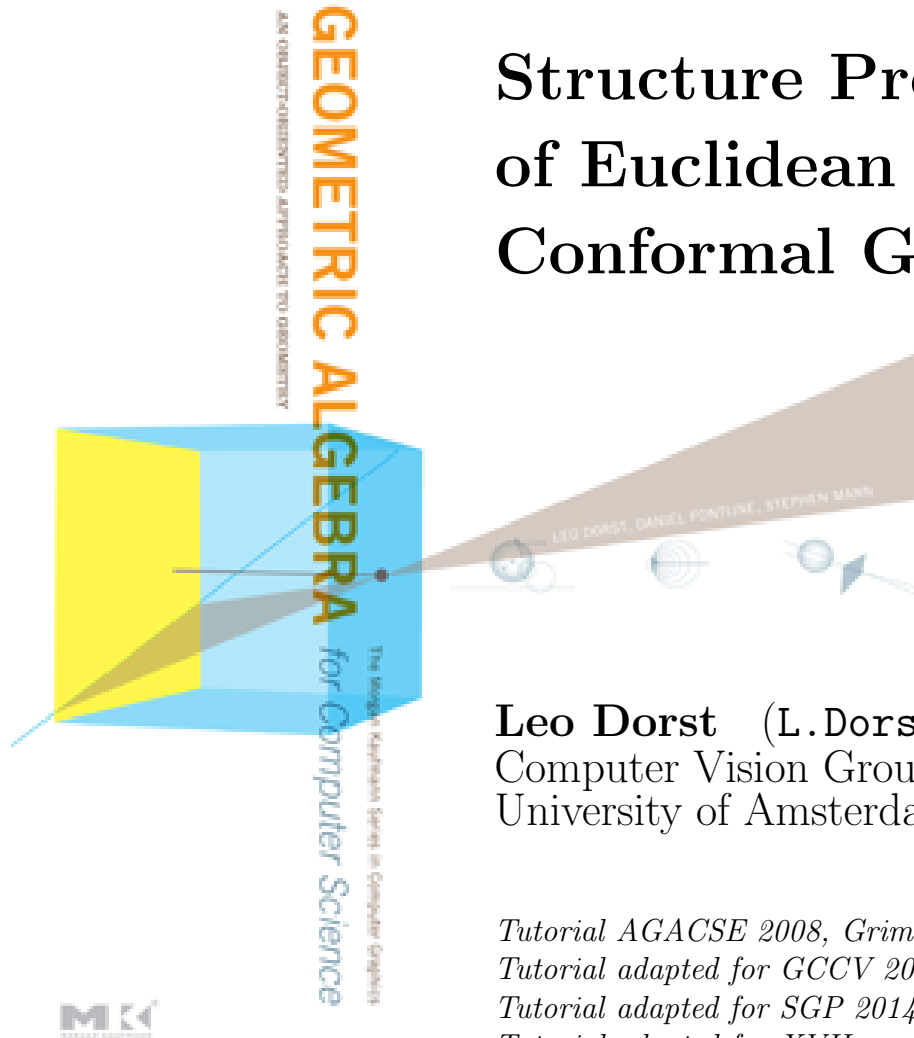


Tutorial:

Structure Preserving Representation of Euclidean Motions through Conformal Geometric Algebra



Leo Dorst (L.Dorst@uva.nl)
Computer Vision Group, Informatics Institute,
University of Amsterdam, The Netherlands

Tutorial AGACSE 2008, Grimma near Leipzig, Germany

Tutorial adapted for GCCV 2013, Guanajuato, Mexico

Tutorial adapted for SGP 2014, Cardiff, Wales

Tutorial adapted for XVII summer school 2016, Santander, Spain

Tutorial adapted for AGACSE 2018, Unicamp, Brazil

1 Geometric Algebra

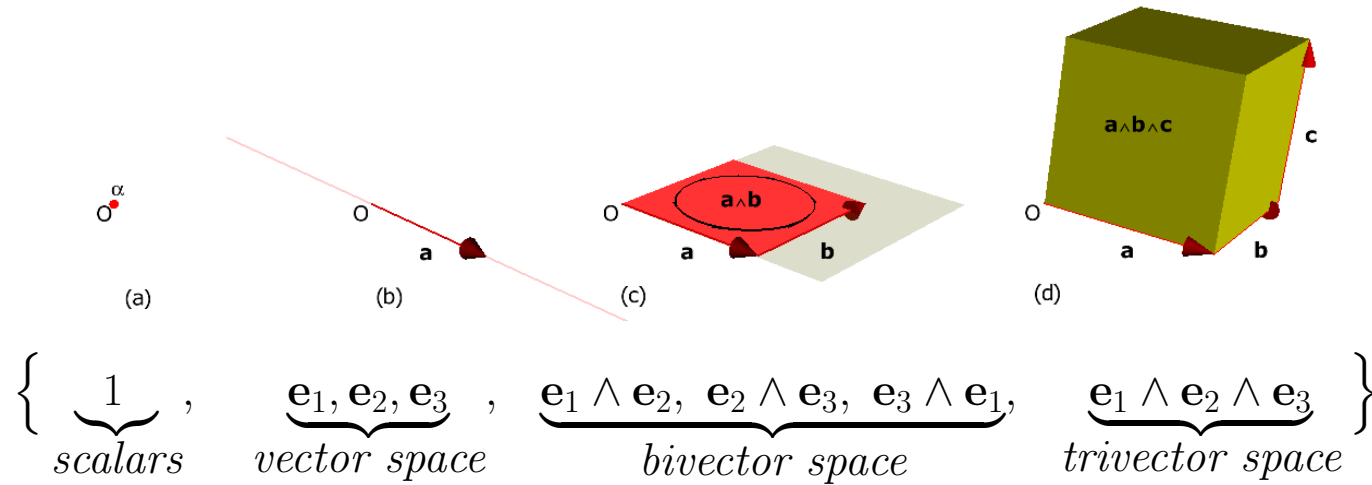
What do we have so far, after the Day 1 and Day 2 Tutorials?

- **Subspaces as computational elements** (by the **outer product** \wedge), basis 2^n for an n -D space.
- **Extended computations**, allowing **division** (of vectors and subspaces) **and duality** (division by the volume element).
- **Non-commutative product** permitting compact treatment of **perpendicularity and parallelness**.
- Compact representation of **orthogonal transformations**, as **rotors** (product of invertible vectors, or exponential of bivector); these act the same on all elements by **sandwiching** (**covariant, structure-preserving**).
- **Geometric calculus**, permitting **differentiation** with respect to all of these elements.
- **Extension of linear mappings** to subspaces (by the **outermorphism**); notably, projection.

Looks great and consistent. Let's do more with it, especially with those **structure-preserving sandwiching rotors**.

2 Beyond Euclidean k -Directions

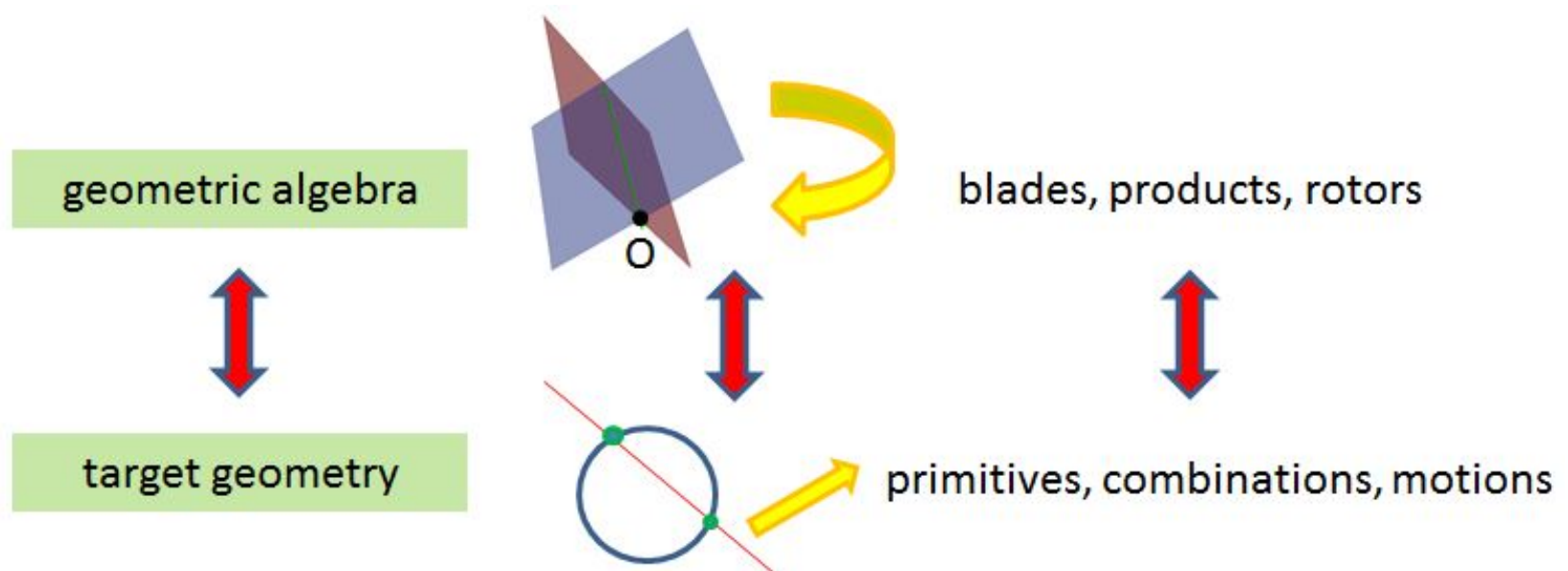
Our conceptual picture so far is fed by a model of directions in Euclidean space.



But the rules and principles of GA [apply to any metric space](#).

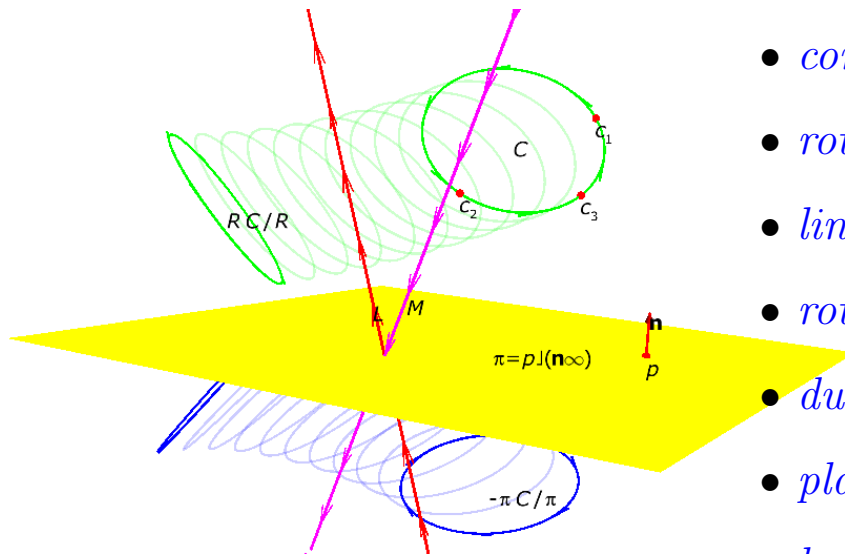
3 GA Modeling Strategy for Geometries

We can design representational spaces for specific geometries, and then use the GA of the representational space to provide a coherent framework for a particular set of transformations and associated primitive objects.



Let us do so for rigid body motions, and spheres as primitive objects.

4 An Example: Reflection of a Rotating Circle (in CGA)



- *construction of a circle*: $C = c_1 \wedge c_2 \wedge c_3$
- *rotation*: $C \mapsto RC/R$
- *line representation*: $L = p_1 \wedge p_2 \wedge n_\infty = p_1 \wedge \mathbf{u} \wedge n_\infty$
- *rotation around line*: $R = \exp(\phi L^*/2)$
- *dual plane representation*: $\pi = p \cdot (\mathbf{n} \wedge n_\infty)$
- *plane reflection*: $X \mapsto -\pi X/\pi$ (for any odd-D X)
- *logarithms of motions*: $R^{1/n} = \exp(\log(R)/n)$

FIG(1,1)

Note that all is specified directly in terms of the geometric elements, and some algebraic operations (\wedge , \cdot , $/$, \exp , \log , $*$, \cdot , fortunately all reducible to one fundamental product).

No coordinates at all in the language (just in the data)!

Most figures from *Geometric Algebra for Computer Science* ©Morgan-Kaufmann 2007,2009.
For the demos: type FIG(i,j) in GAViewer, downloadable at www.geometricalgebra.net.

5 Implementation Matches the Algebra (here Gaigen2)

```
// p1, p2, c1, c2, c3, pt are points
line L; circle C; dualPlane p; vector n;

L = unit_r(p1 ^ p2 ^ ni);
C = c1 ^ c2 ^ c3;
p = pt << (n^ni);

draw(L); draw(C); draw(p);

draw( - p * L * inverse(p) ); // draw reflected line (magenta)
draw( - p * C * inverse(p) ); // draw reflected circle (blue)

// compute rotation versor:
const float phi = (float)(M_PI / 2.0);
TRversor R;
R = exp(0.5f * phi * dual(L));

draw(R * C * inverse(R)); // draw rotated circle (green)
draw(-p * R * C * inverse(R) * inverse(p)); // draw reflected, rotated circle (blue)

// draw interpolated circles
pointPair LR = log(R); // get log of R
for (float alpha = 0; alpha < 1.0; alpha += 0.1f)
{
    TRversor iR;
    iR = exp(alpha * LR); // compute interpolated rotor

    draw(iR * C * inverse(iR)); // draw rotated circle (light green)
    draw(-p * iR * C * inverse(iR) * inverse(p)); // draw reflected, rotated circle (light blue)
}
```

6 By Contrast, the Example in Linear Algebra

- *construction of a circle*: none, resort to treating the points separately.
- *rotation*: by 4×4 homogeneous coordinate matrix $\begin{bmatrix} \mathbf{R} & (\mathbf{I} - \mathbf{R})\mathbf{t} \\ 0^T & 1 \end{bmatrix}$ acting on points $(\mathbf{x}, 1)^T$.
- *line representation*:
 - as (position vector, direction vector)-pair (\mathbf{p}, \mathbf{u}) ; each component moves differently.
 - as the kernel of two homogeneous plane equations: $\llbracket \pi_1, \pi_2 \rrbracket^T$
 - using 6D Plücker coordinates: $\{\mathbf{u}, \mathbf{p} \times \mathbf{u}\}$.
- *rotation around line*: $\llbracket \mathbf{R} \rrbracket = \mathbf{u}\mathbf{u}^T + \cos \phi (\llbracket 1 \rrbracket - \mathbf{u}\mathbf{u}^T) + \sin \phi \llbracket \mathbf{u}^\times \rrbracket$, then move into place.
- *dual plane representation*: $\pi = [\mathbf{n}, -\mathbf{p} \cdot \mathbf{n}]$
- *plane reflection*: Use point reflection $\llbracket \mathbf{P} \rrbracket = \begin{bmatrix} \mathbf{I} - 2\mathbf{nn}^T & 2\delta\mathbf{n} \\ 0^T & 1 \end{bmatrix}$. On planes as $\llbracket \mathbf{P} \rrbracket^{-T}$, on Plücker lines as more involved 6×6 matrix.
- *interpolation of general rotation*: non-elementary (done by specialized logarithm of matrix).

Linear algebra code typically consists of such **coordinate tricks**, applied to the points.

No direct circle rotation, or line reflection, or rotation generation available at basic level.

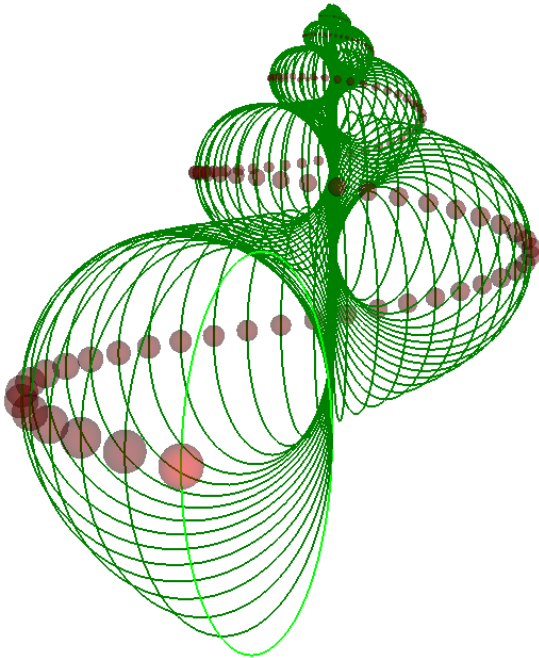
7 The Example in Geometric Algebra of \mathbb{R}^3 : Some Tools, Not Great

- *construction of a circle*: none, resort to treating the points separately.
- *rotation*: in the origin by a rotor; out of origin, **no method** (so worse than homogeneous coordinates!).
- *line representation*: as (position vector, direction vector)-pair (\mathbf{p}, \mathbf{u}) ; each component moves differently.
(But equation $\mathbf{x} \wedge \mathbf{u} = \mathbf{p} \wedge \mathbf{u}$ allows nice algebraic manipulations.)
- *rotation around line*: We do not really have that, but can make a rotor from the direction vector of the line as $\exp(\mathbf{u}^* \phi/2)$.
- *dual plane representation*: by normal vector \mathbf{n} but only at origin.
- *plane reflection*: we can reflect any element X in the origin plane with normal \mathbf{u} by $X \mapsto (-1)^{\text{grade}(X)} \mathbf{u} X \mathbf{u}^{-1}$; but translations need to be handled separately.
- *interpolation of general rotation*: we can interpolate origin rotation by using its logarithm.

The problem is mostly that **translations are not rotors**, meaning we do not have displaced flats as primitives. And circles are composite objects (orbits of points?).

8 Outline: The Six Tricks of Conformal Geometric Algebra

Consider Euclidean geometry not as a specific projective geometry, but as conformal geometry. Embed \mathbb{R}^n isometrically into $\mathbb{R}^{n+1,1}$. Then we get a unification of techniques:



FIG(16,3)

1. Through the isometric embedding, conformal transformations of \mathbb{R}^n are represented as **orthogonal transformations** of $\mathbb{R}^{n+1,1}$.
2. We represent orthogonal transformations as **multiple reflections**, and those using the **geometric product** of Clifford algebra as versors ('spinors'), which **preserve structure**.
3. We automatically get a non-metric **outer product** \wedge as constructor for geometric primitives (points, lines, planes, spheres, circles, tangent vectors, directions etc). This **gives structure**.
4. We automatically get **duality**, providing **quantitative intersections** and a metric **inner product** to do **projections**.
5. Versors as exponentials of bivectors give the **Lie algebra** of motions. Logarithms then permit interpolation.
6. Efficient implementation uses the structural coherence to build a CGA compiler by **automatic code generation**.

9 Trick 1a: Euclidean 3D Point Representation in $\mathbb{R}^{4,1}$ (CGA)

Represent point with 3D Euclidean position vector \mathbf{x} in the 5D Minkowski space $\mathbb{R}^{4,1}$ as a ray vector:

$$x \sim n_o + \mathbf{x} + \frac{1}{2}\mathbf{x}^2 n_\infty$$

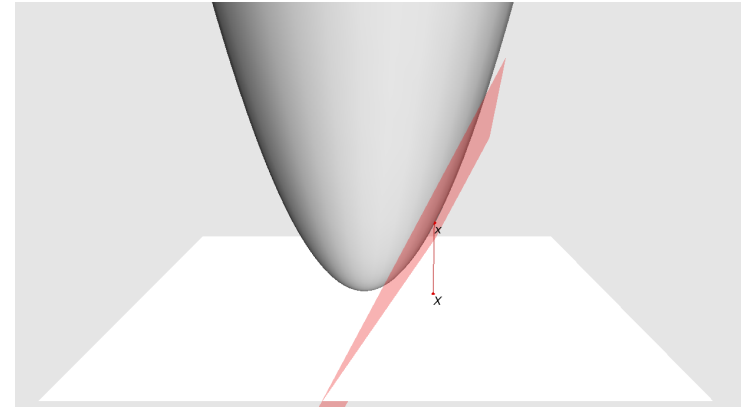
where n_o is the standard point at the origin, \mathbf{x} the Euclidean ‘position vector’, n_∞ is the point at infinity.

Basically, like *two extra homogeneous coordinates*:

$$x \sim (1, \mathbf{x}, \frac{1}{2}\mathbf{x}^2)^T$$

on the 5D basis $\{n_o, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, n_\infty\}$.

This is going to give us the distance between points as a dot product.



A not-so-helpful picture of a CGA point of \mathbb{E}^2 in representational space $\mathbb{R}^{3,1}$

FIG(14,3): point

FIG(14,4): circle

FIG(14,6): circle meet

10 How Can This 2D Up Trick Help?

This is going to give us the distance between points as a dot product ????

$$x \sim n_o + \mathbf{x} + \frac{1}{2}\mathbf{x}^2 n_\infty$$

At first glance, no go. The squares are in the wrong place, with wrong signs:

$$x \cdot y = \begin{bmatrix} 1 \\ \mathbf{x} \\ \frac{1}{2}\mathbf{x}^2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \mathbf{y} \\ \frac{1}{2}\mathbf{y}^2 \end{bmatrix} = \begin{matrix} 1 \\ +\mathbf{x} \cdot \mathbf{y} \\ +\frac{1}{4}\mathbf{x}^2\mathbf{y}^2 \end{matrix} = 1 + \mathbf{x} \cdot \mathbf{y} + \frac{1}{4}\mathbf{x}^2\mathbf{y}^2$$

We can fix it by having the ones and the squares match up, and adding an extra minus sign:

$$\begin{bmatrix} 1 \\ \mathbf{x} \\ \frac{1}{2}\mathbf{x}^2 \end{bmatrix} \cdot \begin{bmatrix} 0 & \mathbf{0} & -1 \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ -1 & \mathbf{0} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{y} \\ \frac{1}{2}\mathbf{y}^2 \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} \\ \frac{1}{2}\mathbf{x}^2 \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{2}\mathbf{y}^2 \\ \mathbf{y} \\ -1 \end{bmatrix} = \begin{matrix} -\frac{1}{2}\mathbf{y}^2 \\ +\mathbf{x} \cdot \mathbf{y} \\ -\frac{1}{2}\mathbf{x}^2 \end{matrix} = -\frac{1}{2}(\mathbf{x} - \mathbf{y})^2$$

So apparently, we need to redefine the dot product.

This amounts to giving the 5D space a handy metric.

11 Trick 1b: Inner Product Represents Squared Euclidean Distance

Metric of the representation space $\mathbb{R}^{4,1}$ is [Minkowski](#). Switch to preferred basis:

\cdot	\mathbf{x}	e_+	e_-
\mathbf{x}	$\ \mathbf{x}\ ^2$	0	0
e_+	0	1	0
e_-	0	0	-1

$$\begin{aligned}
 n_o &= \frac{1}{2}(e_- + e_+) \\
 n_\infty &= e_- - e_+ \\
 &\iff
 \end{aligned}$$

\cdot	n_o	\mathbf{x}	n_∞
n_o	0	0	-1
\mathbf{x}	0	$\ \mathbf{x}\ ^2$	0
n_∞	-1	0	0

Now look what happens between two unit-weight points:

$$\begin{aligned}
 \mathbf{x} \cdot \mathbf{y} &= (n_o + \mathbf{x} + \frac{1}{2}\mathbf{x}^2 n_\infty) \cdot (n_o + \mathbf{y} + \frac{1}{2}\mathbf{y}^2 n_\infty) \\
 &= (0 + 0 - \frac{1}{2}\mathbf{y}^2) + (0 + \mathbf{x} \cdot \mathbf{y} + 0) + (-\frac{1}{2}\mathbf{x}^2 + 0 + 0) \\
 &= -\frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2
 \end{aligned}$$

Weird metric, nice trick: linearization of a squared distance.

The inner product in the representation space gives the squared Euclidean distance!

Therefore, Euclidean motions are represented by [orthogonal transformations](#).

12 Inner Product Represents Squared Euclidean Distance (the Small Print)

Actually, the Euclidean distance corresponds to the inner product of **unit weight vectors** of the form $x = n_o + \mathbf{x} + \frac{1}{2}\mathbf{x}^2 n_\infty$. Any multiple $\alpha\mathbf{x}$ represents a point at the same location, but with a different *weight*. For the general distance formula, we need to normalize the weight to unity:

$$-\frac{1}{2}d_E(P, Q)^2 = \left(\frac{p}{-n_\infty \cdot p} \right) \cdot \left(\frac{q}{-n_\infty \cdot q} \right).$$

So Euclidean motions have to preserve

- the inner product (they are orthogonal transformations),
- the point at infinity n_∞ .

The reason for the name ‘conformal model’ or Conformal Geometric Algebra (CGA) is:

Orthogonal transformations of $\mathbb{R}^{n+1,1}$ represent conformal transformations of \mathbb{R}^n .

Conformal transformations are defined to preserve angles; for example, an isotropic scaling.

That is the context of Euclidean motions in this model. In homogeneous coordinates, the context was **projective transformations**. We don’t have those in the conformal model $\mathbb{R}^{4,1}$ (use $\mathbb{R}^{3,3}$).

13 **Bonus:** Vectors in $\mathbb{R}^{n+1,1}$ Represent Spheres and Planes in \mathbb{R}^n

- general vectors of $\mathbb{R}^{n+1,1}$ are *oriented, weighted (dual) spheres* in \mathbb{R}^n :

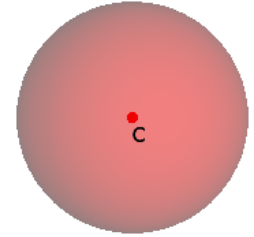
$$d_E^2(X, C) = \rho^2 \Leftrightarrow x \cdot c = -\frac{1}{2}\rho^2 \Leftrightarrow x \cdot (c - \frac{1}{2}\rho^2 n_\infty) = 0$$

Sphere is IPNS of the vector $\sigma = c - \frac{1}{2}\rho^2 n_\infty$.

Squared norm gives radius: $\sigma^2 = (c - \frac{1}{2}\rho^2 n_\infty) \cdot (c - \frac{1}{2}\rho^2 n_\infty) = \rho^2$.

For ‘imaginary’ spheres, this is negative (so they are included!).

Points are (dual) spheres of zero radius.



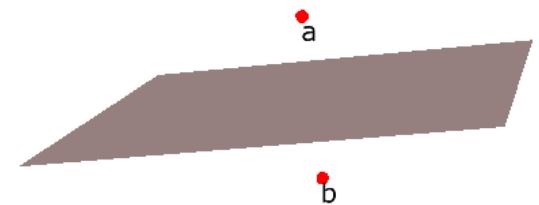
- *oriented, weighted (dual) planes* of \mathbb{R}^n are vectors in $\mathbb{R}^{n+1,1}$ without n_o -component:

$$d_E^2(X, A) = d_E^2(X, B) \Leftrightarrow x \cdot a = x \cdot b \Leftrightarrow x \cdot (a - b) = 0$$

Note that the n_o -component satisfies $-n_\infty \cdot (a - b) = 0$.

(Geometrically, **the point at infinity is on all planes**, or **planes are spheres through infinity**.)

General plane is IPNS of a vector $\pi = \mathbf{n} + \delta n_\infty$, with $\mathbf{n} \in \mathbb{R}^n$. (**Bold will be pure \mathbb{R}^n .**)



14 Trick 2: Geometric Reflections as Algebraic Sandwiching (usual in GAs!)

Reflection in an origin plane with unit normal \mathbf{a}

$$\mathbf{x} \mapsto \mathbf{x} - 2(\mathbf{x} \cdot \mathbf{a}) \mathbf{a} / \|\mathbf{a}\|^2 \quad (\text{classic LA}).$$

Now consider the dot product as the symmetric part of a more fundamental **geometric product**:

$$\mathbf{x} \cdot \mathbf{a} = \frac{1}{2}(\mathbf{x} \mathbf{a} + \mathbf{a} \mathbf{x}).$$

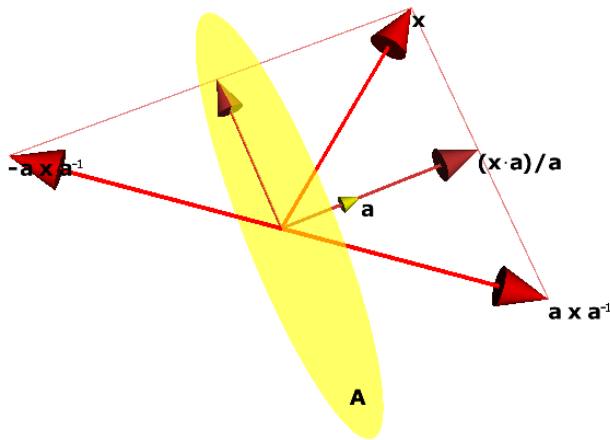
Then rewrite (assuming linearity, associativity):

$$\begin{aligned} \mathbf{x} &\mapsto \mathbf{x} - (\mathbf{x} \mathbf{a} + \mathbf{a} \mathbf{x}) \mathbf{a} / \|\mathbf{a}\|^2 \quad (\text{GA product}) \\ &= \boxed{-\mathbf{a} \mathbf{x} \mathbf{a}^{-1}} \end{aligned}$$

with *the geometric inverse of a vector*: $\mathbf{a}^{-1} = \mathbf{a} / \|\mathbf{a}\|^2$.

Inverse works because

$$\mathbf{a}^{-1} \mathbf{a} = \mathbf{a} \mathbf{a} / \|\mathbf{a}\|^2 = \frac{1}{2}(\mathbf{a} \mathbf{a} + \mathbf{a} \mathbf{a}) / \|\mathbf{a}\|^2 = \mathbf{a} \cdot \mathbf{a} / \|\mathbf{a}\|^2 = 1.$$



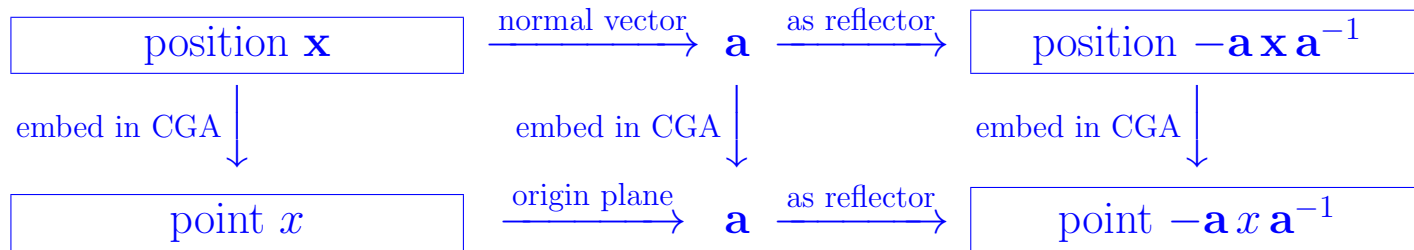
FIG(7,1)

15 (CGA Bonus) Structural Transfer: Reflection Applied to Point

We should reflect the point x itself, rather than merely its Euclidean part \mathbf{x} :

$$\mathbf{x} \mapsto -\mathbf{a}\mathbf{x}\mathbf{a}^{-1}.$$

Try **structural transfer to CGA**:



Use linearity of the geometric product:

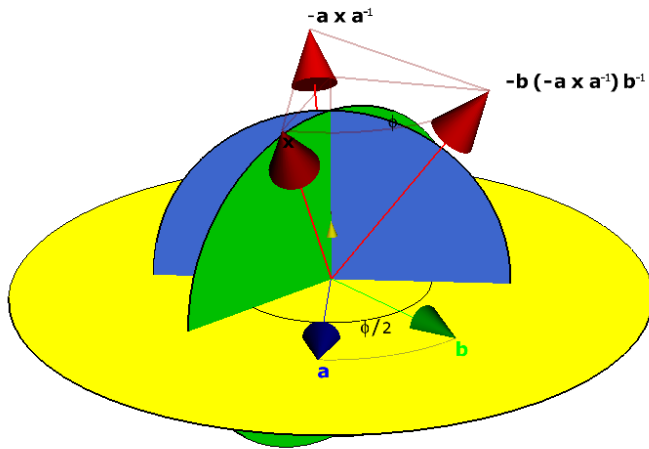
$$\begin{aligned} x &\stackrel{?}{\mapsto} -\mathbf{a}x\mathbf{a}^{-1} \\ &= -\mathbf{a}(n_o + \mathbf{x} + \frac{1}{2}\mathbf{x}^2n_\infty)\mathbf{a}^{-1} \\ &= -\mathbf{a}n_o\mathbf{a}^{-1} - \mathbf{a}\mathbf{x}\mathbf{a}^{-1} - \frac{1}{2}\mathbf{x}^2\mathbf{a}n_\infty\mathbf{a}^{-1}. \end{aligned}$$

Now use $0 = \mathbf{a} \cdot n_o = \frac{1}{2}(\mathbf{a}n_o + n_o\mathbf{a})$ so that $-\mathbf{a}n_o = n_o\mathbf{a}$, same for n_∞ . And $\mathbf{x}^2 = (-\mathbf{a}\mathbf{x}\mathbf{a}^{-1})^2$.

$$\begin{aligned} x &\stackrel{?}{\mapsto} n_o\mathbf{a}\mathbf{a}^{-1} - \mathbf{a}\mathbf{x}\mathbf{a}^{-1} + \frac{1}{2}\mathbf{x}^2n_\infty\mathbf{a}\mathbf{a}^{-1} \\ &= n_o - \mathbf{a}\mathbf{x}\mathbf{a}^{-1} + \frac{1}{2}\mathbf{x}^2n_\infty \\ &\stackrel{!}{=} n_o - \mathbf{a}\mathbf{x}\mathbf{a}^{-1} + \frac{1}{2}(-\mathbf{a}\mathbf{x}\mathbf{a}^{-1})^2n_\infty \end{aligned}$$

So indeed a conformal point at the reflected location. **Automatic in the algebra!**

16 Bonus: Orthogonal Transformations as Versors (usual in GAs!)



FIG(7,2)

A reflection in two successive origin planes **a** and **b**:

$$\begin{aligned} \mathbf{x} &\mapsto -\mathbf{b}(-\mathbf{a} \mathbf{x} \mathbf{a}^{-1}) \mathbf{b}^{-1} \\ &= (\mathbf{b} \mathbf{a}) \mathbf{x} (\mathbf{b} \mathbf{a})^{-1} \end{aligned}$$

So a rotation is represented by the geometric product of two vectors **ba**, also an element of the algebra.

(Actually, in 3D these are quaternions.)

Multiple reflections are the fundamental representation for operators in GA:

The geometric product of (invertible) vectors is called a versor.

It acts as an orthogonal transformation by sandwiching.

As we will see, versors perform structure-preserving actions on all elements.

(It is common to use normalized versors and call those **rotors**. Or do you prefer **spinors**?)

17 Trick 3: Constructing Elements by Anti-Symmetry (usual in GAs!)

Skew-symmetric part of geometric product gives **outer product** (of Grassmann algebra):

$$\mathbf{x} \wedge \mathbf{a} = \frac{1}{2}(\mathbf{x} \mathbf{a} - \mathbf{a} \mathbf{x})$$

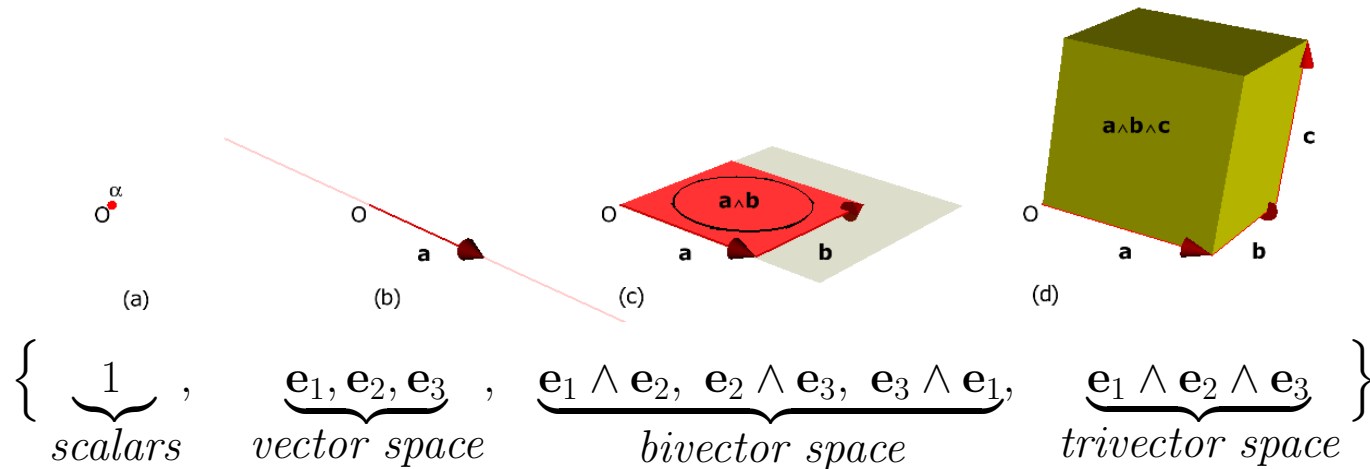
It is bilinear and associative. Use it to span **oriented subspaces** as **blades** (Grassmannians) of various **grades** (dimensionality):

$$\mathbf{x} \wedge (\mathbf{a}_1 \wedge \cdots \wedge \mathbf{a}_k) = 0 \iff \mathbf{x} \text{ in span}(\mathbf{a}_1, \cdots, \mathbf{a}_k)$$

For instance, $\mathbf{x} = \lambda_1 \mathbf{a}_1 + \lambda_2 \mathbf{a}_2$ is a general vector in $\text{span}(\mathbf{a}_1, \mathbf{a}_2)$, and we verify:

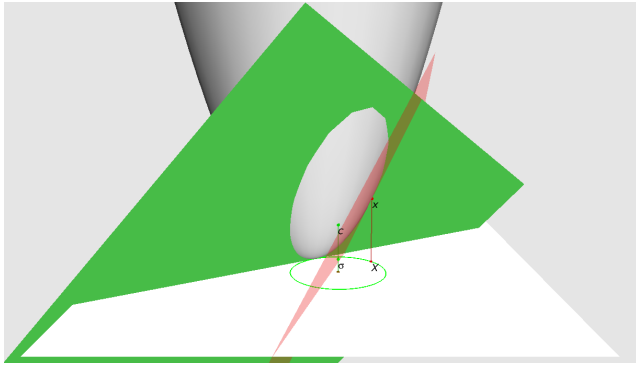
$$\begin{aligned} \mathbf{x} \wedge (\mathbf{a}_1 \wedge \mathbf{a}_2) &= \lambda_1 \mathbf{a}_1 \wedge (\mathbf{a}_1 \wedge \mathbf{a}_2) + \lambda_2 \mathbf{a}_2 \wedge (\mathbf{a}_1 \wedge \mathbf{a}_2) \\ &= \lambda_1 (\mathbf{a}_1 \wedge \mathbf{a}_1) \wedge \mathbf{a}_2 - \lambda_2 \mathbf{a}_1 \wedge (\mathbf{a}_2 \wedge \mathbf{a}_2) = 0 + 0 = 0. \end{aligned}$$

Usual GA visualization (no longer true in CGA!):



18 The Outer Product in the Conformal Model

The basic elements of Euclidean geometry are all represented as \wedge -factorizable multivectors ('blades') in CGA. As usual in GA, they represent proper subspaces in the representational space, but they are interpretable in the Euclidean space by determining the point representatives on them.



Moderately effective visualization
of representational space $\mathbb{R}^{3,1}$ of \mathbb{E}^2

FIG(14,4)

tutorial/DEMOspanning()

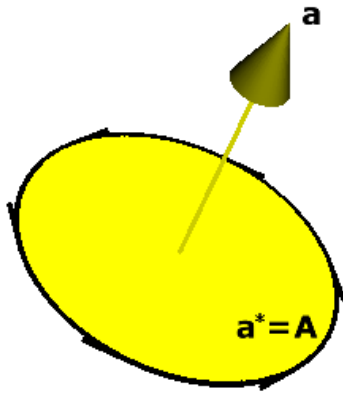
- *Rounds: spheres, circles, point pairs*
 $a \wedge b \wedge c \wedge d$ is an (oriented & weighted) sphere passing orthogonally through 4 given spheres (or points), etc.
- *Flats: planes, lines, flat points*
 $a \wedge b \wedge c \wedge n_\infty$ is an (oriented & weighted) plane passing orthogonally through 3 given spheres (or points), etc.
- *k-dimensional direction elements*
 $\mathbf{B} \wedge n_\infty$ is a pure Euclidean k -space \mathbf{B} , at infinity.
- *k-dimensional tangents*
 $n_o \wedge \mathbf{B}$ is the tangent k -space \mathbf{B} at the origin.
Soon: $p \cdot (p \wedge \mathbf{B} \wedge n_\infty)$ is the tangent k -space \mathbf{B} at p .

19 Bonus: Dualization/Orthogonal Complement (usual in GAs!)

The dual representation of subspaces is obtained by dividing a blade X by the volume blade I_n of the n -D vector space (aka the *pseudoscalar*).

$$X^* \equiv X/I_n$$

If $\text{grade}(X) = k$, in n -D space, then $\text{grade}(X^*) = n - k$.



The dual of a direction vector \mathbf{a} in 3D is a 2-blade \mathbf{A} of which \mathbf{a} is the normal vector.

It allows switching between dual representation (by IPNS), and direct representation (by OPNS).

direct sphere representation: $\Sigma = a \wedge b \wedge c \wedge d$ of grade 4, points satisfy $x \wedge \Sigma = 0$

\longleftrightarrow

dual sphere representation: $\sigma = \alpha (m - \frac{1}{2}\rho^2 n_\infty)$ of grade 1, points satisfy $x \cdot \sigma = 0$

20 IPNS and OPNS: Dual and Direct Representation

direct sphere representation: $\Sigma = a \wedge b \wedge c \wedge d$ of grade 4, points satisfy $x \wedge \Sigma = 0$
 \longleftrightarrow
dual sphere representation: $\sigma = \alpha (m - \frac{1}{2}\rho^2 n_\infty)$ of grade 1, points satisfy $x \cdot \sigma = 0$

This is a general principle, due to the duality of inner and outer product. A blade A can characterize a primitive object in two dually related ways:

$$\text{probe by a point } x \text{ (with } x^2 = 0\text{)} : \quad x \wedge A = 0 \quad \iff \quad x \cdot A^* = 0.$$

The former is sometimes called **OPNS** (outer product nullspace), the latter **IPNS** (inner product nullspace). Or **direct** and **dual**.

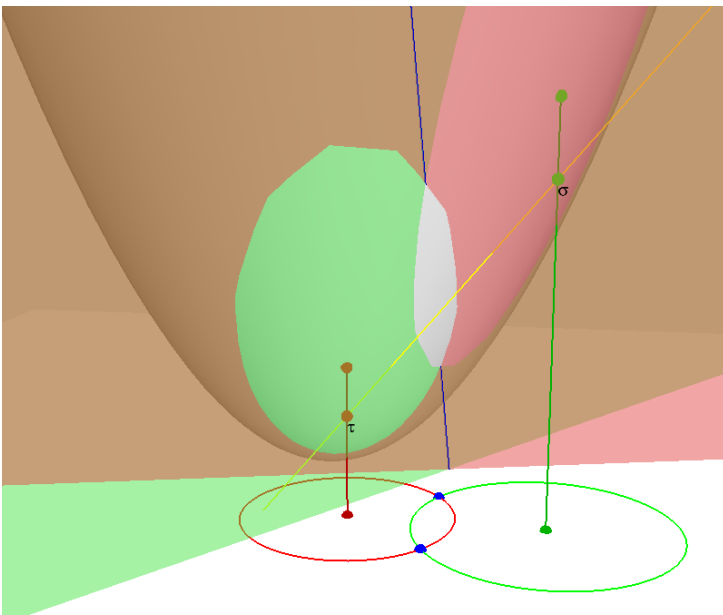
Use whichever matches your puzzle best - but keep in mind what you made! You cannot tell from the CGA form how an element is meant to be interpreted.

21 Bonus: Intersection of Elements

The intersection (**meet**) of two subspaces is dually given as the product of the duals:

$$\text{meet: } (A \cap B)^* = B^* \wedge A^*.$$

with duality relative to the pseudoscalar of the smallest subspace containing A and B (their **join**).



FIG(14,6)

In CGA, the **meet** of course also applies to the Euclidean elements.

Example: Intersection of Euclidean circles in 2D is like the intersection of their subspace blades, which are general planes in the 4D conformal model $\mathbb{R}^{3,1}$. That gives a line, on which there are two point representatives. So two circles meet in a point pair (possibly imaginary).

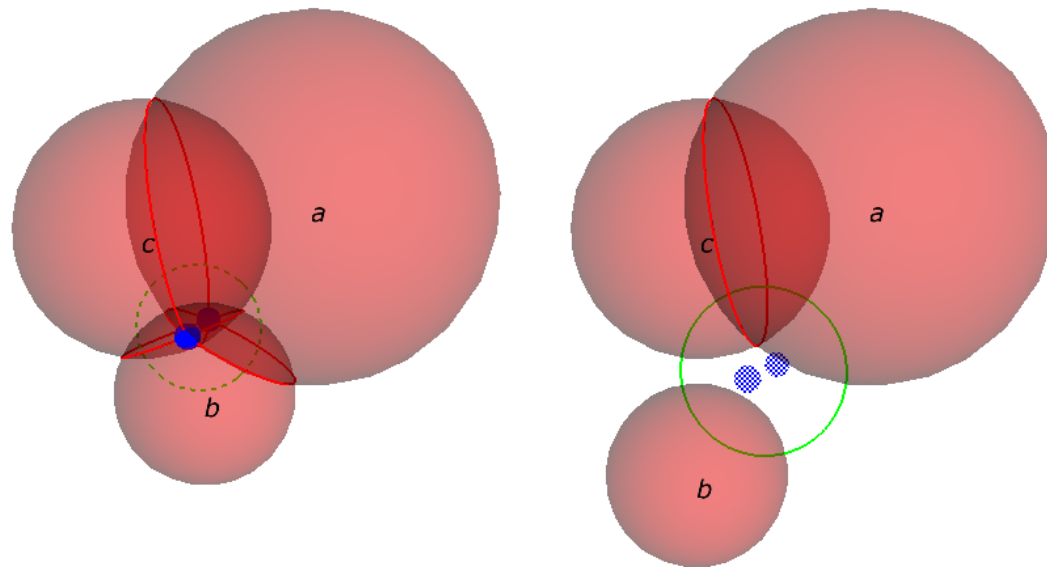
22 **Extra:** Plunge of Elements (A New Operation)

Also, the orthogonal element operation (**plunge**) can be defined:

$$\text{plunge: } A \perp B = B^* \wedge A^*.$$

It determines the subspace that hits A and B orthogonally. It is dual to the **meet**.

For three spheres, when one is real, the other is ‘imaginary’ (negative squared radius).



FIG(15,1)

Duality: ‘imaginary pole pair \leftrightarrow real equator’ and ‘real pole pair \leftrightarrow imaginary equator’. More in my other talk!

23 Big Bonus: Euclid's Elements Through Closure

By starting from spheres or planes dually represented as vectors, and making **plunges** and **meets**, one gets the Euclidean menagerie as algebraic elements (**blades**).

- *dual sphere*: the vector $\sigma = c - \frac{1}{2}\rho^2 n_\infty$.

$$0 = x \cdot \sigma = x \cdot c - \frac{1}{2}\rho^2 x \cdot n_\infty = -\frac{1}{2}((\mathbf{x} - \mathbf{c})^2 - \rho^2).$$

- *dual plane*: the vector $\pi = \mathbf{n} + \delta n_\infty$.

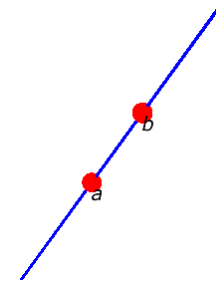
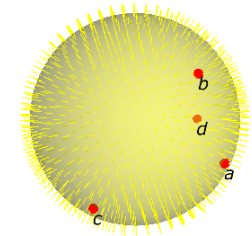
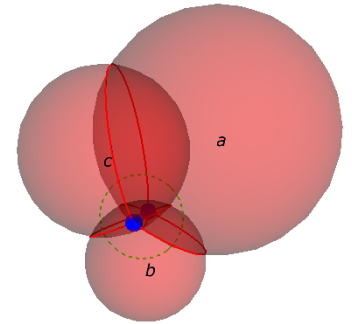
$$0 = x \cdot \pi = x \cdot \mathbf{n} + \delta x \cdot n_\infty = \mathbf{x} \cdot \mathbf{n} - \delta.$$

- *dual circle at origin*: the 2-blade $K = \sigma_0 \wedge \pi_0 = (n_o - \frac{1}{2}\rho^2 n_\infty) \wedge \mathbf{n}$.

$$0 = x \cdot (\sigma_0 \wedge \pi_0) = (x \cdot \sigma_0) \pi_0 - \sigma_0 (x \cdot \pi_0).$$

- *(primal) sphere*: the 4-blade $\Sigma = a \wedge b \wedge c \wedge d$.
- *(primal) plane*: the 4-blade $\Pi = a \wedge b \wedge c \wedge n_\infty = a \wedge \mathbf{B} \wedge n_\infty$.
- *(primal) line*: the 3-blade $L = a \wedge b \wedge n_\infty = a \wedge \mathbf{u} \wedge n_\infty$.
- *flat point*: the 2-blade $P = p \wedge n_\infty$.
- *direction blade*: the 3-blade $\mathbf{B} \wedge n_\infty$ is a 2-direction.
- *tangent vector*: the 2-blade $T = a \cdot (a \wedge \mathbf{u} \wedge n_\infty)$.

FIG(15,1) DEMOspanning()



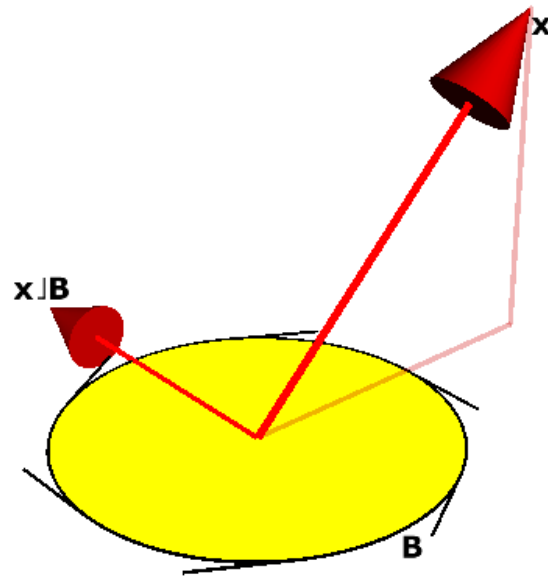
24 Trick 4: The Inner Product in Geometric Algebra (usual in GAs!)

We can define the **inner product** (aka as the **contraction**) as ‘adjoint’ to the outer product:

$$A \cdot B \equiv (A \wedge B^*)^{-*},$$

extending the dot product to blades. It is bilinear, but non-associative. It has properties like:

$$\mathbf{x} \cdot (\mathbf{a} \wedge \mathbf{b}) = (\mathbf{x} \cdot \mathbf{a}) \mathbf{b} - \mathbf{a} (\mathbf{x} \cdot \mathbf{b}).$$



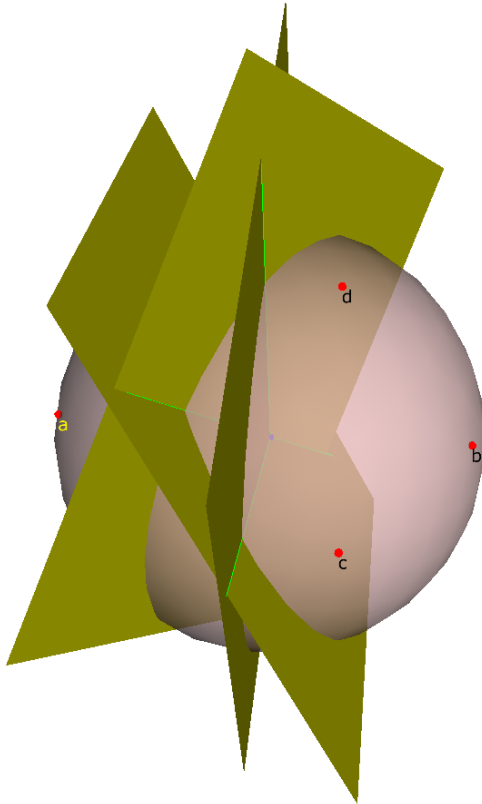
Interpretation of the inner product $\mathbf{x} \cdot \mathbf{B}$ for direction blades:

the orthogonal complement of \mathbf{x} contained in \mathbf{B} .

FIG(13,7)

25 Consistency of Dual Sphere Representations

The 4-blade $\Sigma = a \wedge b \wedge c \wedge d$ represents the sphere through the four points a, b, c, d .



$$\begin{aligned}
 \Sigma &= a \wedge b \wedge c \wedge d \\
 &= a \wedge \underbrace{(b - a) \wedge (c - a) \wedge (d - a)}_{\text{dual meet of 3 midplanes}} \\
 &\propto a \wedge \underbrace{(m \wedge n_\infty)^*}_{\text{flat center}} \\
 &= \left(a \cdot (m \wedge n_\infty) \right)^* \\
 &= \left((a \cdot m) n_\infty - (a \cdot n_\infty) m \right)^* \\
 &= \underbrace{\left(m - \frac{1}{2} \rho^2 n_\infty \right)^*}_{\text{dual sphere}} \\
 &= \sigma^*
 \end{aligned}$$

DEMOspheres()

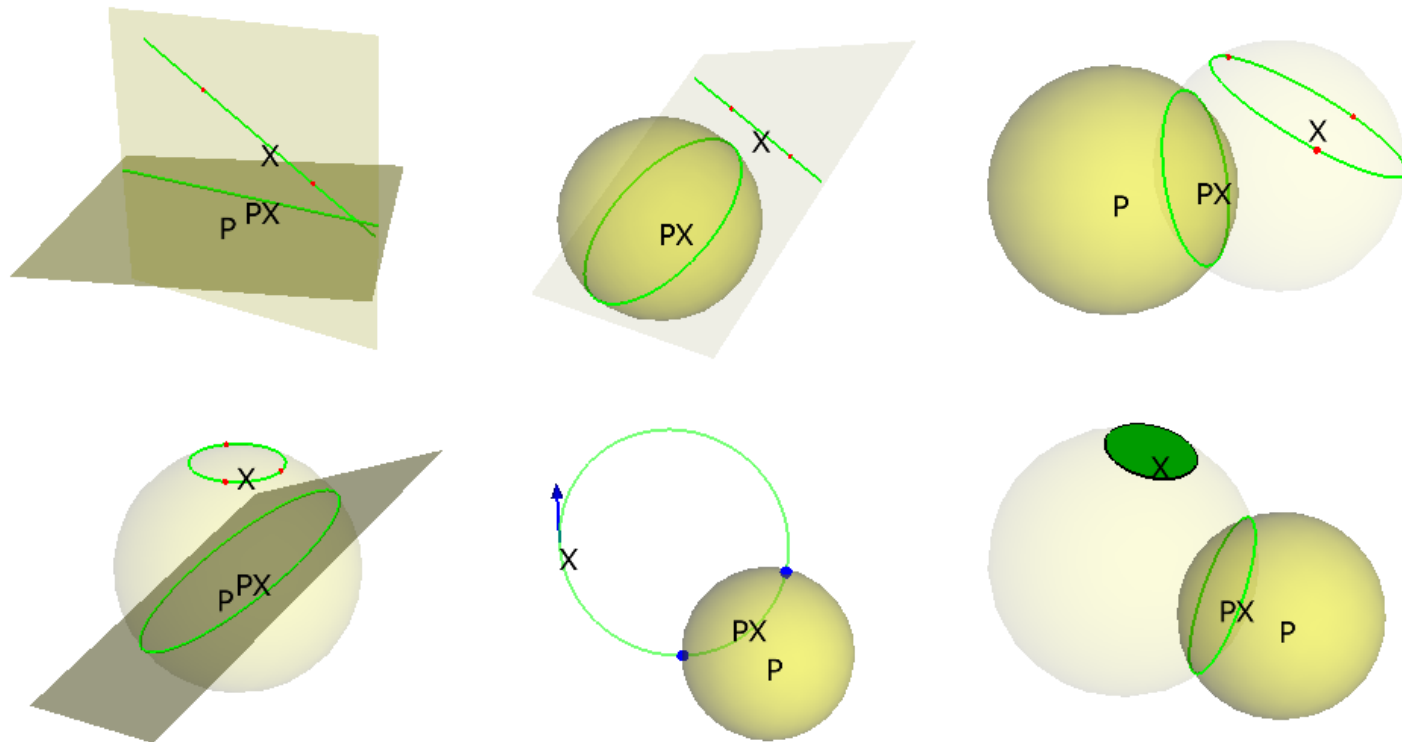
So it is easy to determine center and radius of a sphere through 4 points: they are the components of $(a \wedge b \wedge c \wedge d)^*$ (normalized).

26 Extra (SKIP!): Using the Inner Product for General Projections

CGA has a general projection operator, a structural generalization of $\mathbf{x} \mapsto (\mathbf{x} \cdot \mathbf{a})/\mathbf{a}$:

$$\text{Proj}_P(X) = (X \cdot P)/P \quad (= P^{-1} \cap (X \wedge P^{-*})).$$

This **meets** P^{-1} with a Euclidean element $X \wedge P^{-*}$ ‘containing X and plunging into P orthogonally’.



DEMOprojection()

27 The Really Great Thing about CGA: Euclidean Motions as Versors

Back to the motions. Make them through multiple reflections in the representative vectors of (dual) planes and spheres. (This is Cartan-Dieudonné in action!)

- **Translation:** two parallel planes, $\mathbf{t}/2$ apart.

Computation: $(\mathbf{t} + \frac{1}{2}\mathbf{t} \cdot \mathbf{t} n_\infty) \mathbf{t} = \mathbf{t}^2 (1 - \mathbf{t} n_\infty/2)$.

$$\text{translation versor: } T_{\mathbf{t}} = 1 - \mathbf{t} n_\infty/2$$

- **Rotation at Origin:** two intersecting planes, $\phi/2$ apart.

Computation: $(\cos(\phi/2)\mathbf{e}_1 + \sin(\phi/2)\mathbf{e}_2) \mathbf{e}_1 = \cos(\phi/2) - \sin(\phi/2) \mathbf{e}_1 \wedge \mathbf{e}_2$.

$$\text{rotation versor: } R_{\mathbf{I}\phi} = \cos(\phi/2) - \sin(\phi/2) \mathbf{I}$$

- **Uniform scaling at Origin:** two concentric spheres.

Computation: $(n_o - \frac{1}{2}\rho^2 n_\infty) (n_o - \frac{1}{2}n_\infty) = -\frac{1}{2} \left((1 + \rho^2) - (1 - \rho^2) n_o \wedge n_\infty \right)$.

$$\text{uniform scaling versor: } S_\gamma = \cosh(\gamma/2) + \sinh(\gamma/2) n_o \wedge n_\infty$$

- **Transversion at Origin:** two touching spheres of equal radius.

$$\text{transversion versor: } V_{\mathbf{v}} = 1 + n_o \mathbf{v}$$

28 Big Bonus: Structure Preservation (usual in GAs!)

Reflection of vector \mathbf{x} in a plane with normal \mathbf{a} is: $\mathbf{x} \mapsto -\mathbf{a}\mathbf{x}\mathbf{a}^{-1}$. If X is a product of vectors \mathbf{x}_i , this extends to:

$$X \mapsto (-\mathbf{a}\mathbf{x}_1\mathbf{a}^{-1})(-\mathbf{a}\mathbf{x}_2\mathbf{a}^{-1}) \cdots (-\mathbf{a}\mathbf{x}_k\mathbf{a}^{-1}) = \mathbf{a}\hat{X}\mathbf{a}^{-1}$$

with $\hat{\mathbf{x}} = (-1)^{\dim(\mathbf{x})}\mathbf{X}$, the ‘main involution’. Then distributes over ‘constructive’ products \wedge and \cdot , effectively weighted sums of geometric products:

$$\mathbf{B} \mapsto (-\mathbf{a}\mathbf{b}_1\mathbf{a}^{-1}) \wedge (-\mathbf{a}\mathbf{b}_2\mathbf{a}^{-1}) \wedge \cdots (-\mathbf{a}\mathbf{b}_k\mathbf{a}^{-1}) = (-1)^k \mathbf{a}(\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \cdots \mathbf{b}_k)\mathbf{a}^{-1} = \mathbf{a}\hat{\mathbf{B}}\mathbf{a}^{-1}.$$

So

the sandwiching product is structure preserving.

Also preserved in multiple reflections, so now all motions are universally applicable to all geometric elements using their versors:

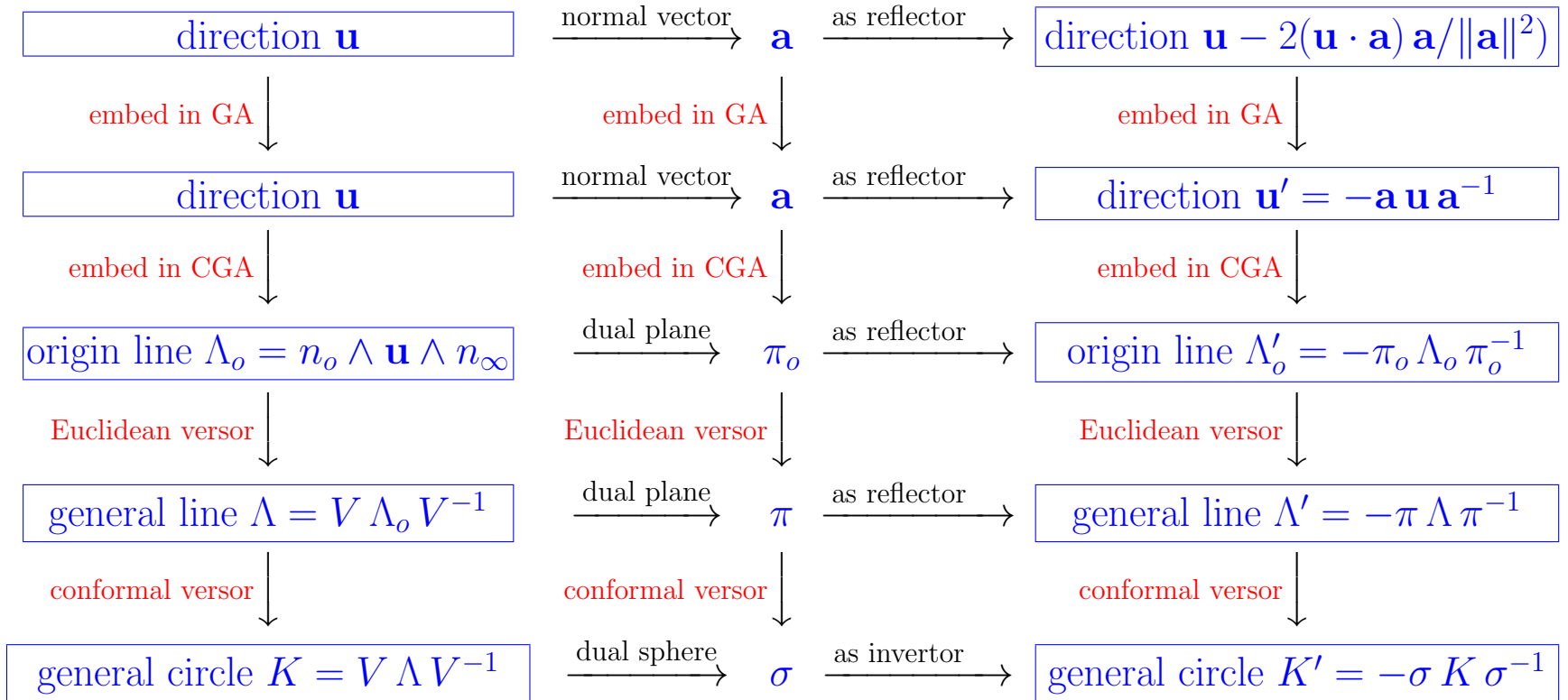
$$\begin{aligned} V_{\text{even}} : X &\mapsto V X V^{-1} \\ V_{\text{odd}} : X &\mapsto V \hat{X} V^{-1} \end{aligned}$$

This universality is very unlike the homogeneous coordinate approach, and enormously simplifies software.

(By the way, even V have determinant 1, odd V have determinant -1 .)

29 Example: Transfer Principle for Reflection in CGA

For reflection $\mathbf{x} \mapsto -\mathbf{a} \mathbf{x} \mathbf{a}^{-1}$: transfer from **direction vector** to **origin line** to **general line**. Simultaneously, the **dual origin plane** becomes the **general dual plane** through the intersection point. Further extension to inversion possible by conformal versor.



This all just holds by structure preservation, there is no need to prove anything!

30 Trick 5: The Bivector Representation of Even Versors (usual in GAs!)

In a rotation, we have the rotation plane \mathbf{I} and angle ϕ . What is the versor?

Versors were introduced through products of invertible vectors. But an even versor V can be written as the *exponential of a bivector* B :

$$V = e^B$$

The bivector specification often corresponds more directly to the desired geometry.

Example: rotation over ϕ in plane \mathbf{I} through the origin as exponential:

$$R = \cos(\phi/2) - \mathbf{I} \sin(\phi/2) = 1 + (-\mathbf{I}\phi/2) + \frac{1}{2!}(-\mathbf{I}\phi/2)^2 + \dots = e^{-\mathbf{I}\phi/2},$$

since $\mathbf{I}^2 = -1$. We will show that a general 3D rotation with rotation line Λ is $R = e^{\Lambda^*\phi/2}$.

Example: translation over \mathbf{t} :

$$T = 1 - \mathbf{t} \wedge n_\infty/2 = 1 - \mathbf{t} \wedge n_\infty/2 + \frac{1}{2!}(-\mathbf{t} \wedge n_\infty/2)^2 + \dots = e^{-\mathbf{t} \wedge n_\infty/2},$$

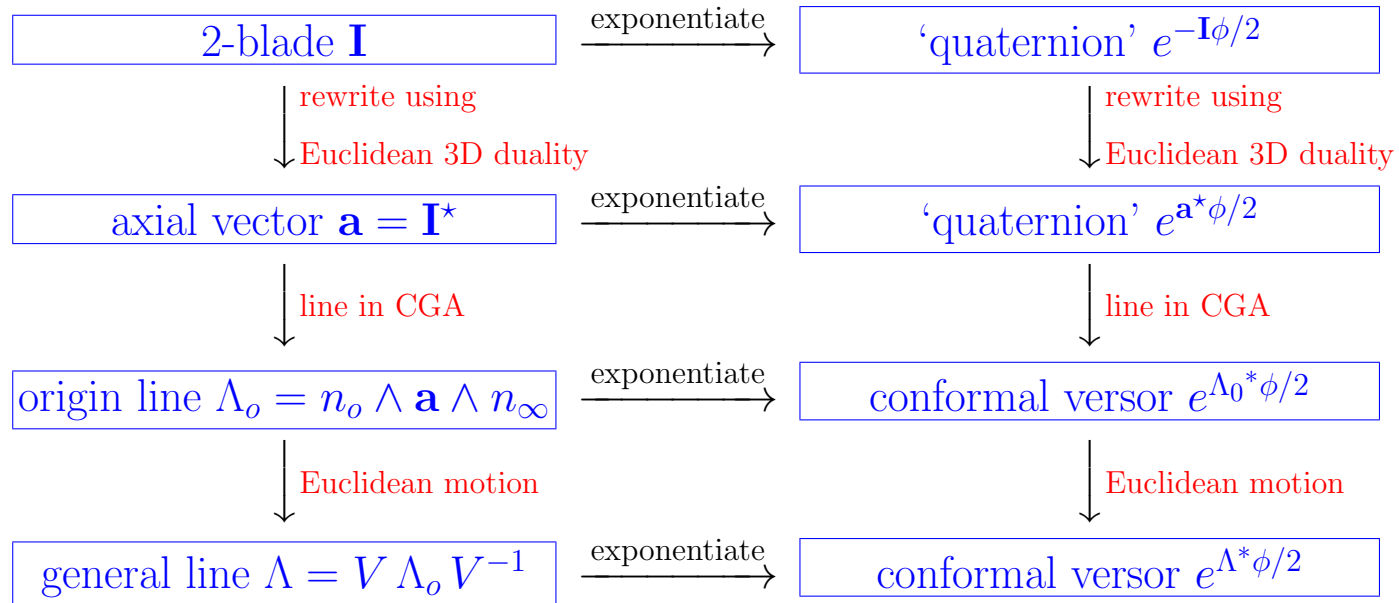
since $(\mathbf{t} \wedge n_\infty)^2 = (\mathbf{t} n_\infty)^2 = \mathbf{t} n_\infty \mathbf{t} n_\infty = -\mathbf{t} \mathbf{t} n_\infty n_\infty = -\mathbf{t}^2 n_\infty^2 = 0$.

31 Bonus: Transfer Principle to Make General Euclidean Motion Versors

Transfer principle for exponentially represented versors:

$$\begin{aligned}
 V \exp(B) V^{-1} &= V (1 + B + \frac{1}{2!} B B + \dots) V^{-1} \\
 &= 1 + V B V^{-1} + \frac{1}{2!} (V B V^{-1}) (V B V^{-1}) + \dots \\
 &= \exp(V B V^{-1}).
 \end{aligned}$$

Application: the conformal versor for rotation around an arbitrary unit line Λ , over ϕ .



The only thing to prove here is the duality transfer from 3D Euclidean to conformal:

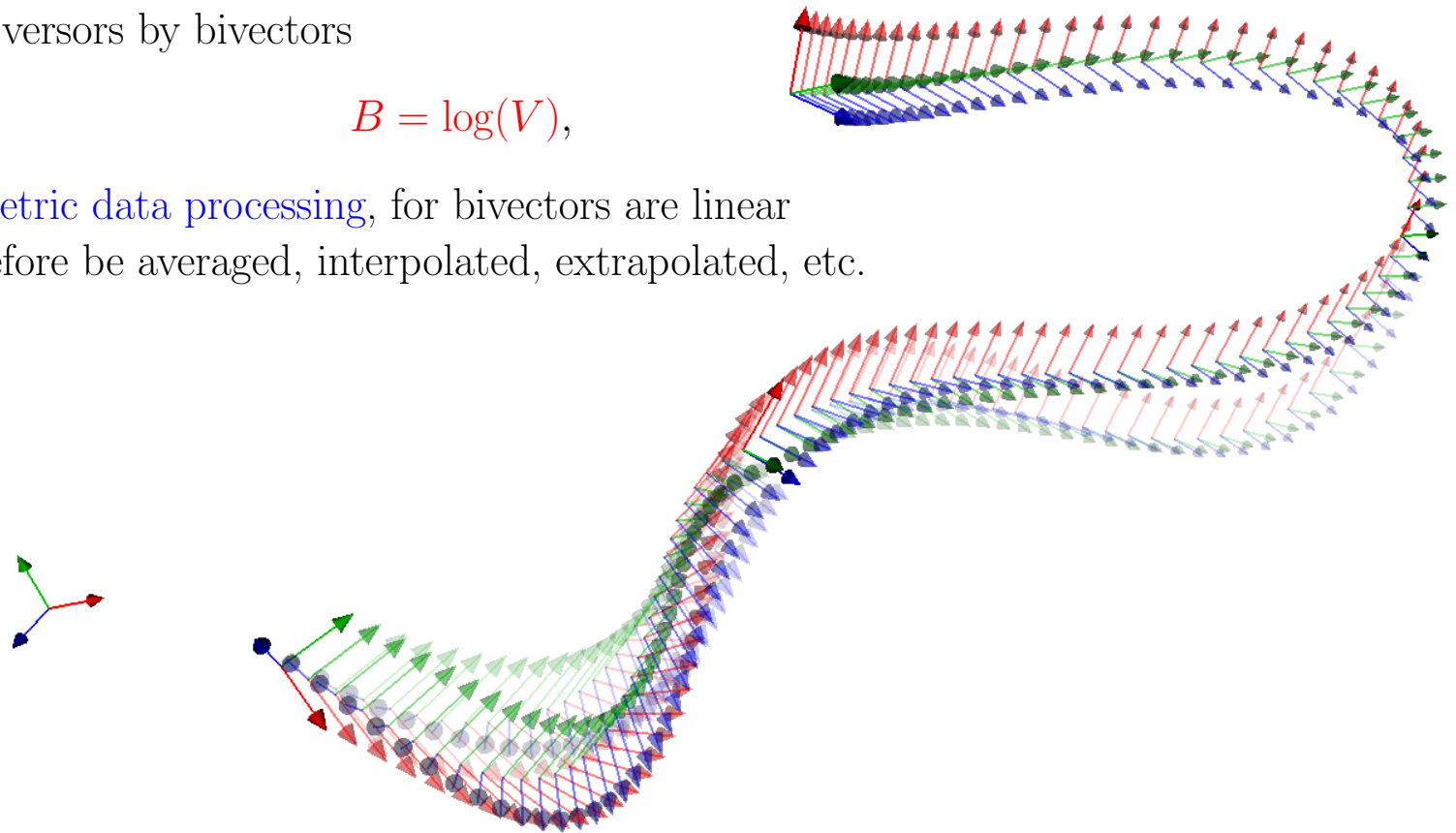
$$\Lambda_o^* = (n_o \wedge \mathbf{a} \wedge n_\infty)^* = (n_o \wedge \mathbf{a} \wedge n_\infty) (n_o \wedge \mathbf{I}_3^{-1} \wedge n_\infty) = \mathbf{a} \mathbf{I}_3^{-1} = \mathbf{a}^*.$$

32 Bonus: Logarithms of Motions Behave Linearly

Representing versors by bivectors

$$B = \log(V),$$

permits [geometric data processing](#), for bivectors are linear and can therefore be averaged, interpolated, extrapolated, etc.



tutorial/DEMOinterpolaterbm()

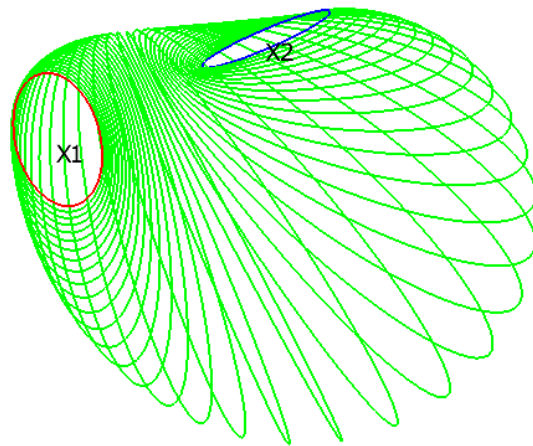
Logarithms for all conformal transformations in 3D are known, from Dorst & Valkenburg's chapter in 'Guide to Geometric Algebra', 2011.

33 Interpolation

- The [logarithm of any rotor in 3D CGA](#) can be computed.
- Therefore any relative rotor R can be [linearly interpolated](#) in its parameters:

$$R(t) = \exp(\log(R) t), \quad t \in [0, 1].$$

- The 2-blades in the logarithm can be used in [more advanced interpolation schemes](#) (such as [B-splines](#)).



roots-and-rbm/Vratio()

The ‘shortest’ conformal motion to transform $X1$ into $X2$.

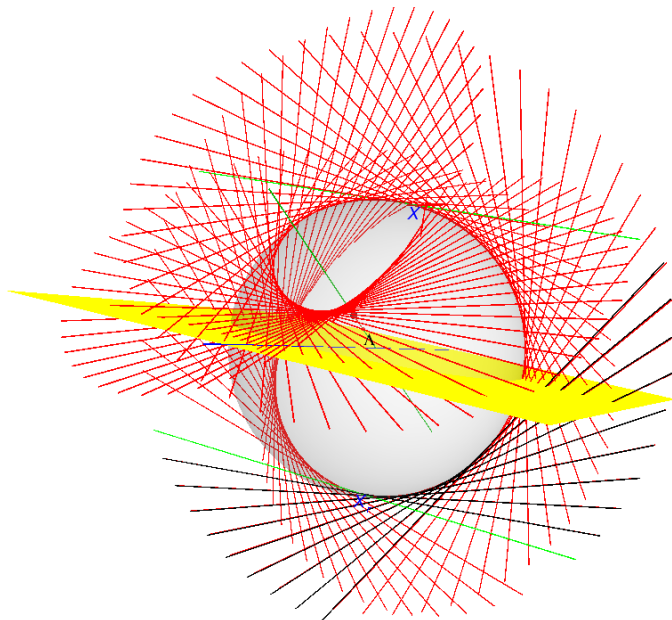
34 Geometric Calculus and Extrapolation (usual in GAs!)

The elements of geometric algebra can also be differentiated with respect to each other. This allows for compact derivations of advanced results.

Perturbations are simple, and linear in B , involve a **commutator product** (like Lie algebra):

$$e^{-\delta B/2} X e^{\delta B/2} \approx X + \frac{1}{2}(X \delta B - \delta B X) \equiv X + X \times \delta B.$$

First order treatment of second-order motions! Exact linearity, so apply linear data processing methods with greatly extended functionality!



Example in CGA: Yellow mirror Π rotates ϕ around green line Λ , where do reflected lines like X' go?

In black: using first order bivector perturbation (i.e., second order approximation of orbit), gives rotation for X' with versor

$$e^{-\phi((\Lambda \cdot \Pi)/\Pi)^*},$$

i.e. turning around the projected line, with angle $2\phi \cos(\Pi, \Lambda)$.

FIG(13,7)

35 Trick 6: Implementation (Size Matters, But Is Not Prohibitive)

- GA can represent all 2^m subspaces of an m -D vector space as elements of computation.
- To represent Euclidean motions in \mathbb{R}^n as versors, you need CGA, i.e. GA of $\mathbb{R}^{n+1,1}$ -D space.
- That is a 32-dimensional representation for 3D Euclidean geometry!
- Efficient implementation is therefore an issue.
- Solved by using the structure of GA in an automatic code generator. (Fontijne 2007 PhD thesis)
- Result: high-level programming in GA available (subspace products, sandwiching).
- The actual algebraic computation takes care of the type administration; the implementation performs this at compile time. Effectively the program does hardly more than linear algebra at the lowest level, but based on a high-level specification using CGA products and operators.

Executive summary:

Your people can now use a high-level language (GA)

- *to specify Euclidean geometry,*
- *at semantic level; this generates code,*
- *for competitive efficiency with classical approach,*
- *but with more compact, more maintainable and less error-prone code.*

36 Structural Aspects of a Language for Geometry

- *primitives*: points, lines, planes, circles, spheres, tangents
- *constructions*: connections, intersections, orthogonal complement, duality
- *motions*: translations, rotations, reflection, projection
- *properties*: size, location, orientation, cross ratio
- *numerics*: approximation, estimation, linearization

Motions are at the basis of geometry (Klein), structure preservation:

Constructions and properties of primitives should be covariant under motions.

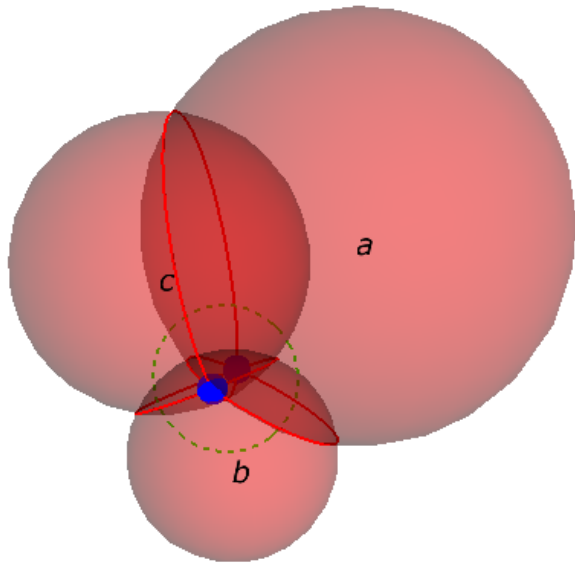
Example: covariant intersection: meet of moved objects should equal moved meet of the objects.

$$\begin{array}{ccc}
 \text{line } \Lambda, \text{ plane } \Pi & \xrightarrow{\text{intersect}} & \text{point } \Lambda \cap \Pi \\
 \text{motion} \downarrow & & \downarrow \text{motion} \\
 \text{line } \Lambda', \text{ plane } \Pi' & \xrightarrow{\text{intersect}} & \Lambda' \cap \Pi' = (\Lambda \cap \Pi)'
 \end{array}$$

Not automatic in standard linear algebra!

(This example would be $M \ker(\text{span}(\Lambda, \Pi)^\top) = \ker(\text{span}(M^{-\top} \Lambda, M^{-\top} \Pi)^\top)$ in hom. coordinates.)

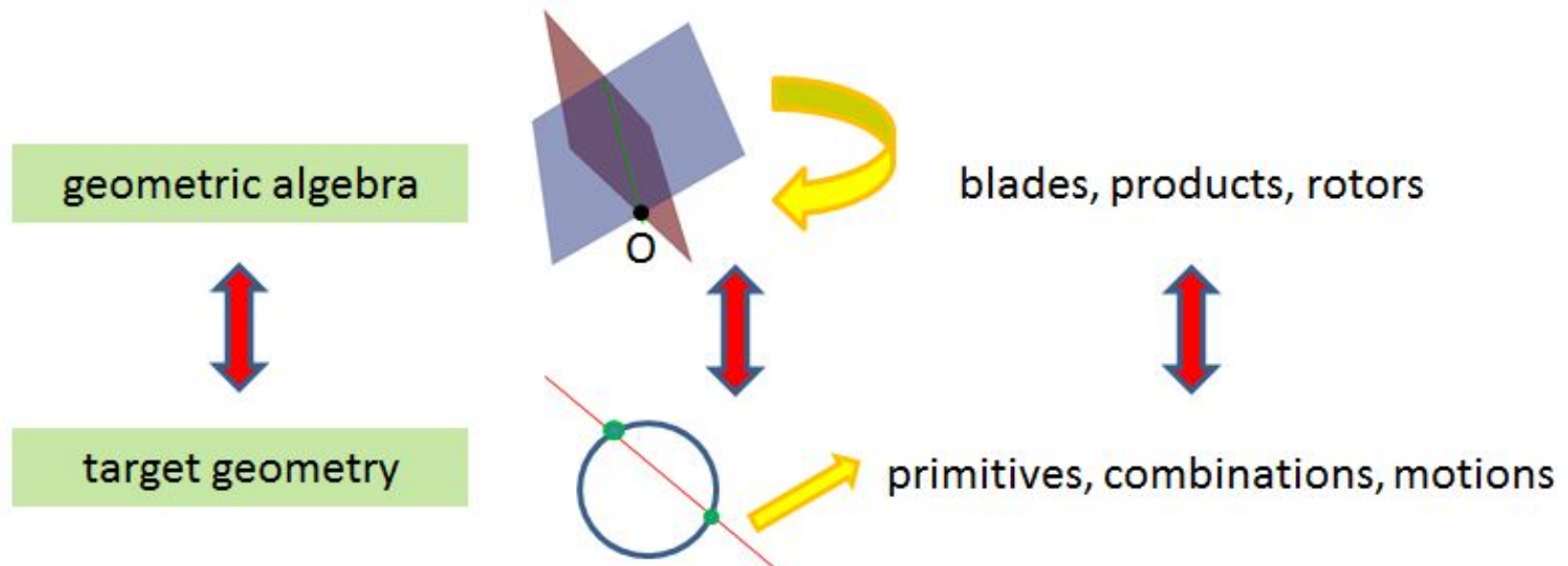
37 Summary: Euclid's Elements in the Conformal Model



FIG(15,1)

- *primitives as subspaces:*
points, lines, planes, circles, spheres, tangents
- *constructions as subspace products:*
connections, intersection, plunge, duality
- *motions as versors:*
translations, rotations, reflections in planes or spheres
(actually, any conformal transformation)
- *properties parametrized:*
size, weight, location, orientation, direction
- *numerics exactly linearized:*
linear (bivector) parametrization of motions

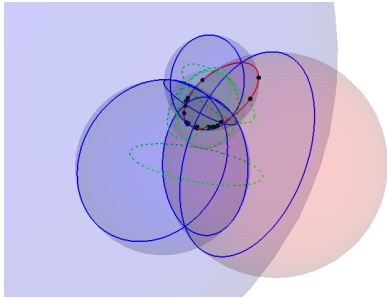
38 Conformal Model is One Example of the GA Modeling Strategy



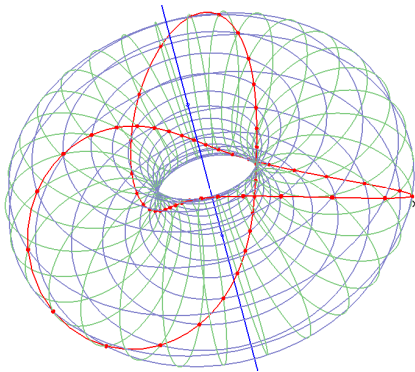
More examples exist, and the list is growing. To mention some:

- $\mathbb{R}^{n+1,1}$ for n -D conformal geometry
- $\mathbb{R}^{3,3}$ for 3-D projective geometry
- $\mathbb{R}^{4,4}$ for 3-D projective geometry (but too big?)
- $\mathbb{R}^{9,6}$ for 3-D quadrics (but is it really?)
- $\mathbb{R}^{4,2}$ for 2-D images (Koenderink)

39 What Will Tomorrow Bring?

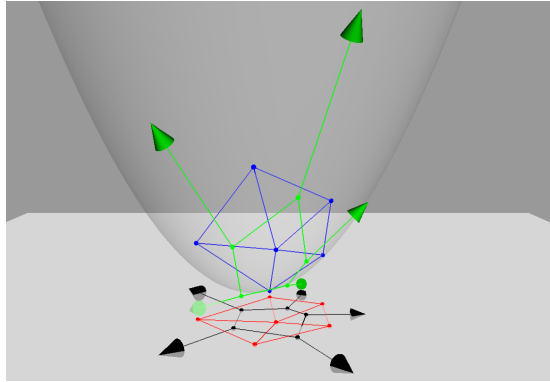


- **CGA in Practice 1: Least Squares Fitting of Spatial Circles**
Given a cloud of points, what is the best circle through them?
We will use geometric differentiation reduce this numerical optimization puzzle to an eigenproblem in CGA.

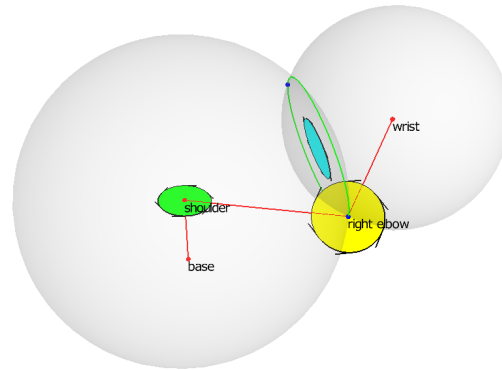


- **CGA in Practice 2: The Construction of 3D Conformal Motions**
What are conformal motions and how can we specify them conveniently?
A geometrically intuitive way to manipulate conformal motions, as exponentials of commuting point pairs. Chasles included!

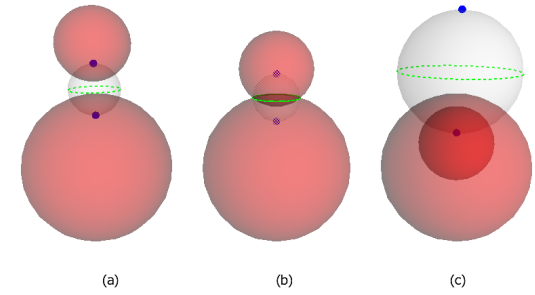
40 Appendix 1: Various CGA Demos



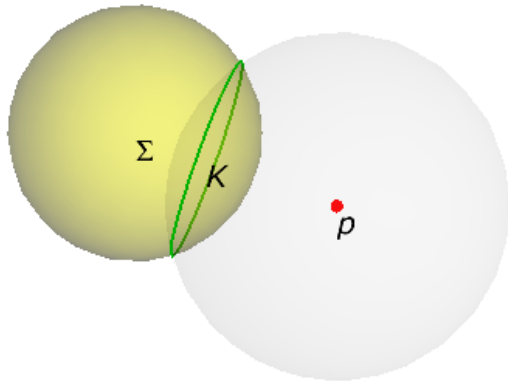
Voronoi diagrams FIG(14,7)



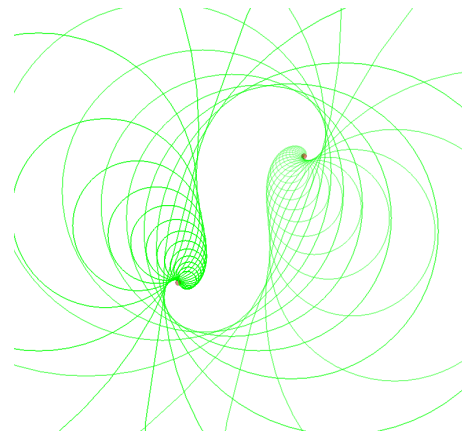
Inverse Kinematics FIG(14,10)



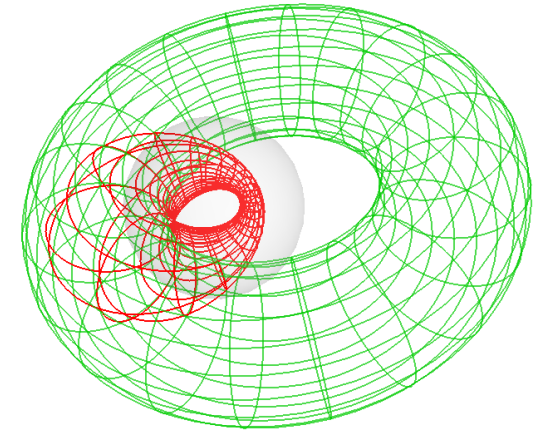
Sphere Intersection FIG(15,3)



Contour circle FIG(15,13)



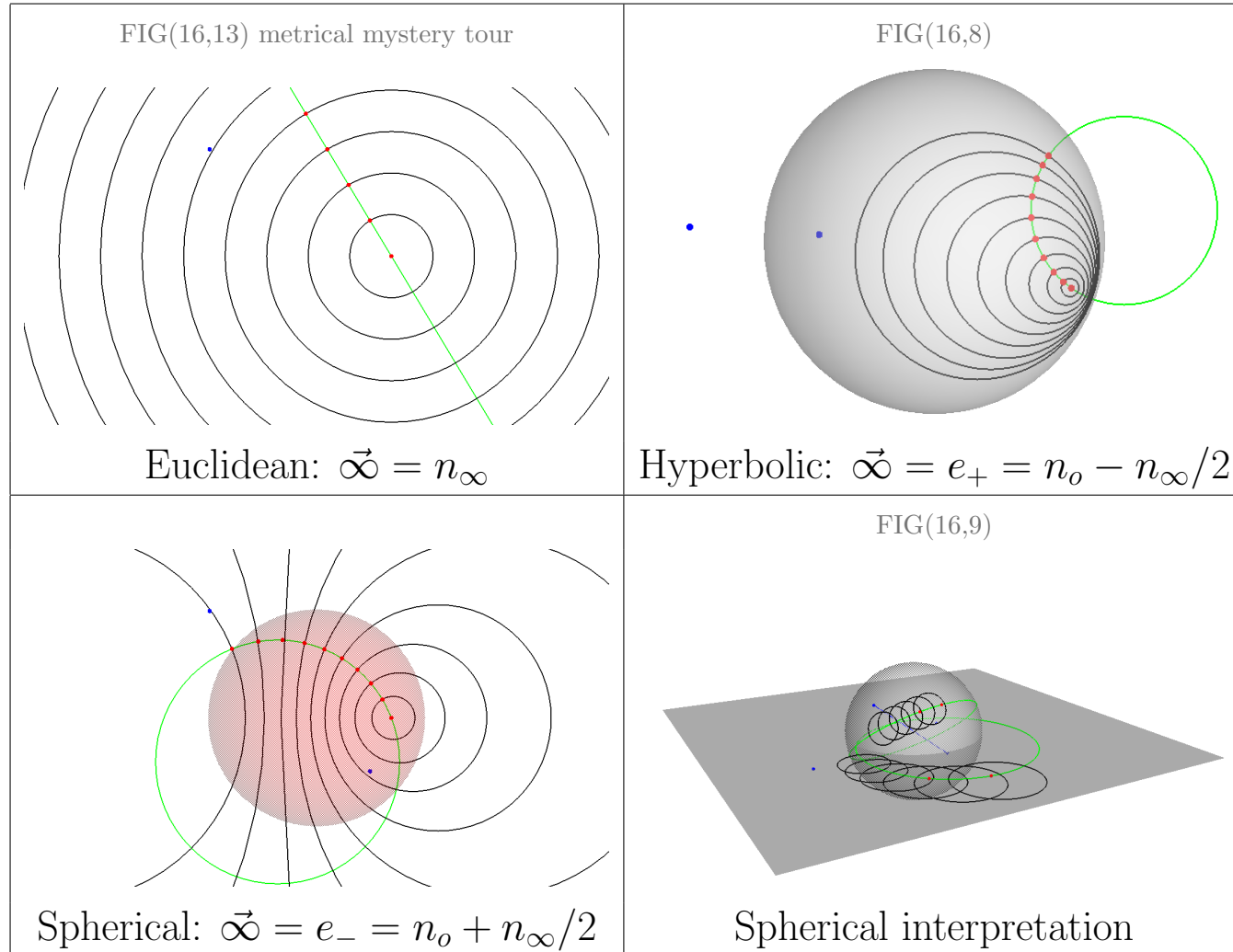
Loxodromes FIG(16,6)



Dupin's Cyclide FIG(16,12)

41 Appendix 2: Non-Euclidean Geometries

By changing the ‘infinity’ vector $\vec{\infty}$, we can get conformal representations of various metrics.

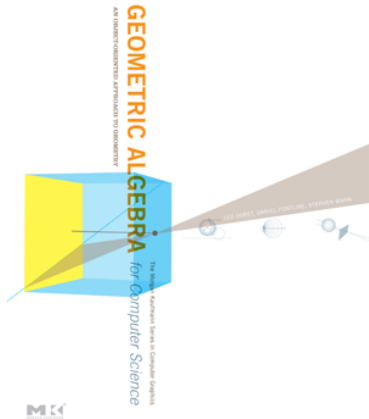


42 Appendix 3: Recommended Reading

Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry

Leo Dorst, Daniel Fontijne, Stephen Mann

(Morgan-Kaufmann Publishers 2009, ISBN 978-0-12-374942-0)

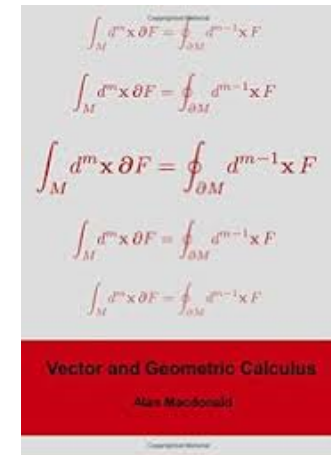
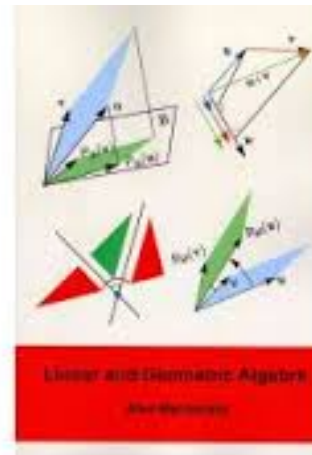


- 22 chapters, 4 appendices, 650 pages, 150+ full color figures, free software.
- Available everywhere, price circa €50, \$100.
- Book website, freely downloadable software and demos:
www.geometricalgebra.net

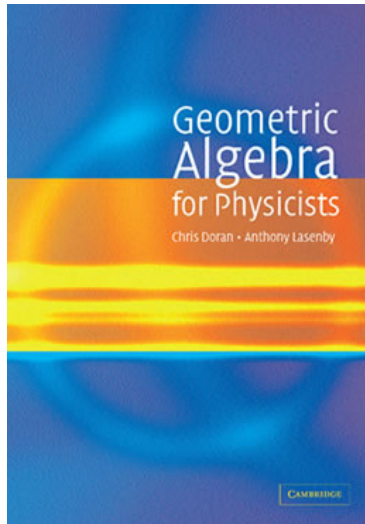
Linear and Geometric Algebra Linear and Geometric Calculus

Alan Macdonald

- 9 chapters, xii+186 pages, Python software
- Price circa \$30, €20.



43 **Appendix 3:** Recommended Reading, Continued



Geometric Algebra for Physicists

Chris Doran and Anthony Lasenby

(Cambridge University Press 2003)

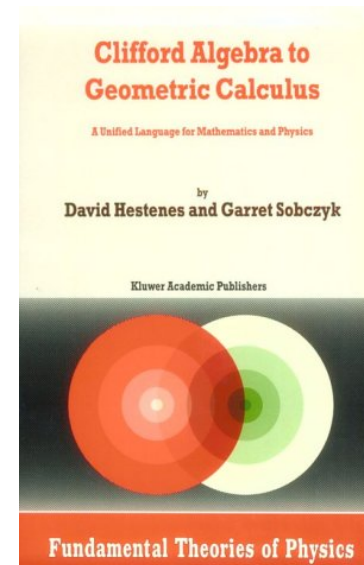
- 14 chapters, 592 pages
- Price circa £42 (paperback)

Clifford Algebra to Geometric Calculus

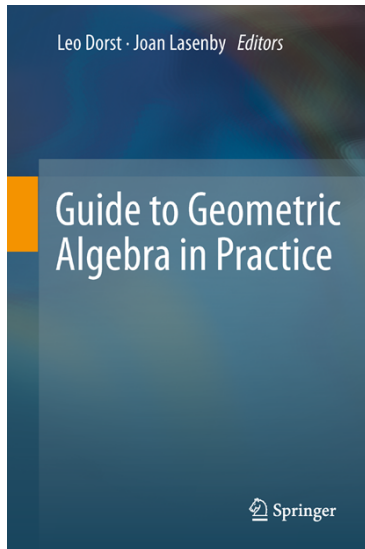
David Hestenes and Garret Sobczyk

(Kluwer Academic Publishers 1984)

- 8 chapters, 336 pages
- Price about \$225



44 **Appendix 3:** Recommended Reading, Continued



Guide to Geometric Algebra in Practice

editors: Leo Dorst and Joan Lasenby

(Springer 2011)

Selected papers of AGACSE 2010, in somewhat textbook format, with tutorial intro and problems for each chapter.

- 21 chapters (including tutorial), 470 pages
- Price circa €100