# Using Geometric Algebra for Navigation in Riemannian and Hard Disc Space


Werner Benger
Center for Computation & Technology
Louisiana State University
239 Johnston Hall
Baton Rouge, LA 70803, USA
werner@cct.lsu.edu

Andrew Hamilton
Center for Astrophysics and Space Astronomy
JILA, University of Colorado
Boulder, CO 80309, USA
Andrew.Hamilton@colorado.edu

Mike Folk/Quincey Koziol
The HDF Group
1901 So. First St. Suite C-2
Champaign, IL 61820. USA
mfolk@hdfgroup.org

Simon Su
Princeton Institute for Computational Science and Engineering
345 Peter B. Lewis Library
Princeton, NJ 08544, USA
simonsu@princeton.edu

Erik Schnetter
Center for Computation & Technology
Dept. of Physics & Astronomy
Louisiana State University
Baton Rouge, LA 70803, USA
schnetter@cct.lsu.edu

Marcel Ritter/Georg Ritter
Department of Computer Science
University of Innsbruck
Technikerstrasse 21a
A-6020 Innsbruck, Austria
csab7885@uibk.ac.at



## ABSTRACT

A "vector" in 3D computer graphics is commonly understood as a triplet of three floating point numbers, eventually equipped with a set of functions operating on them. This hides the fact that there are actually different kinds of vectors, each of them with different algebraic properties and consequently different sets of functions. Differential Geometry (DG) and Geometric Algebra (GA) are the appropriate mathematical theories to describe these different types of "vectors". They consistently define the proper set of operations attached to each class of "floating point triplet" and allow to derive what meta-information is required to uniquely identify a specific type of vector in addition to its purely numerical values. We shortly review the various types of "vectors" in 3D computer graphics, their relations to rotations and quaternions, and connect these to the terminology of co-vectors and bi-vectors in DG and GA. Not only in 3D, but also in 4D, the elegant formulations of GA yield to more clarity, which will be demonstrated on behalf of the use of bi-quaternions in relativity, allowing for instance a more insightful formulation to determine the Newman-Penrose pseudo scalars from the Weyl tensor.


## 1. INTRODUCTION

Geometric Algebra [14] and the sometimes mystified concept of spinors eases implementation and intuition significantly, both in computer graphics and in physics [15]. We demonstrate the concrete application of these concepts in two independently developed computer graphic software packages, where Geometric Algebra is used for navigating the camera position in space and time. Another application example is given by a simulation code solving Einstein's equation in general relativity numerically on supercomputers, outputting the Newman-Penrose pseudo scalars as primary quantities of interest to study gravitational waves, both for visualization and observational verification.

Geometric Algebra moreover provides means to describe how the metadata information required per "vector" can be provided in persistent storage. Given large datasets that are expensively collected or generated by simulations requiring millions of CPU hours, it is increasingly important and difficult to be able to share and correctly interpret such datasets years after their generation, across different research groups from different fields of science. A unique, standardized, extensible identification of the geometric properties of the dataset elements is a necessary pre-requisite for this. similar to the way in which the IEEE standard for floating point values enables sharing floating point values. We utilize the mechanisms as provided by the HDF5 library here, a generic self-describing file format developed for large datasets as used in high performance computing. It allows specifying metadata in addition to the purely numerical data, providing an abstraction layer for specifying the mathematical properties on top of the lower-level binary layout. It is therefore desirable to us the functionality of this powerful I/O library to express the semantics of vector quantities as they arise in Geometric Algebra. This will be discussed in section 5.

## 2. VECTOR SPACES

A vector space over a field $F$ (such as $\mathbb{R}$) is a set $\mathcal{V}$ together with two binary operations *vector addition* $+ : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ and *scalar multiplication* $\circ : F \times \mathcal{V} \to \mathcal{V}$. The elements of $\mathcal{V}$ are called *vectors*. A vector space is closed under the operations $+$ and $\circ$, i.e., for all elements $u, v \in \mathcal{V}$ and all elements $\lambda \in F$ there is $u + v \in \mathcal{V}$ and $\lambda \circ u \in \mathcal{V}$ (vector space axioms). The vector space axioms allow computing the dif-

ferences of vectors and therefore defining the derivative of a vector-valued function $v(s) : \mathbb{R} \to \mathcal{V}$ as

$$\frac{d}{ds}v(s) := \lim_{ds\to 0} \frac{v(s+ds) - v(s)}{ds} \quad . \tag{1}$$

## 2.1 Tangential Vectors

In differential geometry, a tangential vector on a manifold $M$ is the operator $\frac{d}{ds}$ that computes the derivative along a curve $q(s) : \mathbb{R} \to M$ for an arbitrary scalar-valued function $f : M \to \mathbb{R}$:

$$\frac{d}{ds}f\Big|_{q(s)} := \frac{df\,(q(s))}{ds} \quad . \tag{2}$$

Tangential vectors fulfill the vector space axioms and can therefore be expressed as linear combinations of derivatives along the $n$ coordinate functions $x^\mu : M \to \mathbb{R}$ with $\mu = 0 \ldots n-1$, which define a basis of the tangential space $T_{q(s)}(M)$ on the $n$-dimensional manifold $M$ at each point $q(s) \in M$:

$$\frac{d}{ds}f = \sum_{\mu=0}^{n-1} \frac{dx^\mu\,(q(s))}{ds} \frac{\partial}{\partial x^\mu} f =: \sum_{\mu=0}^{n-1} \dot{q}^\mu \partial_\mu f \tag{3}$$

where $\dot{q}^\mu$ are the components of the tangential vector $\frac{d}{ds}$ in the chart $\{x^\mu\}$ and $\{\partial_\mu\}$ are the basis vectors of the tangential space in this chart. We will use the Einstein sum convention in the following text, which assumes implicit summation over indices occurring on the same side of an equation. Often tangential vectors are used synonymous with the term "vectors" in computer graphics when a direction vector from point $A$ to point $B$ is meant. A tangential vector on an $n$-dimensional manifold is represented by $n$ numbers in a chart.

## 2.2 Co-Vectors

The set of operations $df : T(M) \to \mathbb{R}$ that map tangential vectors $v \in T(M)$ to a scalar value $v(f)$ for any function $f : M \to \mathbb{R}$ defines another vector space which is dual to the tangential vectors. Its elements are called *co-vectors*.

$$< df, v >= df(v) := v(f) = v^\mu \partial_\mu f = v^\mu \frac{\partial f}{\partial x^\mu} \tag{4}$$

Co-vectors fulfill the vector space axioms and can be written as linear combination of co-vector basis functions $dx^\mu$:

$$df =: \frac{\partial f}{\partial x^\mu} dx^\mu \tag{5}$$

with the dual basis vectors fulfilling the duality relation

$$< dx^\nu, \partial_\mu >= \begin{cases} \mu = \nu : & 1 \\ \mu \neq \nu : & 0 \end{cases} \tag{6}$$

The space of co-vectors is called the co-tangential space $T^*_p(M)$. A co-vector on an $n$-dimensional manifold is represented by $n$ numbers in a chart, same as a tangential vector. However, co-vector transforms inverse to tangential vectors when changing coordinate systems, as is directly obvious from eq. (6) in the one-dimensional case: As $< dx^0, \partial_0 >= 1$ must be sustained under coordinate transformation, $dx^0$ must shrink by the same amount as $\partial_0$ grows when another coordinate scale is used to represent these vectors. In higher dimensions this is expressed by an inverse transformation matrix, as demonstrated in Fig. 1. In Euclidean

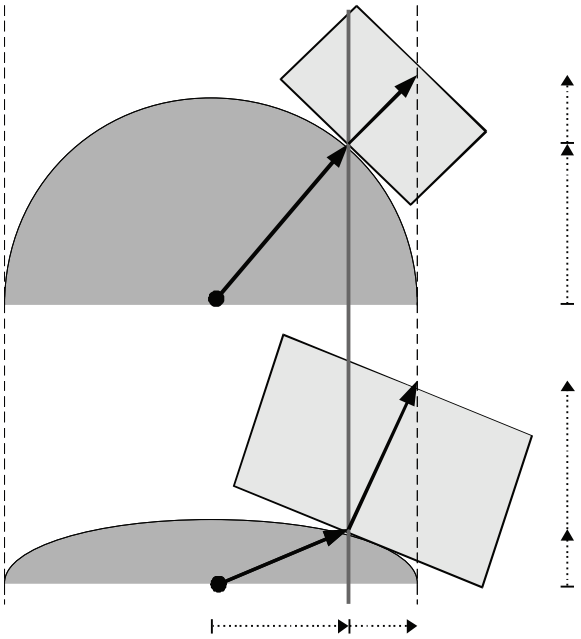

**Figure 1: Vector transformation under shrinking the height coordinate by a factor of two: tangential vectors (differences between two points) shrink in their height component by a factor two as well, whereas surface normal vectors (co-vectors) grow by a factor two in height, see the vertical components of the vector and co-vector shown on the right hand side in the figure.**

three-dimensional space, a plane is equivalently described by a "normal vector", which is orthogonal to the plane. While "normal vectors" are frequently symbolized via a vector arrow, like tangential vectors, they are not the same, rather they are dual to tangential vectors. It is more appropriate to visually symbolize them as a plane. This visual is also supported by (5), which can be interpreted as the total differential of a function $f$: a co-vector describes how a scalar function advances in space, which can be visualized as surfaces of constant function value ("isosurface"). On an $n$-dimensional manifold a co-vector is correspondingly symbolized by an $(n-1)$-dimensional subspace.

## 2.3 Tensors

A *tensor* $T^l_m$ of rank $l \times m$ is a multi-linear map of $l$ vectors and $m$ co-vectors to a scalar

$$T^l_m : \underbrace{T(M) \times \ldots T(M)}_{l} \times \underbrace{T^*(M) \times \ldots T^*(M)}_{m} \to \mathbb{R} \quad . \tag{7}$$

Tensors are elements of a vector space themselves and form the tensor algebra. They are represented relative to a coordinate system by a set of $k^{l+m}$ numbers for a $k$-dimensional manifold. The construction of an tensor of higher rank from lower rank is called the *outer product* (also known as tensor, dyadic or Kronecker product), denoted by $\otimes$:

$$T \equiv T^{\mu\nu}\partial_\mu \otimes \partial_\nu = v^\mu u^\nu \partial_\mu \otimes \partial_\nu = v^\mu \partial_\mu \otimes u^\nu \partial_\nu = v \otimes u \tag{8}$$

Tensors of rank 2 may be represented using matrix notation. Tensors of type $T_1^0$ are equivalent to co-vectors and called co-variant, in matrix notation (relative to a chart) they correspond to rows. Tensors of type $T_0^1$ are equivalent to a tangential vector and are called contra-variant, corresponding to columns in matrix notation. The duality relationship between vectors and co-vectors then corresponds to the matrix multiplication of a $1 \times n$ row with a $n \times 1$ column, yielding a single number

$$< a, b >=< a^\mu \partial_\mu, b_\mu dx^\mu > \equiv \ (a^0 a^1 \dots a^{n-1}) \begin{pmatrix} b^0 \\ b^1 \\ \dots \\ b^{n-1} \end{pmatrix} . \tag{9}$$

By virtue of the duality relationship (6) the contraction of lower and upper indices is defined as the *interior product* $\iota$ of tensors, which reduces the dimensionality of the tensor:

$$\iota : T_n^m \times T_k^l \to T_{n-l}^{m-k} : (u, v) \mapsto \iota_u v \tag{10}$$

The interior product can be understood (visually) as a generalization of some "projection" of a tensor onto another one.

Of special importance are symmetric tensors of rank two $g \in T_2^0$ with $g : T(M) \times T(M) \to \mathbb{R} : u, v \mapsto g(u, v) \equiv u \cdot v$ , $g(u, v) = g(v, u)$, as they can be used to define a *metric* on the tangential vectors, also called the *inner product* or dot product. Its inverse, defined by operating on the co-vectors, is called the co-metric. A metric, same as the co-metric, is represented as a symmetric $n \times n$ matrix in a chart for a $n$-dimensional manifold.

Given a metric tensor, one can define equivalence relationships between tangential vectors and co-vectors, which allow to map one into each other. These maps are called the "musical isomorphisms", $\flat$ and $\sharp$, as they raise or lower an index in the coordinate representation:

$$\flat : \quad T(M) \to T^*(M) : \quad v^\mu \partial_\mu \mapsto v^\mu g_{\mu\nu} dx^\nu \tag{11}$$

$$\sharp : \quad T^*(M) \to T(M) : \quad V_\mu dx^\mu \mapsto V_\mu g^{\mu\nu} \partial_\nu \tag{12}$$

As an example application, the "gradient" of a scalar function is given by $\nabla f = \sharp df$ using this notation. In Euclidean space, the metric is represented by the identity matrix and the components of vectors are identical to the components of co-vectors. As computer graphics usually is considered in Euclidean space, this justifies the usual negligence of distinction among vectors and co-vectors; consequently graphics software only knows about one type of vectors which is uniquely identified by its number of components. However, when dealing with coordinate transformations or curvilinear mesh types then distinguishing between tangential vectors and co-vectors is unavoidable. Treating them both as the same type within a computer program leads to confusions and is not safe. Section 4 will address this issue.

## 2.4 Exterior Product

The *exterior product* $\wedge : \mathcal{V} \times \mathcal{V} \to \Lambda^2(\mathcal{V})$ (also known as wedge product, Grassmann product, or alternating product) generates vector space elements of higher dimensions from elements of a vector space $\mathcal{V}$ by taking the antisymmetric part of the outer product (eq. 8) as

$$u \wedge v = \frac{1}{2} (u \otimes v - v \otimes u) \tag{13}$$

The new vector space is denoted $\Lambda^2(\mathcal{V})$. With the exterior product, $v \wedge u = -u \wedge v \quad \forall u, v \in V$, which consequently results in $v \wedge v = 0 \ \forall \ v \in \mathcal{V}$. The exterior product defines an algebra on its elements, the exterior algebra (or Grassman algebra) [9, 5]. It is a sub-algebra of the Tensor algebra consisting on the anti-symmetric tensors. The exterior algebra is defined intrinsically by the vector space and does not require a metric. For a given $n$-dimensional vector space $\mathcal{V}$, there can at most be $n$-th power of an exterior product, consisting of $n$ different basis vectors. The $n + 1$-th power must vanish, because at least one basis vector would occur twice, and there is exactly one basis vector for $\Lambda^n(\mathcal{V})$.

Elements $v \in \Lambda^k(\mathcal{V})$ are called $k$-vectors, whereby 2-vectors are also called bi-vectors and 3-vectors trivectors. The number of components of an $k$-vector of an $n$-dimensional vector space is given by the binomial coefficient $\binom{n}{k}$. For $n = 2$ there are two 1-vectors and one bi-vector, for $n = 3$ there are three 1-vectors, three bi-vectors and one tri-vector. These relationships are depicted by the Pascal's triangle, with the row representing the dimensionality of the underlying base space and the column the vector type:

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & 1 & & 2 & & 1 & & \\ & 1 & & 3 & & 3 & & 1 & \\ 1 & & 4 & & 6 & & 4 & & 1 \end{array} \tag{14}$$

As can be easily read off, for a four-dimensional vector space there will be four 1-vectors, six bi-vectors, four tri-vectors and one 4-vector. The $n$-vector of a $n$-dimensional vector space is also called a *pseudo-scalar*, the $(n - 1)$ vector a *pseudo-vector*.

## 2.5 Visualizing Exterior Products

An exterior algebra is defined on both the tangential vectors and co-vectors on a manifold. A bi-vector $v$ formed from tangential vectors is written in chart as

$$v = v^{\mu\nu} \partial_\mu \wedge \partial_\nu \quad , \tag{15}$$

a bi-covector $U$ formed from co-vectors is written in chart as

$$U = U_{\mu\nu} dx^\mu \wedge dx^\nu \quad . \tag{16}$$

They both have $\binom{n}{2}$ independent components, due to $v^{\mu\nu} = -v^{\nu\mu}$ and $U_{\mu\nu} = -U_{\nu\mu}$ (three components in 3D, six components in 4D). A bi-tangential vector (15) can be understood visually as an (oriented, i.e., signed) plane that is spun by the two defining tangential vectors, independently of the dimensionality of the underlying base space. A bi-co-vector (16) corresponds to the subspace of an $n$-dimensional hyperspace where a plane is "cut out". In three dimensions these visualizations overlap: both a bi-tangential vector and a co-vector correspond to a plane, and both a tangential vector and a bi-co-vector correspond to one-dimensional direction ("arrow"). In four dimensions, these visuals are more distinct but still overlap: a co-vector corresponds to a three-dimensional volume, but a bi-tangential vector is represented by a plane similar to a bi-co-vector, since cutting out a 2D plane from four-dimensional space yields a 2D plane again. Only in higher dimensions these symbolic representations become unique.

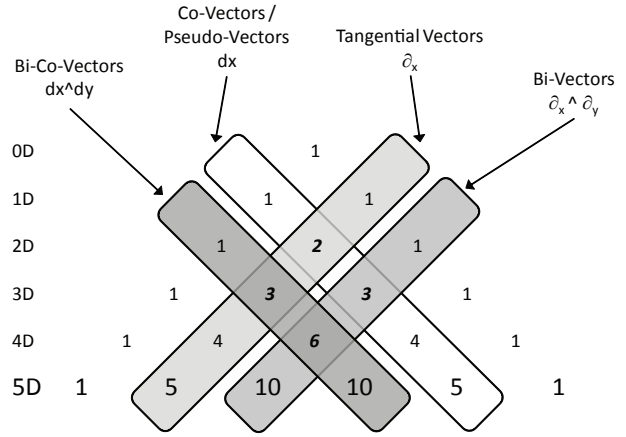


**Figure 2: Pascal's triangle showing the location of tangential vectors, bi-vectors, co-vectors and bi-covectors in the various subspaces in different dimensions. Especially in three dimensions there are many overlaps, indicating ambiguities where different quantities are all represented by "just three numbers". Similar situations occur in 4D, only in 5D all vector types become unambiguous.**

However, in any case a co-vector and a pseudo-vector will have the same appearance as an $n-1$ dimensional hyperspace, same as a tangential vector corresponds to an pseudo-co-vector:

$$V_\mu dx^\mu \quad \Leftrightarrow \quad v^{\alpha_0\alpha_1\dots\alpha_{n-1}}\ \partial_{\alpha_0} \wedge \partial_{\alpha_1} \wedge \dots \partial_{\alpha_{n-1}} \tag{17}$$

$$v^\mu \partial_\mu \quad \Leftrightarrow \quad V_{\alpha_0\alpha_1\dots\alpha_{n-1}}\ dx^{\alpha_0} \wedge dx^{\alpha_1} \wedge \dots dx^{\alpha_{n-1}} \tag{18}$$

A tangential vector – lhs of (18) – can be understood as one specific direction, but equivalently as well as "cutting off" all but one $n-1$-dimensional hyperspaces from an $n$-dimensional hyperspace – rhs of (18). This equivalence is expressed via the interior product of a tangential vector $v$ with an pseudo-co-scalar $\Omega$ yielding a pseudo-co-vector $V$ (19), similarly the interior product of a pseudo-vector with an pseudo-co-scalar yielding a tangential vector (19):

$$\iota_\Omega : T(M) \quad \to \quad (T^*)^{n-1}(M) : V \mapsto \iota_\Omega v \tag{19}$$

$$\iota_\Omega : T^{n-1}(M) \quad \to \quad T^*(M) : V \mapsto \iota_\Omega v \tag{20}$$

Pseudo-scalars and pseudo-co-scalars will always be scalar multiples of the basis vectors $\partial_{\alpha_0} \wedge \partial_{\alpha_1} \wedge \dots \partial_{\alpha_n}$ and $dx^{\alpha_0} \wedge dx^{\alpha_1} \wedge \dots dx^{\alpha_n}$. However, under when inversing a coordinate $x^\mu \to -x^\mu$ they flip sign, whereas a "true" scalar does not. An example known from Euclidean vector algebra is the allegedly scalar value constructed from the dot and cross product of three vectors $V(u,v,w) = u \cdot (v \times w)$ which is the negative of when its arguments are flipped:

$$V(u,v,w) = -V(-u,-v,-w) = -u \cdot (-v \times -w) \quad . \tag{21}$$

which is actually more obvious when (21) is written as exterior product:

$$V(u,v,w) = u \wedge v \wedge w = V \partial_0 \wedge \partial_1 \wedge \partial_2 \tag{22}$$

The result (22) actually describes a multiple of a volume element span by the basis tangential vectors $\partial_\mu$ - any volume must be a scalar multiple of this basis volume element, but can flip sign if another convention on the basis vectors is used. This convention depends on the choice of a right-handed versus left-handed coordinate system, and is expressed by the orientation tensor $\Omega = \pm \partial_0 \wedge \partial_1 \wedge \partial_2$. In computer graphics, both left-handed and right-handed coordinate systems occur, which often causes confusion.

By combining (20) and (12) – requiring a metric – we get a map from pseudo-vectors to vectors and reverse. This map is known as the *Hodge star operator* "$*$":

$$* : T^{n-1}(M) \to T(M) : V \longmapsto \sharp \iota_\Omega V \tag{23}$$

The same operation can be applied to the co-vectors accordingly, and generalized to all vector elements of the exterior algebra on a vector space, establishing a correspondence between $k-vectors$ and $n-k$-vectors. The Hodge star operator allows to identify vectors and pseudo-vectors, similarly to how a metric allows to identify vectors and co-vectors. The Hodge star operator requires a metric and an orientation $\Omega$.

A prominent application in physics using the hodge star operator are the Maxwell equations, which, when written based on the four-dimensional potential $A = V_0 dx^0 + A_k dx^k$ ($V_0$ the electrostatic, $A_k$ the magnetic vector potential) take the form

$$d * dA = J \tag{24}$$

with $J$ the electric current and magnetic flow, which is zero in vacuum. The combination $d * d$ is equivalent to the Laplace operator "$\Box$", which indicates that (24) describes electromagnetic waves in vacuum.

## 2.6 Geometric Algebra

Geometric Algebra is motivated by the intention to find a closed algebra on a vector space with respect to multiplication, which includes existence of an inverse operation. There is no concept of dividing vectors in "standard" vector algebra. Because the result of the inner and exterior product is of different dimensionality than their operands, they are not suited to define a closed GA on the vector space.

Geometric algebra postulates a product on elements of a vector space $u, v, w \in \mathcal{V}$ that is associative, $(uv)w = u(vw)$, left-distributive $u(v+w) = uv + uw$, right-distributive $(u+v)w = uw + vw$, and reduces to the inner product as defined by the metric $v^2 = g(v,v)$. It can be shown that the sum of the exterior product (which, within Geometric Algebra, is also called outer product, but should not be confused with the outer product $\otimes$ on tensors from eq. 8) and the inner product fulfill these requirements; this defines the *geometric product* as the sum of both:

$$uv := u \wedge v + u \cdot v \quad . \tag{25}$$

Since $u \wedge v$ and $u \cdot v$ are of different dimensionality ($\binom{n}{2}$ and $\binom{n}{0}$, respectively), the result must be in a higher dimensional vector space of dimensionality $\binom{n}{2} + \binom{n}{0}$. This space, called $\Lambda(\mathcal{V})$, is formed by the linear combination of $k$-vectors:

$$\Lambda(\mathcal{V}) = \bigoplus_{k=0}^{n} \Lambda^k(\mathcal{V}) \quad . \tag{26}$$

Its elements are called *multivectors*. The dimensionality of $\Lambda(\mathcal{V})$ is $\sum_{k=0}^{n} \binom{n}{k} \equiv 2^n$.

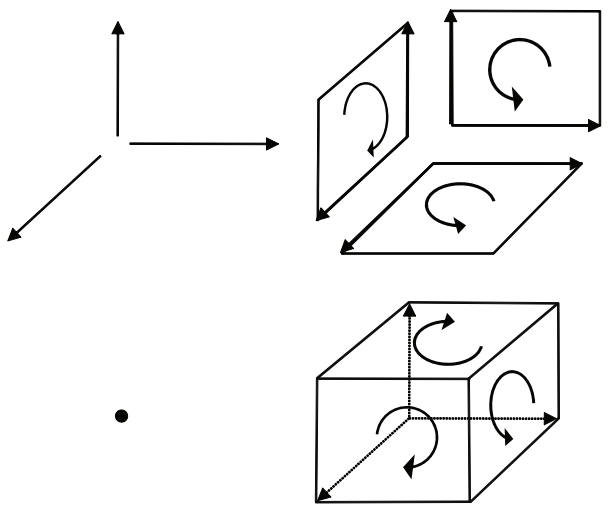

**Figure 3: Graphical representation of the $1+3+3+1$ structure of components that build a 3D multivector: three tangential vectors, three oriented planes, one scalar and one (oriented) volume element.**

For instance, in two dimensions the dimension of the space of multivectors is $2^2 = 4$. A multivector $V$, constructed from tangential-vectors on a two-dimensional manifold, is written as

$$V = V^0 + V^1\partial_0 + V^2\partial_1 + V^3\partial_0 \wedge \partial_1 \tag{27}$$

with $V^\mu$ the four components of the multivector in a chart. For a three-dimensional manifold a multivector on its tangential space has $2^3 = 8$ components and is written as

$$\begin{aligned} V =& V^0 + \\ & V^1\partial_0 + V^2\partial_1 + V^2\partial_2 + \\ & V^4\partial_0 \wedge \partial_1 + V^5\partial_1 \wedge \partial_2 + V^6\partial_2 \wedge \partial_0 + \\ & V^7\partial_0 \wedge \partial_1 \wedge \partial_2 \end{aligned} \tag{28}$$

with $V^\mu$ the eight components of the multivector in a chart. The components of a multivector have a direct visual interpretation, which is one of the key features of geometric algebra. In 3D, a multivector is the sum of a scalar value, three directions, three planes and one volume. These basis elements span the entire space of multivectors.

## 2.7 Spinors and Quaternions

Given a bi-vector $U = u \wedge v$ built from two orthonormal unit vectors $u, v$ (which fulfill $|u| = 1$, $|v| = 1$, $u \cdot v = 0$ under a given metric such that $U = uv$), we find that it provides the same algebraic properties as the imaginary unit $\sqrt{-1}$:

$$U^2 = UU = (uv)(uv) = (uv)(-vu) = -u(vv)u = -1 \tag{29}$$

This is a well known aspect of Geometric Algebra, which leads to $\binom{n}{2}$ distinct imaginary units on an $n$-dimensional vector space. For $n = 3$ we have three imaginary units (usually denoted as $i, j, k$), which relate to the three bi-vectors along the three coordinate axis. These three basis vectors $i = \partial_x \wedge \partial_y$, $j = \partial_y \wedge \partial_z$, $k = \partial_z \wedge \partial_x$ (equivalently to the co-vectors) fulfill $ijk = -1$, which is identical to the definitions used in quaternion algebra [7]. A quaternion consists of four components, a scalar and "vectorial" part. They represent the even parts of a multivector in 3D (28):

$$Q = Q^0 + Q^2\partial_0 \wedge \partial_1 + Q^0\partial_1 \wedge \partial_2 + Q^1\partial_2 \wedge \partial_0 \tag{30}$$

It can be shown that the even multivectors form a closed sub-algebra itself. GA provides a direct geometric insight for quaternions via (30), with the hard-to-memorable quaternion product being immersed within the easily remember-able geometric product. Given an even multivector (30), its dual as provided by the Hodge star operator (23) yields an odd multivector, consisting of a tangential vector and a pseudo-scalar (i.e., a volume element).

Quaternions are known in computer graphics for implementing rotations (for instance, the `SbRotation` class in OpenInventor), alternatively to rotation matrices (such as used in OpenGL). The same functionality is provided by *rotors* in GA. In 2D the right-multiplication of a vector $v = v^x\partial_x + v^y\partial_y$ with the bi-vector $\partial_x \wedge \partial_y = \partial_x\partial_y$ corresponds to a counter-clockwise rotation by $\pi/2$:

$$v(\partial_x \wedge \partial_y) = v^x\partial_x(\partial_x\partial_y) + v^y\partial_y(\partial_x\partial_y) = v^x\partial_y - v^y\partial_x \tag{31}$$

Therefore a rotation by an arbitrary angle $\varphi$ is written as a linear combination of a scalar component and a bi-vector, which is called a *rotor* (or *spinor*):

$$R = \cos\varphi + i\sin\varphi \equiv e^{i\varphi} \tag{32}$$

where $i$ is an arbitrary unit bi-vector fulfilling $i^2 = -1$. $e$ is the Euler number used here for defining the exponential function of a bi-vector, in style of the Euler equation. The inverse rotor (implementing clockwise rotation on right-multiplication, or counter-clockwise when applied from the left) is given by inverting the rotation angle

$$R^{-1} = e^{-i\varphi} = \cos\varphi - i\sin\varphi\ . \tag{33}$$

In two dimensions it is equivalent whether some vector $v$ is left-multiplied or right-multiplied with a rotor

$$vR^{-2} \equiv R^2v \equiv RvR^{-1}\ , \tag{34}$$

however in more than two dimensions the symmetric variant $RvR^{-1}$ with multiplying from the left and from the right has to be used to cancel out a tri-vector component that would otherwise occur (from multiplying the vector with a the bi-vector part of the rotor). While Quaternion Algebra is specific to three dimensions, the concept of a rotor in GA is independent from the dimensions and directly applicable to the 4D case, as will be reviewed in the next section.

## 2.8 Multilinear Multivector Maps

Same as a tensor is a multilinear map of vectors and co-vectors, we may represent multilinear maps of multivectors as a set of numbers given a specific chart. The Riemann tensor $R$, as described in 3.3, is such a case, is it can be seen as a map from bivectors to bivectors:

$$R : T_p(M) \wedge T_p(M) \to T_p(M) \wedge T_p(M) \tag{35}$$

The Riemann tensor can then be interpreted as argument of the Lorentz boost $e^{R(U)}$ resulting from a tiny circuit within a plane defined by the bivector $U$.

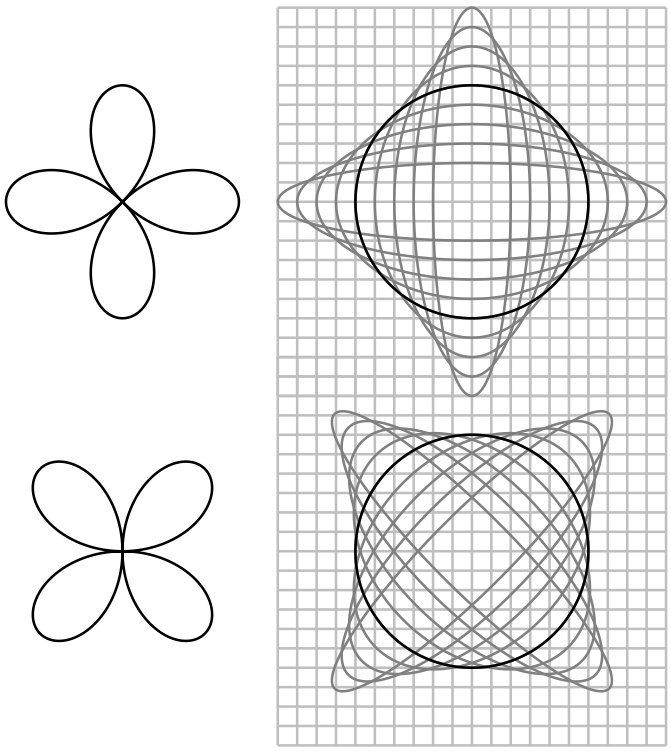

**Figure 4: The two linear polarizations of gravitational waves. The + polarization (top) has a $\cos 2\chi$ shape about the direction of propagation (into the paper), while the $\times$ polarization (bottom) has a $\sin 2\chi$ shape. A gravitational wave causes a system of freely falling test masses to oscillate relative to a grid of points a fixed proper distance apart.**

# 3. NEWMAN-PENROSE FORMALISM

General relativity predicts the existence of gravitational waves. There is a huge effort to detect gravitational waves expected for example from merging pairs of black holes [1]. To date no gravitational waves have been detected directly. There is however indirect evidence for their existence from the gradual decrease in orbital period of the binary pulsar, which is quantitatively consistent with the general relativistic prediction of energy loss by quadrupole emission of gravitational waves [4, 6].

It is conventional to characterize gravitational waves in terms of their Newman-Penrose (1962) (NP) components [16, 2, 18]. The purpose of this section is to give an idea of how this works, and how the geometric algebra offers insight into the NP formalism. The traditional derivation of the NP components of gravitational waves is magical, and shrouded in unnecessary and misleading notation. As Held (1974) [13] politely puts it, the NP formalism presents "a formidable notational barrier to the uninitiate".

The notion of a gravitational wave can be perplexing. A passing gravitational wave causes the distance between two freeling-falling masses to oscillate. But if gravity affects the very measurement of length itself, how can the distance between the masses be measured? The answer is that, despite the fact that in general relativity spacetime has no absolute existence, in the sense that the choice of coordinate system is arbitrary, nevertheless the metric asserts that there is a unique proper distance along a given path (or affine distance, along a null path) between any two points in spacetime (such as the path followed by a beam of laser light). The presence of gravity, or curvature, is expressed by the presence of a gravitational force between two points a fixed proper distance apart. A gravitational wave causes an oscillation in the differential gravitational force, or tidal force, between two points a fixed distance apart.

Figure 4 illustrates gravitational waves, in their two possible linear polarizations, + and $\times$. The grid represents a locally inertial system of points a fixed proper distance apart. The superposed ellipses represent a system of freely-falling test masses whose positions, initially on a circle, are being perturbed by a gravitational wave moving in a direction perpendicular to the paper. The proper distance between freely-falling test masses oscillates. That oscillation can be measured for example by the change in the number of wavelengths along a laser beam between the masses.

Sometimes one sees depictions of gravitational waves similar to Figure 4, but with the grid oscillating along with the ellipses. Such depictions are intended to convey the idea that gravitational waves are waves of spacetime (of the metric), but they are misleading, since they suggest that rulers oscillate along with the test masses, which is false.

## 3.1 Newman-Penrose tetrad

The Newman-Penrose (NP) formalism is particularly well adapted to treating waves that travel at the speed of light, which includes electromagnetic and gravitational waves. The NP formalism starts with the rest frame of an observer, and applies two tricks to it. The axes, or tetrad, of the observer's locally inertial frame form an orthonormal basis of vectors in the geometric algebra

$$\{\boldsymbol{\gamma}_t, \boldsymbol{\gamma}_x, \boldsymbol{\gamma}_y, \boldsymbol{\gamma}_z\} \ , \tag{36}$$

with the metric in Minkowski signature of the form

$$\boldsymbol{\gamma}_m \cdot \boldsymbol{\gamma}_n = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{37}$$

with indices $m$, $n$ running over $t, x, y, z$. The NP formalism chooses one axis, typically the $z$-axis, to be the direction of propagation of the wave.

The first NP trick is to replace the transverse axes $\boldsymbol{\gamma}_x$ and $\boldsymbol{\gamma}_y$ by spinor axes $\boldsymbol{\gamma}_+$ and $\boldsymbol{\gamma}_-$ defined by

$$\boldsymbol{\gamma}_+ \equiv \frac{1}{\sqrt{2}} \left( \boldsymbol{\gamma}_x + I \boldsymbol{\gamma}_y \right) \ , \quad \boldsymbol{\gamma}_- \equiv \frac{1}{\sqrt{2}} \left( \boldsymbol{\gamma}_x - I \boldsymbol{\gamma}_y \right) \ . \tag{38}$$

This is the same trick used to define the spinor components $L_\pm$ of the angular momentum operator $\boldsymbol{L}$ in quantum mechanics.

The second NP trick is to replace the time $t$ and propagation $z$ axes with outgoing and ingoing null axes $\boldsymbol{\gamma}_v$ and $\boldsymbol{\gamma}_u$,

defined by

$$\boldsymbol{\gamma}_v \equiv \frac{1}{\sqrt{2}}\left(\boldsymbol{\gamma}_t + \boldsymbol{\gamma}_z\right) \ , \quad \boldsymbol{\gamma}_u \equiv \frac{1}{\sqrt{2}}\left(\boldsymbol{\gamma}_t - \boldsymbol{\gamma}_z\right) \ . \tag{39}$$

The resulting outgoing, ingoing, and spinor axes form a NP null tetrad

$$\{\boldsymbol{\gamma}_v, \boldsymbol{\gamma}_u, \boldsymbol{\gamma}_+, \boldsymbol{\gamma}_-\} \ , \tag{40}$$

with NP metric

$$\boldsymbol{\gamma}_m \cdot \boldsymbol{\gamma}_n = \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{41}$$

with indices $m$, $n$ running over $v, u, +, -$. The NP metric (41) has zeros down the diagonal. This means that each of the four NP axes $\boldsymbol{\gamma}_m$ is null: the scalar product of each axis with itself is zero. In a profound sense, the null, or light-like, character of each the four NP axes explains why the NP formalism is well adapted to treating fields that propagate at the speed of light.

Three kinds of transformation, considered further below, take a particularly simple form in the NP tetrad:

I Reflections through the transverse axis $y$;
II Rotations about the propagation axis $z$;
III Boosts along the propagation axis $z$.

### 3.1.1 Reflections

Under transformation I, a reflection through the $y$-axis, the spinor axes swap:

$$\boldsymbol{\gamma}_+ \leftrightarrow \boldsymbol{\gamma}_- \ , \tag{42}$$

which may also be accomplished by complex conjugation. Reflection through the $y$-axis, or equivalently complex conjugation, changes the sign of all spinor indices of a tensor component

$$+ \leftrightarrow - \ . \tag{43}$$

In short, complex conjugation flips spin, a pretty feature of the NP formalism.

### 3.1.2 Rotations

Under transformation II, a right-handed rotation by angle $\chi$ about the direction $z$ of propagation, the transverse axes $\boldsymbol{\gamma}_x$ and $\boldsymbol{\gamma}_y$ transform as

$$\begin{aligned} \boldsymbol{\gamma}_x &\rightarrow \cos\chi\, \boldsymbol{\gamma}_x - \sin\chi\, \boldsymbol{\gamma}_y \ , \\ \boldsymbol{\gamma}_y &\rightarrow \sin\chi\, \boldsymbol{\gamma}_x + \cos\chi\, \boldsymbol{\gamma}_y \ . \end{aligned} \tag{44}$$

It follows that the spinor axes $\boldsymbol{\gamma}_+$ and $\boldsymbol{\gamma}_-$ transform under a right-handed rotation by angle $\chi$ as

$$\boldsymbol{\gamma}_\pm \rightarrow e^{\pm I\chi}\, \boldsymbol{\gamma}_\pm \ . \tag{45}$$

The transformation (45) identifies the spinor axes $\boldsymbol{\gamma}_+$ and $\boldsymbol{\gamma}_-$ as having spin $+1$ and $-1$ respectively. More generally, an object can be defined as having spin $s$ if it varies by

$$e^{sI\chi} \tag{46}$$

under a rotation by angle $\chi$ about the direction of propagation. The NP components of a tensor inherit spin properties from that of the spinor basis. The general rule is that the spin $s$ of any tensor component is equal to the number of $+$ covariant indices minus the number of $-$ covariant indices:

$$\text{spin } s = \text{number of } + \text{ minus } - \text{ covariant indices} \ . \tag{47}$$

### 3.1.3 Boosts

The final transformation III, a boost along the $z$-axis, multiplies the outgoing and ingoing axes $\boldsymbol{\gamma}_v$ and $\boldsymbol{\gamma}_u$ by a blueshift factor $\epsilon$ and its reciprocal

$$\begin{aligned} \boldsymbol{\gamma}_v &\rightarrow \epsilon\, \boldsymbol{\gamma}_v \ , \\ \boldsymbol{\gamma}_u &\rightarrow (1/\epsilon)\, \boldsymbol{\gamma}_u \ . \end{aligned} \tag{48}$$

If the observer boosts by velocity $v$ in the $z$-direction away from the source, then the blueshift factor is the special relativistic Doppler shift factor

$$\epsilon = \left(\frac{1-v}{1+v}\right)^{1/2} \ . \tag{49}$$

The exponent $n$ of the power $\epsilon^n$ by which an object changes under a boost along the $z$-axis is called its boost weight. Thus $\boldsymbol{\gamma}_v$ has boost weight $+1$, and $\boldsymbol{\gamma}_u$ has boost weight $-1$. The NP components of a tensor inherit their boost weight properties from those of the NP basis. The general rule is that the boost weight $n$ of any tensor component is equal to the number of $v$ covariant indices minus the number of $u$ covariant indices:

$$\text{boost weight } n = \text{number of } v \text{ minus } u \text{ covariant indices} \ . \tag{50}$$

## 3.2 Electromagnetic waves

The properties of gravitational waves are in many ways similar to those of electromagnetic waves. Both kinds of waves are massless, traveling at the speed of light. A crucial difference is that gravitational waves are spin-2 (tensor) waves, whereas electromagnetic waves are spin-1 (vector) waves.

Recall the nature of electromagnetic waves. Electromagnetic waves are characterized by the electromagnetic field $F_{ij}$, which is an antisymmetric tensor, or bivector, with 6 distinct components. The 6 components are commonly collected into two 3-dimensional vectors, the electric and magnetic fields $E$ and $B$. The geometric algebra gives the insight that the electromagnetic field tensor, being a bivector, has a natural complex structure, in which the electric and magnetic fields together form a complex 3-vector $E + IB$.

With respect to a NP null tetrad (40), the electromagnetic bivector has 3 complex components, of spin respectively $-1$, 0, and $+1$, in accordance with the rule (47):

$$\begin{aligned} -1 &: F_{u-} \\ 0 &: \frac{1}{2}\left(F_{uv} + F_{+-}\right) \\ +1 &: F_{v+} \ . \end{aligned} \tag{51}$$

The complex conjugates of the 3 components are:

$$\begin{aligned} -1^* &: F_{u+} \\ 0^* &: \frac{1}{2}\left(F_{uv} - F_{+-}\right) \\ +1^* &: F_{v-} \ , \end{aligned} \tag{52}$$

whose spins have the opposite sign. Conventionally (Chandrasekhar 1983), the 3 complex spin components of the electromagnetic field bivector in the NP formalism are denoted

$$\begin{aligned} -1 &: \phi_2 \ , \\ 0 &: \phi_1 \ , \\ +1 &: \phi_0 \ . \end{aligned} \tag{53}$$

The notation, like much of the rest of conventional NP notation, is truly awful.

For outgoing electromagnetic waves, only the spin $-1$ component propagates, carrying electromagnetic energy far away from a source:

$$-1 \ : \ \text{propagating, outgoing} \ . \tag{54}$$

This propagating, outgoing $-1$ component has spin $-1$, but its complex conjugate has spin $+1$, so effectively both spin components, or helicities, of an outgoing wave are embodied in the single complex component. The remaining 2 complex NP components (spins 0 and 1) of an outgoing wave are short range, describing the electromagnetic field near the source.

Similarly, for ingoing waves, only the spin $+1$ component propagates.

The isolation of each propagating mode into a single complex NP mode, incorporating both helicities, is simpler than the standard picture of oscillating orthogonal electric and magnetic fields.

## 3.3 Gravitational waves

In electromagnetism, the electromagnetic field tensor is defined by the commutator of the gauge-covariant derivative. In general relativity, the analogous commutator of the covariant derivative is the Riemann curvature tensor $R_{klmn}$. The Riemann curvature tensor has symmetries which can be designated shorthandly

$$R_{([kl][mn])} \ . \tag{55}$$

Here [] denotes antisymmetry, and () symmetry. The designation (55) thus signifies that the Riemann curvature tensor $R_{klmn}$ is antisymmetric in its first two indices $kl$, antisymmetric in its last two indices $mn$, and symmetric under exchange of the first and last pairs of indices, $kl \leftrightarrow mn$. In addition to the symmetries (55), the Riemann curvature tensor has the totally antisymmetric symmetry

$$R_{klmn} + R_{kmnl} + R_{knlm} = 0 \ . \tag{56}$$

The symmetries (55) imply that that the Riemann curvature tensor is a symmetric matrix of antisymmetric tensors, which is to say, a $6 \times 6$ symmetric matrix of bivectors. A $6 \times 6$ symmetric matrix has 21 independent components. The additional condition (56) eliminates one degree of freedom, leaving the Riemann curvature tensor with 20 independent components.

In spacetime algebra any bivector $U$ (6 component) can be written as complex sum $U = (E + IB)\boldsymbol{\gamma}_t$ of two spatial 3-vectors $E = E^x\boldsymbol{\gamma}_x + E^y\boldsymbol{\gamma}_y + B^z\boldsymbol{\gamma}_z$ and $B = B^x\boldsymbol{\gamma}_x + B^y\boldsymbol{\gamma}_y + B^z\boldsymbol{\gamma}_z$, due to the identity $I\boldsymbol{\gamma}_x\boldsymbol{\gamma}_t \equiv \boldsymbol{\gamma}_y\boldsymbol{\gamma}_z$ etc. In analogy to electromagnetism, $E\boldsymbol{\gamma}_t$ is called the electric bivector, $B\boldsymbol{\gamma}_t$ the magnetic bivector. The Riemann tensor, a multilinear multimap on bivectors eq. (35), can then be organized into a $2 \times 2$ matrix of $3 \times 3$ blocks with bivector indices, yielding the structure

$$\begin{pmatrix} R_{EE} & R_{EB} \\ R_{BE} & R_{BB} \end{pmatrix} \ . \tag{57}$$

The condition of being symmetric implies that $R_{EE}$ and $R_{BB}$ are symmetric, while $R_{BE} = (R_{EB})^\top$. The condition (56) states that the $3 \times 3$ block $R_{EB}$ (and likewise $R_{BE}$) is traceless.

The natural complex structure of bivectors in the geometric algebra suggests recasting the $6 \times 6$ Riemann curvature matrix (57) into a $3 \times 3$ complex matrix, which would have the structure $(R_E + IR_B)(R_E + IR_B)$, or equivalently

$$R_{EE} - R_{BB} + I(R_{EB} + R_{BE}) \ , \tag{58}$$

which is a complex linear combination of the four $3 \times 3$ blocks of the Riemann matrix (57). However, it turns out that the complex symmetric $3 \times 3$ matrix (58) encodes only part of the Riemann curvature tensor, namely the Weyl tensor. More specifically, the Riemann curvature tensor decomposes into a trace part, the Ricci tensor $R_{km}$, and a totally traceless part, the Weyl tensor $C_{klmn}$. The Ricci tensor, which is symmetric, has 10 independent components. The Weyl tensor, which inherits the symmetries (55) and (56) of the Rieman tensor, and in addition vanishes on contraction of any pair of indices, also has 10 independent components. Together, the Ricci and Weyl tensors account for the 20 components of the Riemann tensor. The components of the Ricci and Weyl tensors, though algebraically independent, are related by the differential Bianchi identities.

The end result is that the Weyl tensor, the traceless part of the Riemann curvature tensor, can be written as a $3 \times 3$ complex traceless symmetric matrix (58). Such a matrix has 5 distinct complex components.

In empty space (vanishing energy-momentum tensor), the Ricci tensor vanishes identically. Thus the properties of the gravitational field in empty space are specified entirely by the Weyl tensor. In particular, gravitational waves are specified entirely by the Weyl tensor.

When the 5 complex components of the Weyl tensor are expressed in a NP null tetrad (40), the result is 5 complex components, of spins respectively $-2$, $-1$, 0, $+1$, and $+2$:

$$\begin{aligned} -2 &: C_{u-u-} \\ -1 &: C_{uvu-} \\ 0 &: \frac{1}{2}\left(C_{vuvu} + C_{vu-+}\right) \\ +1 &: C_{vuv+} \qquad (59) \\ +2 &: C_{v+v+} \ . \qquad (60) \end{aligned}$$

It can be shown that these 5 complex components exhaust the degrees of freedom of the Weyl tensor.

For outgoing gravitational waves, only the spin $-2$ component propagates, carrying gravitational waves to far dis-

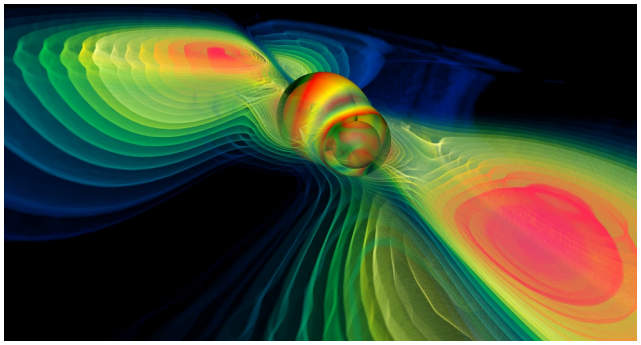

**Figure 5: Volume rendering of the gravitational radiation during a binary black hole merger, represented by the real part of Weyl scalar $r \cdot \psi_4$.**

tances:

$$-2 \ : \ \text{propagating, outgoing .} \tag{61}$$

This propagating, outgoing $-2$ component has spin $-2$, but its complex conjugate has spin $+2$, so effectively both spin components, or helicities, or polarizations, of an outgoing wave gravitational wave are embodied in the single complex component. The remaining 4 complex NP components (spins $-1$ to 2) of an outgoing gravitational waves are short range, describing the gravitational field near the source.

Conventionally (Chandrasekhar 1983), the 5 complex spin components of the Weyl tensor in the NP formalism are impenetrably denoted

$$\begin{aligned} -2 &: \psi_4 \ , \\ -1 &: \psi_3 \ , \\ 0 &: \psi_2 \ , \\ +1 &: \psi_1 \\ +2 &: \psi_0 \ . \end{aligned} \tag{62}$$

Thus the component $\psi_4$ represents propagating, outgoing gravitational waves. The real part of $\psi_4$ represents the $\cos(2\chi)$, or $+$, polarization of the propagating gravitational wave, while (minus) its imaginary part represents the $\sin(2\chi)$, or $\times$, polarization, Figure 4. Next time you see an illustration of gravitational waves where the caption says that $\psi_4$ is plotted, that's what it is (see figure 5). We consider the formulation of the NP scalars as presented here much easier to understand than the usual approach, such as e.g. [18].

# 4. IMPLEMENTING VECTORS IN C++

As demonstrated in section 2, denoting a vector by just its dimensionality $n$ is insufficient to completely identify its algebraic properties including coordinate transformation rules. Additional information is needed, such as the number of covariant and contra-variance indices.

## 4.1 Class Hierarchy

Let us denote an array of fixed size $N$ over some type $T$ as `FixedArray<T,N>`, using C++ template notation. No algebraic operation shall be defined on this type, it just serves as a container for numbers, forming an $N$-tupel of $T$'s. This definition serves as a base class for a type `Vector<T,N>`, which does not add new data members but only adds operators for addition of `Vector<T,N>`'s and multiplication with scalar values, yielding objects of type `Vector<T,N>` again.

$$\texttt{FixedArray<T,N>} \rightarrow \texttt{Vector<T,N>} \tag{63}$$

The resulting class `Vector<T,N>` is a vector in the algebraic sense. It is convenient to make use of matrix algebra in many cases, and since matrices have vector space properties, to express such by deriving the `Matrix` class from the general `Vector` class:

$$\texttt{Vector<T,N*M>} \rightarrow \texttt{Matrix<T,N,M>} \tag{64}$$

The matrix class will add the concept of a matrix product to the general vector space elements. A convenient, though not required, intermediate definition is to define rows and columns – they are rather type definitions than derived classes:

$$\texttt{Matrix<T,1,M>} \rightarrow \texttt{Row<T,M>} \tag{65}$$

$$\texttt{Matrix<T,N,1>} \rightarrow \texttt{Column<T,N>} \tag{66}$$

These definitions of provide a the basis of vector types to be used on the tangential space of a manifold. For a given $N$,$T$ the following classes are derived:

$$\texttt{FixedArray<T,N>} \rightarrow \texttt{point} \tag{67}$$

$$\texttt{Row<T,N>} \rightarrow \texttt{covector} \tag{68}$$

$$\texttt{Column<T,N>} \rightarrow \texttt{tvector} \tag{69}$$

$$\texttt{Vector} < \texttt{T}, \texttt{N}^2 - \texttt{N}(\texttt{N}+1)/2 > \rightarrow \texttt{bivector} \tag{70}$$

$$\texttt{Vector} < \texttt{T}, 1 + \texttt{N}^2 - \texttt{N}(\texttt{N}+1)/2 > \rightarrow \texttt{rotor} \tag{71}$$

$$\texttt{Vector} < \texttt{T}, 2^{\texttt{N}} > \rightarrow \texttt{mulvector} \tag{72}$$

The definition of (68) and (69) directly implements the duality relationship (6) in a type-safe way. Tangential vectors and co-vectors both have vector space properties by virtue of (64), but are different types, yet with the property that their product (inherited from the matrix product) yields a scalar. A `point` (67) by itself has no algebraic properties, it only provides coordinates. However, the difference between two points is to be defined to yield a tangential vector (69). On `tvector`s and `covector`s usual matrix operations are inherently defined, so existing algorithms – that are usually provided using matrix algebra – can still be applied to them. However, objects that directly implement operations from Geometric Algebra such as `bivector`, `rotor` and `multivector` are safe from being used as parameters to matrix algebra, yet they inherit vector space properties. We can not show the actual implementation of the operations here due to space limitations; it is sufficient to emphasize that, by using C++ operator overloading, the API can be made very close to the mathematical notation. In addition it is convenient to overload the function call operator "`()`" for `rotor` objects to denote them to be applied to a vector object, meaning "$R(v)$" $:= RvR^{-1}$. This operator will be used in the following code excerpts.

## 4.2 Camera Navigation using GA

A "camera" in the Vish [8] visualization framework is defined through an observer's location $P$, a point that is looked at $L$, and an horizontal view plane, which is given as a bi-vector $U$ corresponding to the "upwards" direction. The difference $t = L - P$ gives the view direction, a tangential vector.

One algorithm for camera navigation is to rotate the camera by an angle $\varphi$ horizontally around the point of interest $L$ and by an angle $\vartheta$ "upwards" along the line of sight. This algorithm is easily expressed in terms of geometric algebra. First we define the view plane $V$ as

$$V := t \wedge *U \tag{73}$$

and then construct two rotors, a horizontal one and a vertical one

$$R_H := e^{U/|U|\ \varphi} \tag{74}$$

$$R_V := e^{\ V/|V|\ \vartheta} \tag{75}$$

Now the camera motion is achieved by computing the new observer location by adding the rotated view direction to the point of interest:

$$P_{new} = L + (R_H R_V)\,(t) \tag{76}$$

Finally, the horizontal view plane needs to be adjusted as well by the vertical rotation

$$U_{new} = R_V U \tag{77}$$

This algorithm can directly be implemented in six C++ source code statements:

```
void Rotate(Camera&TheCamera,
            double phi, double theta)
{
tvector t = TheCamera.Observer - TheCamera.LookAt;

bivector VerticalPlane = (t ^ *TheCamera.Up).unit();

rotor HorizontalRotation = exp(TheCamera.Up , phi),
      VerticalRotation   = exp(VerticalPlane, theta);

      TheCamera.Up *= VerticalRotation;

      TheCamera.Observer = TheCamera.LookAt +
         (VerticalRotation*HorizontalRotation)( t );
}
```

Another algorithm will rotate the camera around the view direction. This is trivial to implement, since we just need the rotor $R_t$ that corresponds to the view direction, which is given by the exponential of from the dual of the sight vector (a bi-vector),

$$R_t = e^{\varphi *(P-L)/|P-L|} \quad , \tag{78}$$

and apply this to the camera's Up-bivector to rotate it. The corresponding C++ source code is accordingly simple:

```
double RotateAroundViewdir(Camera&theCamera, double phi)
{
tvector t = (Camera.Observer - Camera.LookAt).unit();
rotor ViewRotor = exp(*t, phi);
      Camera.Up = ViewRotor( Camera.Up );
}
```

This formulation is considered to be much simpler than an equivalent formulation using matrices and objects like "axial vectors". Using the operations and involved objects is very intuitive once their meaning in the Geometric Algebra has become clear.

## 4.3 Relativistic observers in the BHFS

### *4.3.1 The BHFS*

The Black Hole Flight Simulator (BHFS) is general relativistic software that can be used to visualize black holes. The BHFS remains work in progress, but has already been used in a number of productions, including the large-format high-resolution dome show "Black Holes: The Other Side of Infinity" (2006, Denver Museum of Nature and Science), and the TV documentaries "Monster of the Milky Way" (2006, NOVA-PBS), and "Monster Black Hole" (2008, Naked Science series, National Geographic). Figure 6 illustrates three frames from a sequence rendered for the National Geographic documentary.

The BHFS provides a complete implementation of the Reissner-Nordström geometry of a charged black hole, including its analytic connections inside the horizon to wormholes, white holes, and other universes. Real astronomical black holes probably have little charge, but they probably do rotate rapidly. A charged black hole is often taken as a surrogate for a rotating black hole, since the interior structure of a spherical charged black hole resembles that of a rotating black hole, but is much easier to model.

The Reissner-Nordström geometry, like its rotating counterpart the Kerr-Newman geometry, is subject to the relativistic counter-streaming instability at the inner horizon first pointed out by Poisson & Israel (1990) [17], and called by them "mass inflation" (see Hamilton & Avelino 2009 [12] for a review). The inflationary instability is expected to eliminate the wormhole and white hole connections inside realistic (astronomical) black holes.

### *4.3.2 Lorentz rotors in the BHFS*

In addition to volume-rendering, the BHFS implements quasi-rigid objects, called "Ships", which by default move along geodesics in the black hole geometry. The camera (observer) is attached to one of the Ships. The orientation and motion of the camera are defined by a Lorentz transformation (which includes both a spatial rotation and a Lorentz boost), or equivalently, by a Lorentz rotor.

A Lorentz rotor $R$ is a unimodular member of the even elements of the spacetime algebra. A Lorentz rotor can be written

$$R = e^{\theta} \tag{79}$$

where $\theta$ is a bivector in the spacetime algebra. The corresponding inverse Lorentz rotor is the reverse $\bar{R}$

$$\bar{R} = e^{-\theta}\ . \tag{80}$$

The condition of being unimodular means $\bar{R}R = 1$.

The even spacetime algebra is isomorphic to the algebra of complex quaternions, also called biquaternions. A complex quaternion can be written

$$q = q_R + I q_I \tag{81}$$

where $q_R$ and $q_I$ are two real quaternions comprising the real and imaginary parts of the complex quaternion $q$

$$q_R = i x_R + j y_R + k z_R + w_R\ ,\ \ q_I = i x_I + j y_I + k z_I + w_I\ . \tag{82}$$

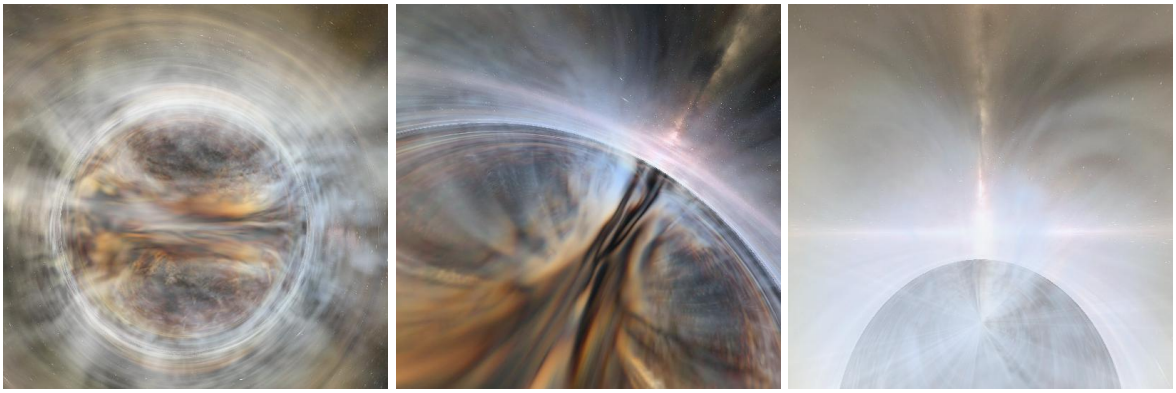

**Figure 6: Three frames from a 3000-frame general relativistic volume-rendering with the BHFS of a general relativistic magnetohydrodynamic supercomputer simulation of a disk and jet around a black hole (John Hawley, 2007, private communication). The three frames show, from left to right, (a) outside the black hole, (b) passing through the black hole's outer horizon, (c) hitting the black hole's inner horizon, where the infinite blueshift and energy density triggers the mass inflation instability (Poisson & Israel 1990). The background texture was created from a 3D model of the Milky Way by Donna Cox's team at NCSA. The sequence was prepared for "Monster Black Hole", an episode of National Geographic's Naked Science series.**

The imaginary $I$ is the pseudoscalar of the spacetime algebra. It commutes with the quaternionic imaginaries $i$, $j$, $k$. The quaternionic imaginaries themselves satisfy

$$i^2 = j^2 = k^2 = -1 \ , \quad ijk = 1 \ , \tag{83}$$

from which it follows that the quaternionic imaginaries anticommute between each other, for example $ij = -k = ji$. The convention $ijk = 1$, equation (83), agrees with the convention for quaternions in OpenGL, but is opposite to William Rowan Hamilton's carved-in-stone convention $ijk = -1$. In OpenGL, rotations accumulate to the right: a rotation $R = R_1 R_2$ means rotation $R_1$ followed by rotation $R_2$.

The BHFS stores a complex quaternion $q$ as an 8-component object

$$q = \begin{pmatrix} x_R & y_R & z_R & w_R \\ x_I & y_I & z_I & w_I \end{pmatrix} \ . \tag{84}$$

The reverse $\bar{q}$ of the complex quaternion $q$ is its quaternionic conjugate

$$\bar{q} = \begin{pmatrix} -x_R & -y_R & -z_R & w_R \\ -x_I & -y_I & -z_I & w_I \end{pmatrix} \ . \tag{85}$$

The group of Lorentz transformations, or Lorentz rotors, corresponds to complex quaternions of unit modulus. The unimodular condition $\bar{R}R = 1$, a complex condition, removes 2 degrees of freedom from the 8 degrees of freedom of complex quaternions, leaving the Lorentz group with 6 degrees of freedom, which is as it should be.

Spatial rotations correspond to real unimodular quaternions, and account for 3 of the 6 degrees of freedom of Lorentz transformations. A spatial rotation by angle $\theta$ right-handedly about the $x$-axis is the real Lorentz rotor

$$R = \cos(\theta/2) + i \sin(\theta/2) \ , \tag{86}$$

or, stored as a complex quaternion,

$$R = \begin{pmatrix} \sin(\theta/2) & 0 & 0 & \cos(\theta/2) \\ 0 & 0 & 0 & 0 \end{pmatrix} \ . \tag{87}$$

Lorentz boosts account for the remaining 3 of the 6 degrees of freedom of Lorentz transformations. A Lorentz boost by velocity $v$, or equivalently by boost angle $\theta = \text{atanh}(v)$, along the $x$-axis is the complex Lorentz rotor

$$R = \cosh(\theta/2) + Ii \sinh(\theta/2) \ , \tag{88}$$

or, stored as a complex quaternion,

$$R = \begin{pmatrix} 0 & 0 & 0 & \cosh(\theta/2) \\ \sinh(\theta/2) & 0 & 0 & 0 \end{pmatrix} \ . \tag{89}$$

### 4.3.3 *Simplicity of Lorentz rotors*

The advantages of quaternions for implementing spatial rotations are well-known to 3D game programmers. Compared to standard rotation matrices, quaternions offer increased speed and require less storage, and their algebraic properties simplify interpolation and splining.

Complex quaternions retain similar advantages for implementing Lorentz transformations. They are fast, compact, and straightforward to interpolate or spline.

Under a spacetime rotation by Lorentz rotor $R$, a general multivector $a$ in the spacetime algebra transforms as

$$a \to \bar{R} a R \ . \tag{90}$$

A general such multivector in the spacetime algebra is a 16-component object, with 8 even components, and 8 odd components.

As remarked earlier, the 8-component even spacetime subalgebra is isormorphic to the algebra of complex quaternions. As an example, the electromagnetic field constitutes a 6-component bivector, an even element of the spacetime algebra. The electric and magnetic fields $E$ and $B$ can be

encoded as the complex quaternion

$$F = \begin{pmatrix} E_x & E_y & E_z & 0 \\ B_x & B_y & B_z & 0 \end{pmatrix} . \quad (91)$$

The transformation (90) then becomes

$$F \to \bar{R} F R , \quad (92)$$

which is a powerful and elegant way to Lorentz transform the electromagnetic field. The electromagnetic field $F$ in the transformation (92) is the complex quaternion (91), and the rotor $R$ is another complex quaternion, so the Lorentz transformation (92) amounts to multiplying 3 complex quaternions, a one-line expression in a c++ program.

The most common need in the BHFS is to Lorentz transform odd multivectors, not even multivectors. For example, every point on a scene that an observer sees is represented by the energy-momentum 4-vector of a photon emitted by the point and observed by the observer. Each such 4-vector $a = a^m \gamma_m$ is an odd multivector in the spacetime algebra. A general odd multivector is a sum of a vector part $a$ and a pseudovector part $Ib$. The odd multivector can be written as a product of $\gamma_t$ (the time basis element of the spacetime algebra) and an even multivector $q$

$$a + Ib = \gamma_t q \quad (93)$$

where $q$ is the even multivector, or complex quaternion,

$$q = \begin{pmatrix} -b^x & -b^y & -b^z & a^t \\ a^x & a^y & a^z & b^t \end{pmatrix} . \quad (94)$$

The Lorentz transformation (90) implies $\gamma_t q \to \bar{R} \gamma_t q R = \gamma_t \bar{R}^* q R$, where $^*$ denotes complex conjugation with respect to the peudoscalar imaginary $I$. It follows that the complex quaternion $q$, equation (94), transforms as

$$q \to \bar{R}^* q R . \quad (95)$$

The transformation (95) of the complex quaternion (94) provides a simple and elegant way to Lorentz transform a 4-vector $a^m$ and 4-pseudovector $Ib^m$. Since $b^m$ (without the $I$ factor) is just another 4-vector, the transformation (95) effectively transforms two 4-vectors, $a^m$ and $b^m$, simultaneously. The transformation (95) amounts to multiplying 3 complex quaternions, a one-line expression in a c++ program.

# 5. VECTORS ON THE HARD DISK

## 5.1 Meta-Data on Vector Types

Storing a specific vector on hard disk, entails storing its numerical representation in a chosen coordinate system. However, when reading an unkown object from disk, solely the information on its numerical representation is insufficient to know what kind of vector it might be. We need some meta-data, additional information about the data itself, that tells what properties the object on disk has.

Within a C++ program, this meta-information is available via the `typeid` function of a type. For instance, it allows to distinguish between a `FixedArray<3,double>` and a `Vector<3,double>`, because `typeid(FixedArray<3,double>) != Vector<3,double>`, even though the memory layout of both types is exactly the same. However, the function value of `typeid` cannot be stored to disk – it is a compiler-internal property that makes only sense at runtime for this specific compiler.

We therefore need to assign certain properties to a type that are associated with its algebraic properties. These properties must not be stored with the vector type itself for performance reasons. They could be stored within a class as `enum`s, `typedef`s or static member functions, thereby not requiring memory for the actual numerical type. An alternative technique is to associate information to a type via C++ type trait templates. This technique, common in C++ template meta-programming [21], allows to specify information about a type independently from this type, thereby achieving some encapsulation between the original type and the meta-information about it. Type traits are templates that are specialized for known types and provide information on these types without the need to modify the type itself. They can be applied to native types as well as user-defined types, and including to types that are defined externally, for example by a library. They can be added independently to an existing type. An example of a type trait definition is given in the following code excerpt:

```
template <class Type> struct MetaInfo;

template <>
struct MetaInfo<double>
{   enum   { SIZE = 1 } };

template <int N>
struct MetaInfo<FixedArray<N, double> >
{   enum   { SIZE = N }   };
```

The type trait `MetaInfo` associates an integer value `SIZE` with an arbitrary type `Type`. This information is available at compile-time, and can be reduced to an usual integer in a template class at any time, such as in:

```
template <class Type>
int NumberOfElements(const Type&T)
{
   return MetaInfo<T>::SIZE;
}
```

Note that a type trait class may also specify default values (by specifying a non-specialized definition) and can be functions on template types itself (as demonstrated in the second specialization). This mechanism allows to equip existing types, e.g. as provided by external libraries, with meta-information as required for our framework.

The objective is to specify complete meta-information about a "vector space element" as required to uniquely identify it. As introduced in section 2, such information includes a reference to the metric (or metric field) and the orientation form $\iota$, to know perform the correct algebraic operations. This information can be provided via a "coordinate system", which can be a global type definition – not more than providing the implicit knowledge on how to perform these operations, such as in Euclidean space. In such a case, no memory or computational resources are implied, but another type definition could require explicit formulae for expressions that

are implicit in Euclidean space. Such a chart object may be expressed via a convention on how the coordinate functions are named, for instance $\{x, y, z\}$ for Cartesian coordinates versus $\{r, \vartheta, \varphi\}$ for polar coordinates. While this is yet work in progress, the following quantities have been found to be required for at least basic distinction and identification of vector types:

I *multiplicity*: an integer value expressing the number of components of this type.

II *rank*: the power $k = a + b$ of the vector space in terms of the tangential space $T^a(M) \times (T^*)^b(M)$; it is the dimensionality of the index space when considering the vector type as an array: zero indicates a scalar type, one is a one-dimensional vectorial type (tangential vector, co-vector, pseudo-vector, pseudo-covector), two are objects representable as matrix, etc.

III *grade*: for quantities from geometric algebra, specifies the grade $k$ of the $k$-vector; the default is zero, for instance for symmetric tensor fields. For example, a bivector in 3D will have a grade of 2 whereas its rank is 1.

IV *dimensions*: the dimensionality $n$ of the $n$-dimensional manifold on which this vector type is attached.

V *coordinatename(i)*: textual functions specifying the naming convention for each of the $n$ coordinate functions.

VI *covariance(i)*: for each index, a flag specifying whether the index is an upper index or lower index. It can be implemented via some function that returns true or false for each index; this function may be evaluated fully at compile-time (a template function that is known) or via lookup into some static array.

VII *symmetries(n)*: often, tensors have symmetric or antisymmetric index pairs. For efficiency reasons it is then important to calculate and store only a minimum subset of the components. This can be implemented via two lookup tables: one table lists those components which are actually stored, the other table contains the prescription for obtaining each tensor component. In a simple scheme, each tensor component is either stored, or is the negative of a stored component, or is zero. (See tables 1 and 2 for examples.) More complex schemes also allow cyclic symmetries, where tensor components can be linear combinations of stored components.

VIII *coordinate systems(i)*: tensor components are only defined with respect to a particular coordinate system. It is necessary to store (for each index) the name of the associated coordinate system. There are objects, such as basis systems or operators that transform between different coordinate systems, where different tensor indices correspond to different coordinate systems.

These properties have been chosen such that some operations on the given types can also succeed with partial knowledge, since certain algorithms do not require full knowledge of the entire algebraic operations of all types.

List of stored components mapping the component name to each storage index:

| [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|
| $g_{xx}$ | $g_{xy}$ | $g_{xz}$ | $g_{yy}$ | $g_{yz}$ | $g_{zz}$ |

Obtaining tensor components from stored components via prescription for each entry:

| $g_{xx}$ | $g_{xy}$ | $g_{xz}$ | $g_{yx}$ | $g_{yy}$ | $g_{yz}$ | $g_{zx}$ | $g_{zy}$ | $g_{zz}$ |
|---|---|---|---|---|---|---|---|---|
| $+[0]$ | $+[1]$ | $+[2]$ | $+[1]$ | $+[3]$ | $+[4]$ | $+[2]$ | $+[4]$ | $+[5]$ |

**Table 1: Storing a symmetric $3 \times 3$ tensor: The component table works like a pointer to the stored components.**

List of stored components, mapping the component name to each storage index:

| [0] | [1] | [2] |
|---|---|---|
| $B_{xy}$ | $B_{xz}$ | $B_{yz}$ |

Obtaining tensor components from stored components via prescription for each entry:

| $B_{xx}$ | $B_{xy}$ | $B_{xz}$ | $B_{yx}$ | $B_{yy}$ | $B_{yz}$ | $B_{zx}$ | $B_{zy}$ | $B_{zz}$ |
|---|---|---|---|---|---|---|---|---|
| 0 | $+[0]$ | $+[1]$ | $-[0]$ | 0 | $+[2]$ | $-[1]$ | $-[2]$ | 0 |

**Table 2: Storing an antisymmetric $3 \times 3$ tensor: The component table defines also signs during dereferencing, or in general, a polynomial expression of components.**

This list of "vector properties" is not claimed to be complete; it is an early attempt to find a comprehensive scheme to cover all geometric and algebraic quantities that occur when performing numerical computations on manifolds. Special attention must also be given to the case of non-tensorial quantities such as Christoffel symbols, which do not yet fit into this ontology.

The Cactus framework [11, 3] currently uses a scheme that is simpler than the above; it is based on tensor algebra only and does not support *grades*. However, it does offer support for tensor densities (by associating a *weight* with each quantity), and it handles also certain special non-tensorial objects, such as logarithms of scalar densities and Christoffel symbols. These special cases are handled as exceptions; there is no generic scheme for them. This scheme is mostly used for symmetry conditions, which require either reflecting (mirroring) or rotating tensors. These operations require only the *symmetry* information above.

What is left is a sufficiently powerful I/O layer that allows to store and retrieve this meta-information persistently on disk, such that a set of pure numbers can be identified for their algebraic properties.

## 5.2 Storing Vector Types in HDF5

HDF5[19] is a generic scientific data format with supporting software, primarily an API provided in C. An HDF5

file can be viewed as a container, in which data objects are organized in ways that are meaningful and convenient to an application. HDF5 can be seen as a framework, rather than a specific format itself, allowing adaption to the various needs of diverse scientific domains [10]. The basic HDF5 object model is relatively simple, yet extremely versatile in terms of the types of data that it can store. The model contains two primary objects: groups, and datasets. Groups provide the organizing structures, and datasets are the basic storage structures. HDF5 groups and datasets may also have associated attributes, which are small data objects for storing metadata defined by applications.

HDF5 allows the specification of user-defined types that shall be stored in a file via its H5T API [20]. For instance, a struct in C/C++ of the form

```
struct CartesianVector
{
    double x,y,z;
};
```

can be expressed in the H5T API as *compound type*:

```
hid_t id = H5Tcreate(H5T_COMPOUND,
                     sizeof(CartesianVector) );
H5Tinsert( id, "x", 0, H5T_DOUBLE);
H5Tinsert( id, "y",   sizeof(double), H5T_DOUBLE);
H5Tinsert( id, "z", 2*sizeof(double), H5T_DOUBLE);
```

This code fragment creates an HDF5 identifier `id` that represents a type of the memory layout as in the aforementioned structure definition. This functionality provides an implementation of the component storage indices as used in table 1 and 2. More details can be found in the HDF5 reference manual.

When writing or reading a dataset to disk, the HDF5 API requires a type identifier to be specified with a `void*`. This tells the HDF5 library how to interpret some chunk of memory. Various generic tools exist to investigate the contents of an HDF5 file, which has a structure of a file system itself. "Datasets" play the role of a file, "Groups" the role of a directory. The tool `h5ls` – part of the HDF5 distribution – lists the contents of an HDF5 file in the fashion of the Unix tool `ls`, enhanced with additional information about the *type* of a dataset. The following example shows how a three-dimensional dataset `CartesianVector data[5][13][9];` appears in this file listing (shortened as compared with actual output):

```
/Block00001 Dataset {5/5, 13/13, 9/9}
    Location:  1:15768
    Links:     1
    Storage:   7020 allocated bytes
    Type:      struct {
                   "x"      +0    native float
                   "y"      +4    native float
                   "z"      +8    native float
               } 12 bytes
    Data:
      (0,0,0) {0.210951, -0.0406732, 0.0611351},
              {0.210204, -0.0443333, 0.0611199},
              {0.209324, -0.0483009, 0.0611070},
              {0.208286, -0.0525892, 0.0610958},
      (0,0,4) {0.207065, -0.0571980, 0.0610863},
              {0.205640, -0.0621138, 0.0610815},
```

By virtue of HDF5, we can easily attach *names* to the purely numerical values in the data field. Hereby the HDF5 library offers various features that are very useful in practice, such as not only taking care of conversions between big-endian and little-endian platforms, but also conversions from double to float component types as well as transformations between different layouts such as $\{x, y, z\} \Leftrightarrow \{z, x, y\}$.

The availability of a naming scheme attached to numerical values is already sufficient to identify a coordinate system that is supposed to be "attached" to these numbers, in spirit of 5.1, V. Knowing the coordinate system relative to which the numbers are stored, in addition we need to specify the various attributes defining the algebraic properties of this vector type HDF5 allows to attach attributes with a dataset, group or "named data type". A named data type is a type id that was created by the `H5Tcreate()` call but saved to disk. It needs to be associated with a group in the file. Attributes attached to such a named data type are shared among all data sets of this type – the data type acts like a pointer to a common location of a set of attributes. We now need to define an HDF5 type for each of the vector types as defined from the meta-information about a specific data type. The following HDF5 listing shows the created named type, stored in a group `/Charts/Cartesian3D`, as it is named "Point" and equipped with an integer telling this data type refers to a manifold of dimension three. This data type "Point" is then later used to declare a dataset of points (shown with two attributes denoting the name of the associated chart and the dimension of the related manifold):

```
/Charts/Cartesian3D/Point Type
    Attribute: ChartDomain scalar
        Type:  null-terminated ASCII string
        Data:  "Cartesian3D"

    Attribute: Dimensions scalar
        Type:      native int
        Data:  3
    Type:      shared-1:13328 struct {
                   "x"      +0    native float
                   "y"      +4    native float
                   "z"      +8    native float
               } 12 bytes

/Block00001 Dataset {5/5, 13/13, 9/9}
    Location:  1:15768
    Links:     1
    Storage:   7020 allocated bytes
    Type:      { shared-1:13328} struct {
                   "x"      +0    native float
                   "y"      +4    native float
                   "z"      +8    native float
               } 12 bytes
    Data:
```

This scheme allows to identify the dataset named "`Blocks`" as representing Cartesian coordinates of point locations. Accessing the dataset "`Blocks`" during reading, the software

application can easily check for the attributes of the dataset to retrieve its algebraic properties. However, doing so is *optional*. Many applications might not implement the full set of tensor algebra, but might still provide a set of useful operations – such as displaying a dataset numerical as a spreadsheet etc. The information that the dataset consists of three floating point numbers, the only information required for a generic operation such as displaying as spreadsheet, is immediately available, more complex properties require further lookup.

This naming scheme is work in progress and not yet implemented or available in its full generality. Various questions have yet to be addressed, such as a generic naming scheme for types or the specification of multivectors. For the latter, one might utilize the HDF5 feature that a compound type may contain other compound types as well. If such is the appropriate solution here, will be subject of further investigation.

## 5.3 Storing Multi-Vector Types in HDF5

Multivectors are linear combinations of vectors of different basis elements, thereby forming an higher-dimensional space. A similar functionality is achieved using HDF5 by creating compound types from the basic vector types. For instance, given a bivector type in 3D, created by HDF5 API calls of the form

```
hid_t   bivector3D_id   =
       H5Tcreate( H5T_COMPOUND, 3*sizeof(double) );

H5Tinsert( bivector3D_id, "yz",  0, H5T_NATIVE_DOUBLE);
H5Tinsert( bivector3D_id, "zx",  8, H5T_NATIVE_DOUBLE);
H5Tinsert( bivector3D_id, "xy", 16, H5T_NATIVE_DOUBLE);
```

we may create a rotor in the following as compound containing the bivector, and adding a scalar:

```
hid_t   rotor3D_id   = H5Tcreate( H5T_COMPOUND, 32 );

H5Tinsert( rotor3D_id, "cos", 0, H5T_NATIVE_DOUBLE);
H5Tinsert( rotor3D_id, "sin", 8, bivector3D_id);
```

We name the scalar and bivector component "cos" and "sin" here, inspired by the construction of a rotor. What naming scheme to use here in general, will yet need to be explored. It is now a nice feature of HDF5 that different storage schemes are automatically mapped, i.e. datasets stored as the following type

```
hid_t antirotor3D_id =
H5Tcreate( H5T_COMPOUND, 4*sizeof(double) );
H5Tinsert( antirotor3D_id, "sin", 0 , bivector3D_id);
H5Tinsert( antirotor3D_id, "cos", 24, H5T_NATIVE_DOUBLE);
```

can be directly read without further specific treatment as a `rotor3D_id` dataset. This way HDF5 easily provides the notion of $a + c \wedge b \equiv c \wedge b + a$, i.e., commutativity of the "+" operator. One can also define a type which only retrieves the bivector component of a dataset of rotors, or the scalar component. This functionality is already provided by HDF5.

The specification of maps on multivectors, section 2.8, appears non-trivial, due to the many symmetry conditions that occur in these cases. For instance, if the Riemann tensor as in 3.3 would be stored by each of its tensor components, this results in 256 values (at each point). However, only 20 need to be stored, and under certain conditions (such as matter-free spacetime) that may be known in advance, only 10 . A smart type definition system that is able to express such properties yet has to be developed. Symmetry tables such as discussed in 5.1 might be a way to go, and a formulation of those as attributes on HDF5 types will be developed.

# 6. CONCLUSION

In this article we have reviewed the various types of what is usually called a "vector" in the context of differential geometry and geometric algebra. Various algebraic types have been identified, which are all represented numerically by three floating point numbers in three dimensions: tangential vectors, co-vectors, bi-vectors and bi-co-vectors. Yet these four different types have distinct algebraic properties and should be distinguished. We demonstrated the application of diverse vector types in four dimensions, leading to an easier formulation of the Newmann-Penrose formalism by virtue of Geometric Algebra. The clarity of the diverse algebraic types as achieved via GA thereby eases "navigation" in Riemann space, computer graphic applications (where two examples are given), and identification of quantities stored in files.

# 7. REFERENCES


[1] LIGO: Laser Interferometer Gravitational Wave Observatory, URL `http://www.ligo.caltech.edu/`.

[2] GRwiki: a repository of basic definitions and formulas for gravitational physics, URL `http://grwiki.physics.ncsu.edu/`.

[3] Cactus Computational Toolkit home page, URL `http://www.cactuscode.org/`.

[4] Binary pulsar. Wikipedia, 2009. URL `http://en.wikipedia.org/wiki/Binary_pulsar`.

[5] Exterior algebra. Wikipedia, 2009. URL `http://en.wikipedia.org/wiki/Exterior_algebra`.

[6] The Nobel Prize in physics 1993. Wikipedia, 2009. URL `http://nobelprize.org/nobel_prizes/physics/laureates/1993/illpres/discovery.html`.

[7] Quaternion. Wikipedia, 2009. URL `http://en.wikipedia.org/wiki/Quaternion`.

[8] W. Benger, G. Ritter, and R. Heinzl. The Concepts of VISH. In $4^{th}$ *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007*, pages 26–39. Berlin, Lehmanns Media-LOB.de, 2007.

[9] A. Bossavit. Differential geometry for the student of numerical methods in electromagnetism. Technical report, Tampere University of Technology, 1991. URL `http://butler.cc.tut.fi/~bossavit/`.

[10] M. T. Dougherty, M. J. Folk, E. Zadok, H. J. Bernstein, F. C. Bernstein, K. W. Eliceiri, W. Benger, and C. Best. Unifying biological image formats with hdf5. *Communications of the ACM (CACM)*, 52(10):42–47, October 2009.

[11] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In

*Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[12] A. J. S. Hamilton and P. P. Avelino. The physics of the relativistic counter-streaming instability that drives mass inflation inside black holes. *Physics Reports*, accepted, 2009. gr-qc/0811.1926.

[13] A. Held. A formalism for the investigation of algebraically special metrics. i. *Commun. Math. Phys.*, 37:311–26, 1974.

[14] D. Hestenes. *New Foundations for Classical Mechanics, 2nd ed.* Springer Verlag., 1999.

[15] D. Hestenes. Oersted medal lecture 2002: Reforming the mathematical language of physics. *American Journal of Physics*, 71(2):104–121, 2003. URL `http://link.aip.org/link/?AJP/71/104/1`.

[16] E. T. Newman and R. Penrose. An approach to gravitational radiation by a method of spin coefficients. *J. Math. Phys.*, 3:566–79, 1962.

[17] E. Poisson and W. Israel. Inner-horizon instability and mass inflation in black holes. *Phys. Rev.*, D41:1796–1809, 1990.

[18] E. (Ted) Newman and R. Penrose. Spin-coefficient formalism. *Scholarpedia*, 4(6):7445, 2009. URL `http://www.scholarpedia.org/article/Newman-Penrose_formalism`.

[19] The HDF Group. Hierarchical data format version 5. `http://www.hdfgroup.org/HDF5`, 2000-2009.

[20] The HDF Group. HDF5 H5T API, 2009. URL `http://www.hdfgroup.org/HDF5/doc/RM/RM_H5T.html`.

[21] T. Veldhuizen. Using C++ template metaprograms. *C++ Report*, 7(4):36–43, May 1995. Reprinted in C++ Gems, ed. Stanley Lippman.