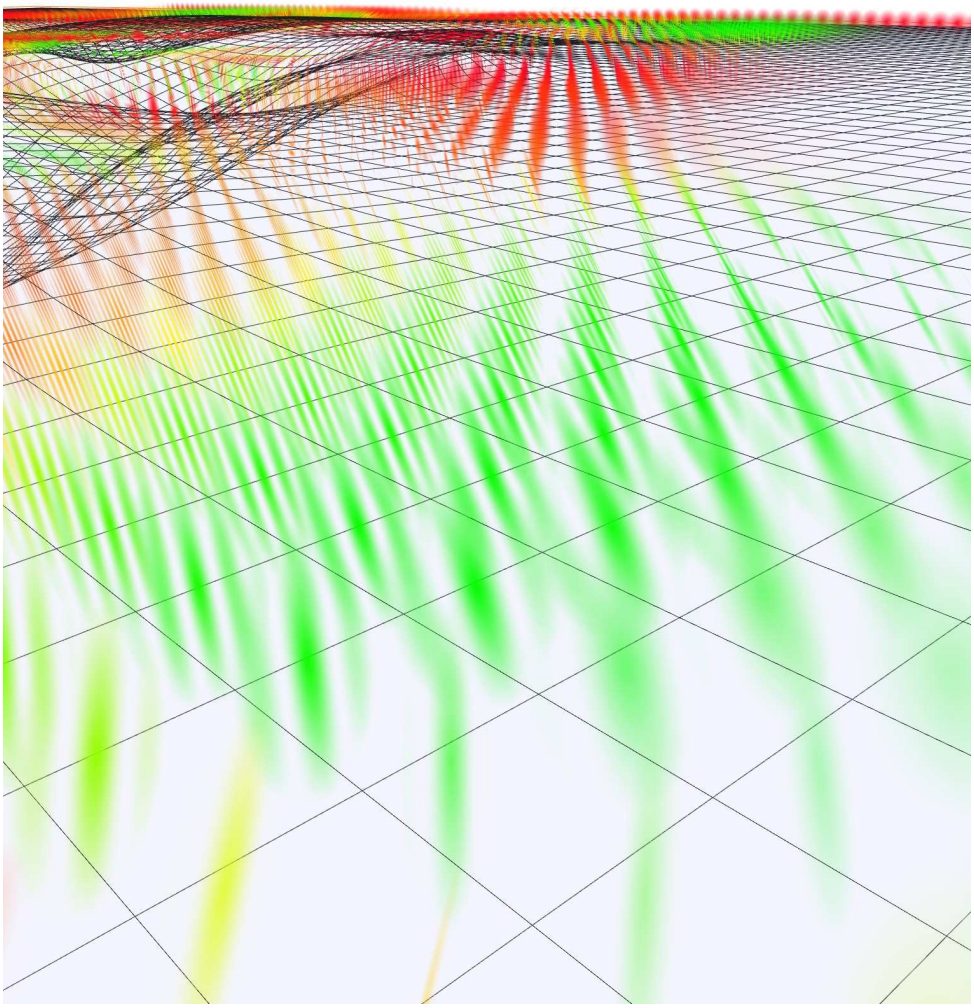


Figure A.16: 3D Zoom onto the region of high linearity in fig. A.15, revealing strong grid stretching perpendicular to the selected grid plane.

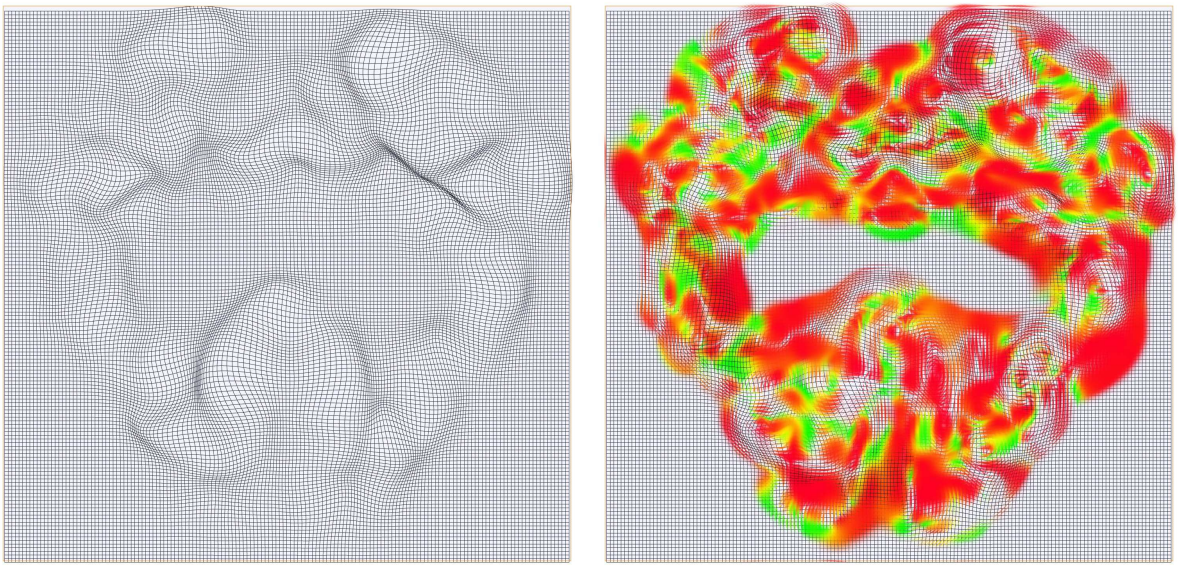


Figure A.17: Overview of the complete data set of a distorted grid and the corresponding induced metric tensor field as visualized by tensor splats.

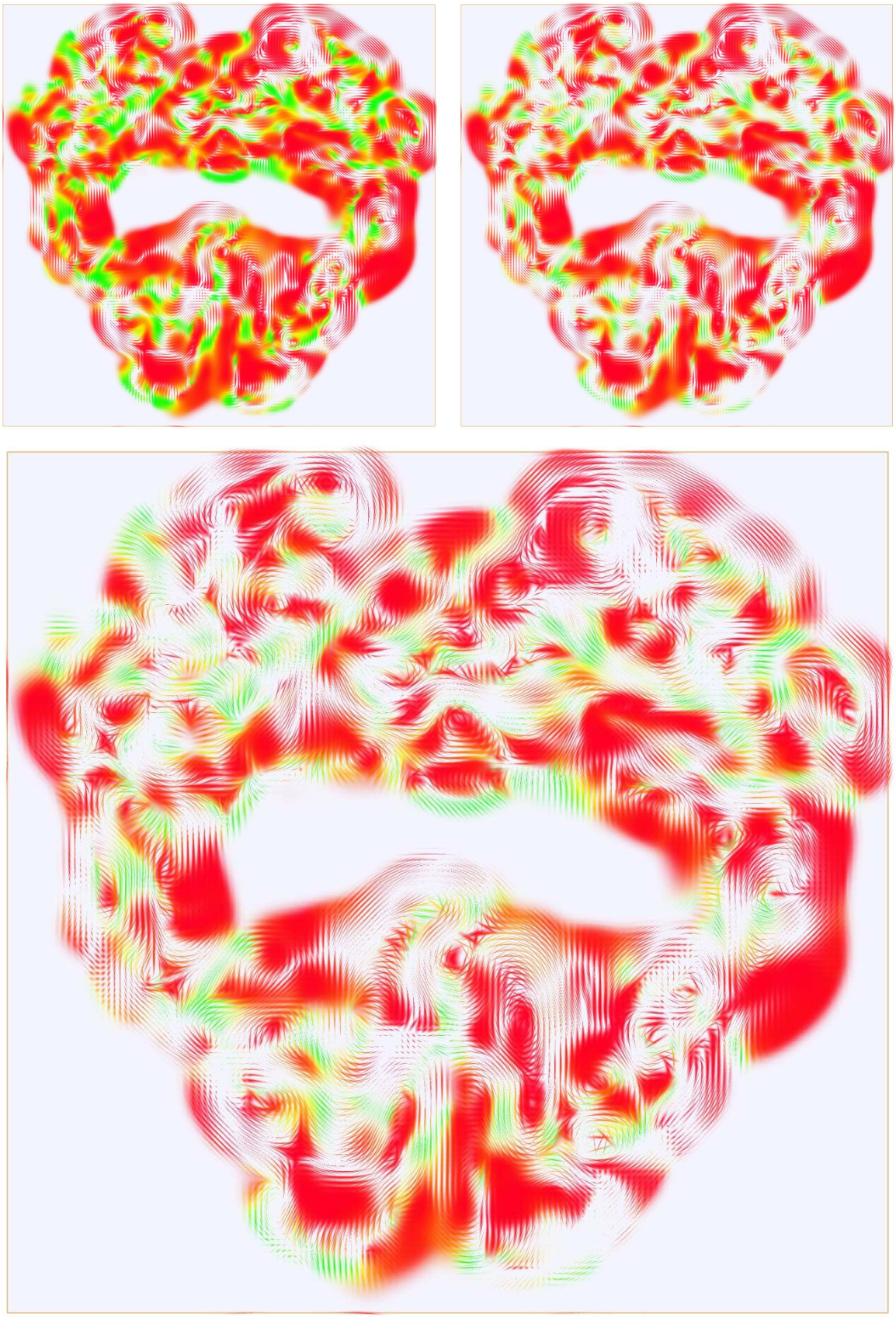


Figure A.18: Influence of the tensor splat texture frequency: upper left no texture, upper right one period, lower image using two periods per splat. Texturing of tensor splats helps to increase the visual perception of linearity as compared to planar regions, but also requires higher image resolution.

Bibliography

[ABB+01] M. Alcubierre, W. Benger, B. Bruegmann, G. Lanfermann, L. Nerger, E. Seidel, and R. Takahashi, *The 3d grazing collision of two black holes*, Phys.Rev.Lett. **87** (2001), http://de.arxiv.org/abs/gr-qc/0012079.

[ABD+01a] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radtke, and E. Seidel, *Cactus grid computing: Review of current development*, Proceedings of Euro-Par 2001: 7th International Euro-Par Conference on Parallel Processing (Rizos Sakellariou, Jon Keane, John R. Gurd, and Len Freeman, eds.), Lectures Notes in Computer Science, vol. 2150, Springer Verlag, Berlin Heidelberg 2001, October 2001, http://link.springer.de/link/service/series/0558/bibs/2150/21500817.htm, pp. 817–824.

[ABD+01b] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radtke, E. Seidel, and J. Shalf, *Cactus tools for grid applications*, Cluster Computing **4** (2001), no. 3, 179–188.

[AF92] K. Andrew and C. Fleming, *Space-time geometries characterized by solutions to the geodesic equations*, Computers in Physics **6** (1992), 498–504.

[AGL+99] G. Allen, T. Goodale, G. Lanfermann, E. Seidel, W. Benger, H.-C. Hege, A. Merzky, J.Massó, and J.Shalf, *Solving Einstein's Equation on Supercomputers*, IEEE Computer **32** (1999), no. 12, 52–59, http://www.computer.org/computer/articles/einstein_1299_1.htm.

[AHK+00] A.L. Alexander, K. Hasan, G. Kindlmann, D.L. Parker, and J.S. Tsuruda, *A geometrical analysis of diffusion tensor measurements of the human brain*, Magnetic Resonance in Medicine **44** (2000), 283–291.

[Alc94] Miguel Alcubierre, *The warp drive: hyper-fast travel within general relativity*, Class.Quant.Grav. **11** (1994), L73–L77, http://arxiv.org/abs/gr-qc/0009013.

[Bae99] John Baez, *This week's finds in mathematical physics (week 141)*, http://math.ucr.edu/home/baez/week141.html, 1999.

[BAS02] M. Bondarescu, M. Alcubierre, and E. Seidel, *Isometric embeddings of black hole horizons in three-dimensional flat space*, Class.Quant.Grav. **19** (2002), http://arxiv.org/abs/gr-qc/0109093.

[Bau75] Bruce G. Baumgart, *A polyhedron representation for computer vision*, 1975, pp. 589–596.

[BB92] David M. Butler and Steve Bryson, *Vector bundle classes from powerful tool for scientific visualization*, Computers in Physics **6** (1992), 576–584.

[BB02] C. Beetle and L.M. Burko, *A radiation scalar for numerical relativity*, http://arxiv.org/abs/gr-qc/0210019.

[Ben96] W. Benger, *Simulation of a black hole by raytracing*, Relativity and Scientific Computing - Computer Algebra, Numerics, Visualization (Berlin Heidelberg New York) (R.A. Puntigam F.W. Hehl and H. Ruder, eds.), Springer Verlag, 1996, http://www.photon.at/~werner/bh/, pp. 2–3.

[Ben97] ———, *Voids - der Einfluß der kosmologischen Konstanten auf die Vakuumblasen im Universum*, Master's thesis, Department of Astronomy, University of Innsbruck, 1997, http://www.photon.at/~werner/Voids/.

[Ben99a] ———, *Biggest crashes in the universe*, Max-Planck-Research Notes (1999), pp30, image publication for article by Uwe Seidenfaden.

[Ben99b] ———, *Lauscher fuer das urknall-echo*, Max-Planck-Forschung (1999), pp40, image publication for article by Uwe Seidenfaden.

[Ben00] ———, *Jagd auf Gravitationswellen*, Spektrum der Wissenschaften (2000), Cover Image, http://www.spektrum.de/archiv/aktuelles_heft.phtml?jahr=2000&monat=12.

[Ben01a] ———, *Beobachtungen im Datenraum*, SuW Special 6 Gravitation (2001), http://www.mpia-hd.mpg.de/suw/SuW/Programm/SuW-Special/P-Special.html.

[Ben01b] ———, *Gravitational physics - black hole blockbuster*, Nature (2001), no. 4, image publication, http://www.nature.com/cgi-taf/DynaPage.taf?file=/nature/journal/v413/n6855/full/413473a0_fs.html.

[Ben02a] ———, *Schwarze Löcher - die Monster im All*, Bild der Wissenschaft (2002), 48–49, image publication for the title story.

[Ben02b] ———, *Trout noirs*, Science & Vie (2002), no. 1022, Cover image.

[Ben03] ———, *Collisions De Trous Noirs*, Pour La Science (Edition Francaise De Scientific American) (2003), 48.

[Ber00] Guntram Berti, *Generic software components for scientific computing*, Ph.D. thesis, University of Cottbus, 2000, http://www.math.tu-cottbus.de/~berti/diss/.

[Bet98] C. Betts, *Fast rendering of relativistic objects*, The Journal of Visualization and Computer Animation (1998), no. 9, 17–31.

[BFN+99] W. Benger, I. Foster, J. Novotny, E. Seidel, J. Shalf, W. Smith, and P. Walker, *Numerical Relativity in a Distributed Environment*, Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, March 1999, http://www.zib.de/visual/papers/bengerw.ps.

[BG55] H. Bondy and T. Gold, *The field of a uniformly accelerated charge, with special reference to the problem of gravitational acceleration*, Proc.Roy.Soc A **229** (1955), 416–424.

[BH03] W. Benger and H.-C. Hege, *Analysing curved spacetimes with tensor splats*, 10th Marcel Grossmann Meeting, Rio de Janeiro, July 20-26, 2003, 2003, http://www.zib.de/visual/publications/sources/MGX.pdf.

[BH04] ———, *Tensor splats*, Conference on Visualization and Data Analysis 2004 (Erbacher, Chen, Roberts, Gröhn, and Börner, eds.), Proceedings of SPIE Vol. #5295, 2004, IS&T/SPIE Electronic Imaging Symposium in San Jose, CA, pp. 151–162.

[BHH00] W. Benger, H.-C. Hege, and S. Heusler, *Visions of numerical relativity*, Proceedings of the 3d International Conference on the Interaction of Art and Fluid Mechanics (SCART2000) (ETH Zürich Switzerland) (A. Gyr et al., ed.), Kluwer Academic Publishers, Feb. 28 - March 3 2000, http://www.zib.de/PaperWeb/abstracts/SC-99-53/, pp. 239–246.

[BHM+00] W. Benger, H.-C. Hege, A. Merzky, T. Radke, and E. Seidel, *Efficient distributed file i/o for visualization in grid environments*, Simulation and Visualization on the Grid (B. Engquist, L. Johnsson, M. Hammill, and F. Short, eds.), Lecture Notes in Computational Science and Engineering, vol. 13, Springer Verlag, 2000, http://www.zib.de/PaperWeb/abstracts/SC-99-43/, pp. 1–6.

[BM94] John C. Baez and Javier P. Muniain, *Gauge fields, knots, and gravity*, World Scientific Pub Co., September 1994.

[BMSW98] Carles Bona, Joan Masso, Edward Seidel, and Paul Walker, *Three dimensional numerical relativity with a hyperbolic formulation*, 1998, http://arXiv.org/abs/gr-qc/9804052.

[Bor98] E. Boring, *Visualization of tensor fields*, Master's thesis, University of California, Santa Cruz, 1998.

[BP89] David M. Butler and M. H. Pendley, *A visualization model based on the mathematics of fiber bundles*, Computers in Physics **3** (1989), no. 5, 45–51.

[BP98] E. Boring and A. Pang, *Interactive deformations from tensor fields*, Visualization '98, June 1998, pp. 297–304.

[Bri59] D. R. Brill, Ann. Phys. **7** (1959), 466–83.

[Bro98] Walter Brown, *Introduction to the SI library of unit-based computation*, `http://www.fnal.gov/docs/working-groups/fpcltf/Pkg/SIunits/doc/0SIunits.html`, 1998.

[Bru00] B. Bruegmann, *Numerical relativity in 3+1 dimensions*, Annalen Phys. **9** (2000), 227–246.

[Bry92] S. Bryson, *Virtual spacetime: An environment for the visualization of curved spacetimes via geodesic flows*, Visualization '92, Boston, 1992, pp. 291–298.

[Cha83] S. Chandrasekhar, *The mathematical theory of black holes*, Oxford, Clarendon Press, 1983.

[Cla70] C. J. S. Clarke, *On the global isometric embedding of pseudo-riemannian manifolds*, Proc. Roy. Soc. **A314** (1970), 417–428, `http://www.lns.cornell.edu/spr/2001-01/msg0030661.html`.

[CM] L.M. Cook and C.M. Matarazzo, *The tri-lab data models and format (dmf) project*, `http://www.ca.sandia.gov/asci-sdm/cgi-bin/sdmframedisplay.cgi/asci-sdm/DMFNecdc/DMFNecdcv4.html`.

[CM93a] R. A. Crawfis and N. Max, *Texture splats for 3d scalar and vector field visualization*, 1993, pp. 91–98.

[CM93b] R.A. Crawfis and N. Max, *Textured splats for 3d scalar and vector field visualization*, Visualization '93, San Jose (Nielson and Bergeron, eds.), IEEE CS Press, 1993, pp. 261–266.

[Cod] CodeSourcery, *Pooma - parallel object-oriented methods and applications*, `http://www.codesourcery.com/pooma/pooma`.

[Con03] Indeed Visual Concepts, *The amira visualization environment*, `http://www.amiravis.com/`, 2003.

[DB02] P. Deuflhard and F. Bornemann, *Scientific Computing with Ordinary Differential Equations*, Springer Verlag, New York, 2002.

[Deu76] P. Deuflhard, *On algorithms for the summation of certain special functions*, Computing **17** (1976), 37–48.

[DH93] T. Delmarcelle and L. Hesselink, *Visualizing second order tensor fields with hyperstream lines*, IEEE Computer Graphics and Applications **13** (1993), 25–33.

[DH03] P. Deuflhard and A. Hohmann, *Numerical Analysis in Modern Scientific Computing, Second Edition*, Springer Verlag, New York, 2003.

[Dra01] Norbert Dragon, *Geometrie der Relativitätstheorie / Geometry of the Theory of Relativity*, `http://www.itp.uni-hannover.de/~dragon/`, 2001.

[DW00] H. Ruder D. Weiskopf, U. Kraus, *Illumination and acceleration in the visualization of special relativity: a comment on fast rendering of relativistic objects*, The Journal of Visualization and Computer Animation (2000), no. 11, 185–195.

[Ein16] A. Einstein, *Die Grundlage der allgemeinen Relativitätstheorie*, Annalen der Physik **4** (1916), no. 49, p.284–339, available on the Albert Einstein Archives of the Jewish National and University Library/ the Hebrew University of Jerusalem, Archival call No. 120-788, `http://www.alberteinstein.info/gallery/pdf/CP6Doc30_pp284-339.pdf`.

[Eis49] L. Eisenhart, *Riemannian geometry*, Princeton University Press, 1949.

[Eng02] Henning Engeln, *Orte ohne wiederkehr*, GEO Magazin (2002), p.56/57, p.74/75, `http://www.geo.de/GEO/service/hefte/GEO/2002/11.html`.

[ES] Ellis and Stroustrup, *The annotated c++ reference manual*.

[Geo01] National Geographic, *Schwerkraftwellen erstmals sichtbar*, 2001, image publication, `http://www.nationalgeographic.de/php/magazin/redaktion/2001/12/redaktion_geographica.htm`.

[Gre99] Karen Green, *Colliding with a supercomputing record*, Access magazine (alliance/NCSA) **12** (1999), no. 3, cover image.

[GS85] Leonidas Guibas and Jorge Stolfi, *Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams*, ACM Transactions on Graphics **4** (1985), no. 2, 75–123.

[Hab90] R.B. Haber, *Visualization techniques for engineering mechanics*, Comp. Sys. in Engineering **1** (1990), no. 1, 37–50.

[Har99] J.C. Hart, *Using the cw-complex to represent the topological structure of implicit surfaces and solids*, Implicit Surfaces '99, Eurographics/SIGGRAPH, 1999, `http://basalt.cs.uiuc.edu/~jch/papers/cw.pdf`, pp. 107–112.

[Hau14] Felix Hausdorff, *Grundzüge der Mengenlehre*, von Veit, Leipzig, Germany, 1914, `http://mathworld.wolfram.com/TopologicalSpace.html`.

[HBM+00] H.-C. Hege, W. Benger, A. Merzky, F. Kasper, T. Radke, and E. Seidel, *Schwarze Löcher in Sicht - Immersive überwachung und Steuerung von Remote-Simulationen*, DFN-Mitteilungen (2000), no. 52, 4–6.

[HE73] Stephen W. Hawking and George F.R. Ellis, *The large scale structure of space-time*, Cambridge University Press, 1973.

[Hot02] I. Hotz, *Isometric embedding by surface reconstruction from distances*, Visualization 2002, 2002, pp. 251–257.

[HTR+02] J. Haueisen, D.S. Tuch, C. Ramon, P. Schmipf, V.J. Wedeen, J.S. George, , and J.W. Belliveau, *The influence of brain tissue anisotropy on human eeg and meg*, Neuroimage **15** (2002), 159–166.

[Hul92] Jeff P.M. Hultquist, *Constructing stream surfaces in steady 3d vector fields*, Visualization '92, IEEE Computer Society, 1992, pp. 171–178.

[IG97] V. Interrante and C. Grosch, *Strategies for effectively visualizing 3d flow with volume lic*, Visualization '97, ACM Press, 1997, pp. 421–424.

[Isr97] W. Israel, *An accelerated mirror emits negative radiation*, Personal communication at Bad Honnef Meeting, 1997.

[JB02] C. Lousto J. Baker, M. Campanelli, *The Lazarus project: A pragmatic approach to binary black hole evolutions*, Phys.Rev. **D65** (2002), gr-qc/0104063.

[JM88] J.Boehm and M.Weiser, *Garbage collection in an uncooperative environment*, Software Practice & Experience **18** (1988), no. 9, 807–820.

[KGM95] R.D. Kriz, E.H. Glaessgen, and J.D. MacRae, *Eigenvalue-eigenvector glyphs: Visualizing zeroth, second, fourth and higher order tensors in a continuum*, Workshop on Modeling the Development of Residual Stresses During Thermoset Composite Curing (1995), Report No. 95-19.

[Kit] Kitware, *Visualization toolkit*, `http://www.kitware.org/`.

[Kod86] Kunihiko Kodairo, *Complex manifolds and deformations of complex structures*, Springer Verlag, 1986.

[KW99] G. Kindlmann and D. Weinstein, *Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields*, Visualization '99, 1999, pp. 183 – 189.

[KWH00] G. Kindlmann, D. Weinstein, and D. Hart, *Strategies for direct volume rendering of diffusion tensor fields*, IEEE Transactions on Visualization and Computer Graphics **6** (2000), no. 2, 124–138.

[LAK+98] D.H. Laidlaw, E.T. Ahrens, D. Kremers, M. J. Avalos, R. E. Jacobs, and C. Readhead, *Visualizing diffusion tensor images of the mouse spinal cord*, June 1998, `www.cs.brown.edu/people/dhl/pdf/vis98dti.pdf`, pp. 127–134.

[LSP01] J. Li, H.-Y. Shum, and Q. Peng, *An improved spacetime ray tracing system for the visualization of relativistic effects*, Eurographics 2001 (short presentation), 2001.

[MKH95] H. Munthe-Kaas and M. Haveraaen, *Coordinate Free Numerics; How to avoid index-wrestling in tensor computations*, Tech. Report no. 101, Department of Informatics, University of Bergen, 1995, `http://www.ii.uib.no/~hans/papers/munthe-kaas95cfn1.ps`.

[MPIfGP03] Max-Planck-Institute for Gravitational Physics, *The cactus computational toolkit*, `http://www.cactuscode.org/`, 2003.

[MSM95] J.G. Moore, S.A. Schorn, and J. Moore, *Methods of classical mechanics applied to turbulence stresses in a tip leakage vortex*, Proceedings of the ASME Gas Turbine Conference, Houston, Texas, no. 95-GT-220, 1995.

[MTW73] C.W. Misner, K.S. Thorne, and J.A. Wheeler, *Gravitation*, W.H.Freeman and Company, 1973.

[Nas65] John Nash, *The imbedding problem for Riemannian manifolds*, Annals of Mathematics (1965), no. 63, 20–63, also (Ann. Math. 56 (1952) p405-421). `http://www.wikipedia.org/wiki/Nash_embedding_theorem`.

[NCS03] NCSA, *Hierarchical data format version 5*, `http://hdf.ncsa.uiuc.edu/HDF5/`, 2003, National Center for Supercomputing Applications, Illinois.

[NH96] H.P. Nollert and H. Herold, *Visualization in curved spacetimes - visualization of surfaces via embeddings*, Relativity and Scientific Computing (F.E. Hehl, R.A. Puntigam, and H. Ruder, eds.), Springer Verlag, 1996, pp. 330–351.

[NP62] E.T. Newman and R. Penrose, J. Math. Phys. (1962), no. 3.

[O'N83] B. O'Neill, *Semi-riemannian geometry, with applications to relativity*, Academic Press, Inc., 1983.

[Pet69] A. Z. Petrov, *Einstein spaces*, Pergamon Press, Oxford, 1969.

[Res97] IBM Research, *Data explorer data model*, `http://www.research.ibm.com/people/l/lloydt/dm/dx/dx_dm.htm`, March 1997.

[Rin93] W. Rindler, *Die Paradoxien der Relativitätstheorie*, Physik und Didaktik (1993), 259–272.

[Rot99] Josef Rothleitner, *Einführung in die Theoretische Physik, Teil I: Mechanik*, `http://th-physics.uibk.ac.at/teaching/scripts/einfg1.pdf`, 1999.

[RP95] J.D. Romano and R.H. Price, *Embedding initial data for black-hole collisions*, Class. Quantum Grav. **12** (1995), 875–894, `http://arxiv.org/abs/gr-qc/9409047`.

[S+99] SGI et al., *Standard template library*, `http://www.sgi.com/Technology/STL/`, 1999.

[Sch03] B.F. Schutz, *Gravity from the ground up*, Cambridge University Press, 2003.

[SEF99] Schneider, J. Ehlers, and Falco, *Gravitational lenses*, Springer-Verlag Berlin Heidelberg, 1999.

[SEHW02] A. Sigfridsson, T. Ebbers, E. Heiberg, and L. Wigstroem, *Tensor field visualization using adaptive filtering of noise fields combined with glyph rendering*, Visualization 2002, 2002, pp. 371–378.

[Set01] G.S. Settles, *Schlieren and shadowgraph techniques: Visualizing phenomena in transparent media*, Springer Verlag, 2001, `http://web.mit.edu/edgerton/people/vandiver/schlieren.html`.

[Sha03] John Shalf, *Personal communication*, 2003.

[Smo99] Lee Smolin, *The new universe around the next corner - theory of everything*, Physics World **12** (1999), 80p.

[SS01] J. Striegnitz and S.A. Smith, *An expression template aware lambda function*, Proceedings of the 2000 Workshop on C++ Template Programming, 10.10.2000, Erfurt Germany, 2001, `http://www.fz-juelich.de/zam/docs/autoren/striegnitz.html`, `http://www.kfa-juelich.de/zam/FACT/start/`.

[SSE94] S. Seitz, P. Schneider, and J. Ehlers, *Light propagation in arbitrary spacetimes and the gravitational lens approximation*, Class. Quant. Grav (1994), `http://arXiv.org/abs/astro-ph/950356`.

[Sta98] D. Stalling, *Fast texture-based algorithms for vector field visualization*, Ph.D. thesis, Free University Berlin, 1998.

[Ste91] J. Stewart, *Advanced general relativity*, Cambridge University Press, 1991.

[Str91] N. Straumann, *General Relativity and Relativistic Astrophysics*, second ed., Springer Verlag Berlin, 1991.

[Str96] ———, *Die Physik der Schwarzen Löcher*, 1996.

[SWH04] D. Stalling, M. Westerhoff, and H.-C. Hege, *Amira - an object oriented system for visual data analysis*, Visualization Handbook (Christopher R. Johnson and Charles D. Hansen, eds.), Academic Press, to appear 2004, `http://www.amiravis.com/`.

[Tho93] Kip S. Thorne, *Gekrümmter raum und verbogene zeit - einsteins vermächtnis*, Knaur Verlag, 1993, English title: Black Holes and Time Warps. Einsteins's Outrageous Legacy.

[TRW+02] D.S. Tuch, T.G. Reese, M.R. Wiegell, N.G. Makris, J.W. Belliveau, and V.J. Wedeen, *High angular resolution diffusion imaging reveals intravoxel white matter fiber heterogeneity*, Magn Reson Med. **48** (2002), 577–582.

[TWD+01] D.S. Tuch, V.J. Wedeen, A.M. Dale, J.S. George, and J.W. Belliveau, *Conductivity tensor mapping of the human brain using diffusion tensor mri*, Proc. Natl. Acad. Sci. USA (2001), no. 98, 11697–11701.

[TWHS03] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel, *Saddle connectors - an approach to visualizing the topological skeleton of complex 3d vector fields*, Proc. IEEE Visualization 2003 (Seattle, U.S.A.) (G. Turk, J. J. van Wijk, and R. Moorhead, eds.), October 2003, `http://www.zib.de/weinkauf/`.

[Vel] T.L. Veldhuizen, *C++ Templates are Turing Complete*, `http://citeseer.nj.nec.com/581150.html`.

[Vel95] ———, *Using C++ template metaprograms*, C++ Report **7** (1995), no. 4, 36–43, Reprinted in C++ Gems, ed. Stanley Lippman.

[Wag94] Peter Wagner, *Differenzierbare Mannigfaltigkeiten*, Lecture, 1994, Institute of Mathematics at the University of Innsbruck.

[WE] Jeffrey Winicour and Jürgen Ehlers, Personal communication.

[Wei00] D. Weiskopf, *Four-Dimensional Non-Linear Ray Tracing as a Visualization Tool for Gravitational Physics*, Visualization '00 (T. Ertl, B. Hamann, and A. Varshney, eds.), ACM Press, Oct. 2000, pp. 445–448.

[Wes90] Lee Westover, *Footprint evaluation for volume rendering*, 367–376.

[Whi32] H. Whitney, *Congruent graphs and the connectivity of graphs*, American J. Math (1932), no. 54, 150–168.

[Wit59] L. Witten, Phys. Rev. **113** (1959), 357.

[WKL99] D. Weinstein, G. Kindlmann, and E. Lundberg, *Tensorlines: Advection-diffusion based propagation through diffusion tensor fields*, IEEE Visualization 1999, IEEE Computer Society Press, 1999, pp. 249–253.

[WMK+99] C.F. Westin, S.E. Maier, B. Khidhir, P. Everett, F.A. Jolesz, and R. Kikinis, *Image processing for diffusion tensor magnetic resonance imaging*, MICCAI 99: Second International Conference on Medical Image Computing and Computer-Assisted Intervention (editors Taylor C, Colchester A, ed.), Springer Verlag, Heidelberg, Germany, 1999, pp. 441–452.

[WPG+97] C.F. Westin, S. Peled, H. Gudbjartsson, R. Kikinis, and FA. Jolesz, *Geometrical diffusion measures for mri from tensor basis analysis*, Proceedings of ISMRM, Fifth Meeting, Vancouver, Canada, April 1997, p. 1742.

[Yam89] Y. Yamashita, *Computer graphics of black holes: The extension of ray-tracings to 4-dimensional curved space-time*, Trans. IPS Japan **30** (1989), no. 5, 642–651, in Japanese.

[ZDD+01] Song Zhang, Cagatay Demiralp, Marco DaSilva, Daniel Keefe, David H. Laidlaw, Benjamin D. Greenberg, Peter J. Basser, Carlo Pierpaoli, E.A. Chiocca, and T. S. Diesboeck, *Toward application of virtual reality to visualization of DT-MRI volumes*, Proceedings MICCAI, October 2001.

[ZDK+01] Song Zhang, Cagatay Demiralp, Daniel Keefe, Marco DaSilva, Benjamin D. Greenberg, Peter J. Basser, Carlo Pierpaoli, E. A. Chiocca, T. S. Deisboeck, and David Laidlaw, *An immersive virtual environment for DT-MRI volume visualization applications: a case study.*

[ZMB+03] L. Zhukov, K. Museth, D. Breen, R. Whitaker, and A. Barr, *Level set segmentation and modeling of dt-mri human brain data*, Journal of Electronic Imaging **12** (2003), no. 1, 125–133.

[ZP02] X. Zheng and A. Pang, *Volume deformation for tensor visualization*, Visualization 2002, 2002, pp. 379–386.

[ZP03] ———, *Interaction of light and tensor fields*, VisSym '03, 2003, pp. 157 – 166.

[ZSH96] M. Zöckler, D. Stalling, and H.-C. Hege, *Interactive visualization of 3d-vector fields using illuminated streamlines*, Visualization '96, Oct./Nov. 1996, pp. 107–113.

Index

Appendix B

Acknowledgements

I am deeply grateful to Prof.Dr.Dr.h.c. Peter Deuflhard for shouldering supervision of this thesis and for supporting this work at the Zuse Institute Berlin under excellent working conditions. His visions and valuable comments lead the path on how to improve the proceeding work even further. I am especially appreciative for many substantial discussions and suggestions on numerical methods, such as pointing to an improved technique for summing spherical harmonic functions and providing hints on selecting the optimal integration schemes for computing geodesics.

Special thanks to Prof.Dr. Ed Seidel, leader of the numerical relativity group at the AEI (now at LSU). His constant energy on pushing research frontiers was always source of motivation and inspiration for me. His perpetual support in confronting theoretical physicist who are used to one-dimensional black and white diagrams with the requirement of using three-dimensional color-based visualizations of their data constituted a vital boundary condition for improving visualization methods and their applicability.

Special thanks also to the head of the ZIB visualization department, Hans-Christian Hege, for support within the ZIB and inspiring ideas. His enthusiasm about the idea of using fiber bundles as foundation of a data model was especially motivating.

Steffen Prohaska from the ZIB visualization department deserves thanks for his proof-reading, deep steeping in the topic of topology and data models and resulting valuable comments and suggestions. Similarly, great thanks go to Dr. Stefan Zachow, Dr. Konrad Polthier, Dr. Brygg Ullmer and Dr. Alexander Maye for their efforts of digging through the text.

Many valuable inspiring and supporting discussions have been exchanged with John Shalf from Lawrence Berkeley Laboratories, formerly at NCSA, whose unique experience with data models and scientific visualizations corresponds to many manpowers collected in a single person. Invaluable support was provided by the members of the CACTUS group at the Albert-Einstein-Institute in Golm and their worldwide collaboration members for providing data and in-depth discussions on all topics.

This work could not have been done without the colleagues from the ZIB Visualization department and their workhorse software “Amira”, which provides a suitably general framework for implementing even exotic ideas like the fiber bundle data model in a user-friendly way.

Special thanks go to Dr. habil. Peter Wagner for many fruitful discussions about the topics of topology and differential geometry, and for repeated proof-reading with his instantaneous view for even minor deviations from precise mathematical definitions.

The surface model of a Pegasus horse used for a demonstration of the surface contour shading technique was created by Marcel Ritter and Alex Senfter. The diffusion tensor field data set of the human brain was kindly provided by Gordon Kindlmann and Andrew L. Alexander.

Appendix C

Biographical Information

Werner Benger - Curriculum vitae

6.6.1968	Date of birth, Innsbruck. Parents: Univ. Prof. DDr. Johannes Benger, head of department for hygiene at the University of Innsbruck and Werhild Benger, born Hoppe
Nationality	Austrian and German
Education	elementary school in Igls, then humanistic branch at the Akademisches Gymnasium Innsbruck
July 1987	school leaving examination and matriculation at university of Innsbruck, inscription for physics
1988	additional inscription for astronomy
Oct. 1995 - Sep. 1996	alternative civilian service, Bahnhofsozialdienst der Caritas in Innsbruck
11. July 1997	degree in astronomy as "Mag.rer.nat." at the university of Innsbruck, passed with distinction
Dec. 1997 - present	scientific employee at the Zuse-Institute Berlin and the Max-Planck Institute for Gravitational Physics (Albert-Einstein-Institute AEI).

Appendix D

Zusammenfassung

Mit analytischen Methoden konnte der allgemeine Fall des Zweikörperproblems im Rahmen der allgemeinen Relativitätstheorie bislang nicht behandelt werden. Nur numerische Methoden sind aussichtsreich, astrophysikalisch realistische Kollisionsprozesse von Schwarzen Löchern, Neutronensternen und ähnlichen Prozessen mit relativistischen Gravitationsfeldern beschreiben zu können. Die Ergebnisse derartiger Simulationsrechnungen werden zur Datenanalyse neuartiger Gravitationswellendetektoren benötigt. Zur Begutachtung der bei derartigen Simulationen anfallenden Datensätze werden geeignete Visualisierungsverfahren benötigt.

Die vorliegende Arbeit stellt zum einen ein Rahmenmodell zur Behandlung unterschiedlicher Datentypen vor, die bei numerischen Simulationsprozessen allgemein auftreten können. Die dabei entwickelten Strukturen wurden aufgrund der Anforderungen aus der allgemeinen Relativitätstheorie entwickelt, ihre mögliche Anwendbarkeit geht jedoch weit darüber hinaus. Ein mögliches Einsatzgebiet ist der standardisierte Datenaustausch zwischen unabhängigen Computeranwendungen, beispielsweise als allgemeines Dateiformat für wissenschaftliche Daten oder als Kommunikationsprotokoll im Netzwerkverkehr.

Zum anderen werden in der Arbeit wissenschaftliche Visualisierungsverfahren für Tensorfelder zweiter Ordnung verglichen und neu entwickelt. Derartige Daten stellen die Basisinformation numerisch bestimmter Raumzeiten dar, treten aber auch in anderen Wissenschaftsgebieten wie z.B. Strömungsmechanik und in der Medizintechnik auf. Während für Skalar- und Vektorfelder bereits viele ausgereifte Verfahren existieren, sind Verfahren zur Visualisierung von Tensorfeldern eher selten zu finden. Im Gegensatz zu Vektorfeldern sind hier (mindestens) doppelt so viele Informationen pro Raumpunkt vorhanden, die visuell umgesetzt werden müssen.

Obwohl es nicht unbedingt zweckmässig ist, nach einem “allgemein besten” Darstellungsverfahren zu suchen, da sich die möglichen Bedeutungen und die sich daraus ergebenden Anforderungen an die Darstellung eines Tensorfeldes stark unterscheiden können, erscheint die neu entwickelte Methode der “Tensor Splats” für weite Einsatzgebiete geeignet. Mithilfe einfacher Mittel können qualitativ hochwertige Visualisierungen mit hoher Aussagekraft in Echtzeit erzielt werden, die eine unmittelbare visuelle Interpretation des zugrundeliegenden sechsdimensionalen Datensatzes erlauben. Während ein geübter Wissenschaftler quantitative Informationen abzulesen imstande ist, kann auch der ungeübte Betrachter zumindest die relevanten Strukturen mit einem Blick erkennen.