2. Honing geometric algebra for its use in the computer sciences

Leo Dorst

Dept. of Computer Science, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
leo@wins.uva.nl

2.1 Introduction

A computer scientist first pointed to geometric algebra as a promising way to 'do geometry' is likely to find a rather confusing collection of material, of which very little is experienced as immediately relevant to the kind of geometrical problems occurring in practice. Literature ranges from highly theoretical mathematics to highly theoretical physics, with relatively little in between apart from some papers on the projective geometry of vision [8]. After perusing some of these, the computer scientist may well wonder what all the fuss is about, and decide to stick with the old way of doing things, i.e. in every application a bit of linear algebra, a bit of differential geometry, a bit of vector calculus, each sensibly used, but *ad hoc* in their connections. That this approach tends to split up actual issues in the application into modules that match this traditional way of doing geometry (rather than into natural divisions matching the nature of the problem) is seen as 'the way things are'.

However, if one spends at least a year in absorbing this material, a different picture emerges. One obtains increased clarity and prowess in handling geometry. This is due to being able to do computations without using coordinates; and by having elements of computation which are higher-dimensional than vectors, and thus collate geometrical coherence. The operators that can be applied are at the same time more limited in number, and more powerful in purity and general validity. Through this, one obtains the confidence to tackle higher-dimensional parameter spaces with the intuition obtained from 3-dimensional geometry. Programs written are magically insensitive to the dimensionality of the embedding space, or of the objects they act on. The concept of a 'split' endows the limited set of operators with a varied semantics, which begins to suggests an applicability to all geometries one is likely to encounter.

The hardest part in achieving such a re-appraisal is actually letting go of the usual geometrical concepts, and embracing new ones. It is not hard to rewrite, say, linear algebra into geometric algebra; but it is a different matter altogether to use the full power of geometric algebra to solve problems for which one would otherwise employ linear algebra. This overhaul of the mind takes time; and would be greatly aided by material aimed towards computer

scientists. This will doubtlessly appear, for an evangelical zeal appears to be common to all who have been touched by geometric algebra, but at the moment it is scarce.

So geometric algebra can (and will) change computer science; but vice versa, the need for a clear syntax and semantics for the geometric objects and operators in a specification language requires a rigor beyond the needs of its current applications in physics, and this is where computer science may affect geometric algebra. Imposing this necessary formalization – always with the applications in mind – reveals some ambiguities in the structure of geometric algebra which need to be repaired. This paper reports on some issues encountered when preparing the wealth of geometric algebra for its application in the computer sciences. They involve simply making the internal structure explicit (section 2.2); redesigning the operators (even the rather basic inner product can be improved, in section 2.3); the development of new techniques to enable the user to adapt the structure to his or her needs (section 2.4); and making mathematical isomorphisms explicit in applicable operators (section 2.5).

When this is done, many individual 'tricks' occurring in different branches of classical geometry become unified (this is shown for the 'meet' in section 2.6), and therefore implementable in a generic toolbox structuring the thinking and increasing the capabilities of the geometrical computer scientist. This is an ongoing effort; as a consequence, this paper is still directed more towards the *developers* of geometric algebra than towards its *users*. Yet it should help potential users to assess these exciting new developments in unified geometrical computation.

2.2 The internal structure of geometric algebra

The monolithic term 'geometric algebra' hides an internal structure that consists of various levels, each of which are relevant to the computer scientist desiring to use it in an application. It is important to distinguish them, for various branches of literature deal with different levels – so you may not find what you need in any one book or paper. I have found the levels sketched in table 2.1 useful in expositions on the subject, since they explicitly indicate the scope of each part of the formalism. They are depicted in a 'bottom-up' manner from the mathematics of Clifford algebras (at the basis) to various applications (at the top). (Some levels and their names are my own suggestions, for the purpose of this paper; I put them in quotes throughout.)

– **Clifford algebra**
  At the basis of all geometric algebra is Clifford algebra. This introduces a *(Clifford) product* in a vector space V^n over a field of scalars K, thereby

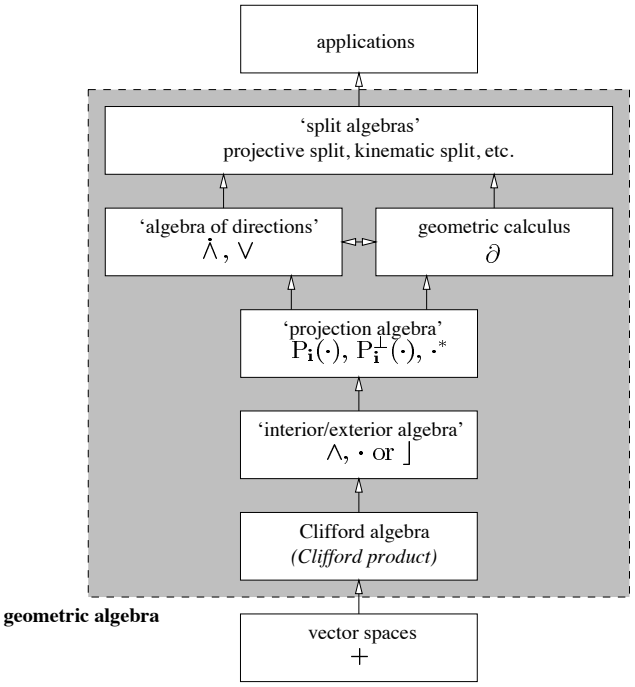


Fig. 2.1. Levels in geometric algebra with their operators (non-standard terms in quotes).

extending it to a 2^n-dimensional linear space of multivectors.[1] This product is commonly introduced using a bilinear form $\langle\,,\,\rangle : V^n \times V^n \to K$ or a quadratic form $Q : V^n \to K$, to satisfy the axioms:

1. *scalars commute with everything*: $\alpha\, u = u\, \alpha$, for $\alpha \in K$, $u \in \mathcal{C}\!\ell_n$.
2. *vectors* $\mathbf{x} \in V^n$ *obey:* $\mathbf{x}\mathbf{x} = Q(\mathbf{x})$ (which is a scalar!).
3. *algebraic properties:* geometric product is linear in both factors, associative, and distributive over $+$. Do *not* demand commutativity!

Repeated application of the geometric product then produces the basic elements for the whole Clifford algebra, consisting of scalars, vectors, bivectors, etcetera. A big mathematical advantage of the Clifford product is that it is in principle invertible (the inverse of a vector $\mathbf{x}$ is $\mathbf{x}/Q(\mathbf{x})$). This gives a much richer algebraic structure than other products on vectors (such as the inner product) – with far-reaching practical consequences. For instance, a subject that can be studied fully within Clifford algebra, just using the Clifford product, is *n-dimensional rotations*, represented by *spinors*. Rotations are directly represented as elements of the space of the algebra,

[1] Several levels higher, the geometric semantics of this product suggests itself so strongly that it has become custom to denote the Clifford product as a 'geometric product'; but at this basic level that is not obvious yet, and leads to confusion.

just as vectors are, rather than as elements of an algebra of mappings on a vector space (as they are in linear algebra).
When one starts studying the properties and relationships of various Clifford algebras, it turns out that these depend on the signature of the quadratic form; but in this contribution we will not emphasize this, using $\mathcal{C}\ell_n$ to denote a Clifford algebra for the vector space V_n, and (slightly casually) for its space of multivectors.[2]
The mathematics of Clifford algebra has been studied sufficiently for all immediate purposes in computer science, and good accounts exist (try [13], chapter 1). The style of explanation in such accounts is often 'permutation of indices' rather than 'geometrically motivated construction', a consequence of its close (and, to mathematicians, interesting) relationship to *tensor algebra*. Although this is somewhat off-putting at first, it does give a clear indication to the computer scientist of how the basic operations can be implemented efficiently, and how their syntax is defined independent of any geometric semantics we might choose to impose later.

– **'interior/exterior algebra'**
In derivations in Clifford algebra, one often uses commutativity or anti-commutativity of Clifford products. This occurs so often that it makes sense to decompose the Clifford product of vectors into a symmetric and anti-symmetric part under commutation, and use those as higher level 'macros' to develop higher level insights. There is an unambiguous and agreed-upon choice for the anti-symmetric *outer product* $\wedge$ which is introduced by the axioms:

$$\begin{array}{ll} 1. & \mathbf{x} \wedge u = \frac{1}{2}(\mathbf{x}\,u + \widehat{u}\,\mathbf{x}) \quad \text{for } \mathbf{x} \in V^n, u \in \mathcal{C}\ell_n \\ 2. & \wedge \text{ is linear in both arguments, and associative.} \end{array} \tag{2.1}$$

For the symmetric counterpart there are two choices, the *inner product* '$\cdot$' or the *contraction* '$\rfloor$', both agreeing on vectors:

$$\mathbf{x} \cdot u = \mathbf{x} \rfloor u = \tfrac{1}{2}(\mathbf{x}\,u - \widehat{u}\,\mathbf{x}) \quad \text{for } \mathbf{x} \in V^n, u \in \mathcal{C}\ell_n,$$

but differing in action on general multivectors (details later). In geometric algebra as developed for physics [4], the inner product '$\cdot$' is used. We will argue below why the contraction '$\rfloor$' is preferable since it gives a cleaner algebraic computational structure, without exceptions or conditions to geometrically meaningful results.

– **'projection algebra'**
The fresh contribution of Clifford algebras to the way we compute in geometry is the treatment of *higher dimensional subspaces as basic elements of*

[2] It is a dilemma, when learning Clifford algebra, whether you should do algebras of purely Euclidean spaces first (most intuitive!), or learn it in its full generality from the start (most general!). In any case, you will have to learn non-Euclidean Clifford algebras eventually, because the projective split (section 2.5 shows that non-Euclidean Clifford algebras are a very convenient representation for computations on the affine geometry of purely Euclidean spaces!

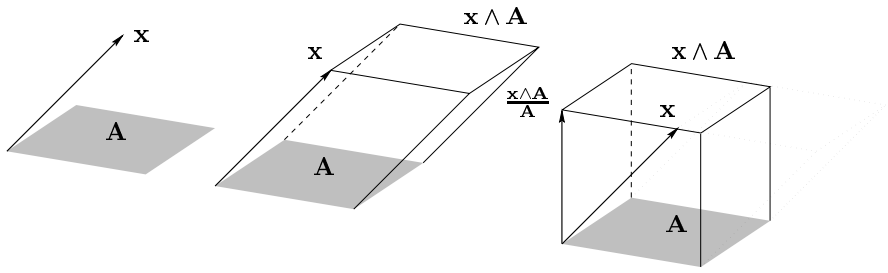


Fig. 2.2. The perpendicular component $\mathbf{x}_\perp$ of a vector $\mathbf{x}$ to a subspace characterized by a pseudoscalar $\mathbf{A}$: make the volume $\mathbf{x} \wedge \mathbf{A}$, straighten it in your mind (to view it as geometric product), then factor out $\mathbf{A}$ by division – but beware that division is not commutative, so compute it as $\mathbf{x}_\perp = (\mathbf{x} \wedge \mathbf{A})\mathbf{A}^{-1}$.

computation. This leads to new geometric and computational insights, and new methods, even for such basic constructions as, say, the intersection of two lines (section 2.6). The main consequence is that the use of geometric algebra makes our algorithms coordinate-free, valid in or extendible to n-dimensional spaces, and fully specific on direction parity (which is useful for consistent treatment of inside/outside, a famous issue in computer graphics).

A k-dimensional subspace of a vector space V^n is characterized by a *pseudoscalar* i, which is an outer product of k independent vectors in that subspace:

$$i = \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \cdots \wedge \mathbf{a}_k, \quad \text{with} \quad \mathbf{a}_i \in V^n. \tag{2.2}$$

Such a multivector is called *simple*; its magnitude is the directed volume spanned by the $\mathbf{a}_i$. The subspace spanned by i is denoted $\mathcal{G}(i)$. Eq.(2.2) explains the relevance of the outer product: it codifies 'linear (in)dependence' in an operational manner. The interaction of the *non-invertible* outer product with the *invertible* Clifford product produces compact notation and computation for algorithms involving *orthogonality*. For instance, the determination of a vector perpendicular to the subspace $\mathcal{G}(i)$ is:

$$\mathrm{P}_i^\perp(\mathbf{x}) \equiv (\mathbf{x} \wedge i)\, i^{-1}.$$

This leads to a compact and computable formulation of such algorithms as 'Gram-Schmidt orthogonalization'. Also, we can construct the *dual* $a^* \equiv aI^{-1}$ of a simple multivector a within a subspace I, and interpret it as the pseudoscalar of the subspace *perpendicular* to a in $\mathcal{G}(I)$.

These constructions have very intuitive geometrical interpretations (see figure 2.2); it is at this level that it becomes natural to call the Clifford product a *geometric product*.

– **'algebra of directions'**

Closely related to the above, but often used more qualitatively, is the idea of

union and intersection of subspaces to produce higher or lower dimensional subspaces. The operations that do this are known as the *join* and *meet* operations. They are a precise extension of set union and set intersection for directed subspaces; usually, they are treated modulo positive scalar factors since pseudoscalars signify a directed subspace modulo such a factor. We will argue in section 2.6 that there is a quantitative structure to these operations which is very useful in computations, since it determines how well-conditioned the operations are (similar to the use of the condition number of a matrix equation in numerical linear algebra) on the basis of 'distance measures' between the subspaces.
Join $\dot{\wedge}$ and meet $\vee$ of spaces are definable in terms of outer product and the contraction (or the inner product):

$$a \dot{\wedge} b = b \wedge a \quad \text{and} \quad a \vee_i b = (ai^{-1}) \rfloor b,$$

but their geometrical use involves some care, as we will see in section 2.6. Hestenes [4] pg. 19 calls this use of geometric algebra an 'algebra of directions', since the relationships between the pseudoscalars implement the lattice of k-dimensional directed subspaces of a vector space V_n.

– **geometric calculus**
Differentiation operators in geometric algebra are associated with multivectors; as a consequence, they have both properties of calculus and of geometry [4][2]. The geometrical properties need to obey the various product rules sketched before for multivectors; so differentiation with respect to a (multi)vector has commutation rules, decomposition rules, and orthogonality properties that fit the above scheme. This leads to a powerful calculus, which can usefully redo and extend the constructions of differential geometry. The popular differential forms, for instance, can be viewed advantageously within the more general framework of geometric algebra.

– **'split algebras'**
The above gives the framework of basic techniques in geometric algebra. This needs to be augmented by specific techniques for *mapping the geometric structure of an application* to a properly identified algebra. This is the *modeling* step, which is part of the application domain more than of the algebraic mathematics. It is of course highly important to applied computer science.
There is an important construction technique which brings some unification in these various required embeddings: Hestenes' *split* [7]. This is a technique in which the geometric algebra of an $(n+k)$-dimensional space is used to model the geometry of an n-dimensional space V^n. The split explicitly relates multivectors in the two spaces. The advantage is that the 'orthogonality algebra' and 'directed intersection algebra' of $\mathcal{C}\ell_{n+k}$ (which were developed for *homogeneous*, because simple, multivectors), now can describe the *non*-homogeneous quantities of projective and affine geometry in V^n (using a *projective split*) and of kinematics in V^n (using a *conformal*

split) [7]. Mathematically, the split makes an n-dimensional vector *isomorphic* to, say, an $n+1$-dimensional bivector. This is often denoted as '=' in literature. We will see in section 2.5 that to actually use the split in an implementation, it is more proper to be explicit about the mapping relating the elements of the algebras $\mathcal{C}\ell_{n+k}$ and $\mathcal{C}\ell_n$.

With these refinements of the monolithic term 'geometric algebra' into various levels of meaning and associated operators, we can better state its relevance to computer scientists needing to 'do' geometry.

> *Geometric algebra is a collection of computation rules and techniques relevant to doing computations in models of the geometric aspects of applications. Its structure contains several distinct but exactly related levels, each with its own syntax of operators, and an accompanying interpretation. A specific application will probably need them all; fortunately they are generic in their construction. It is thus advantageous to connect to this framework, both for unified theoretical developments and for the actual software performing the calculations.*

If our hopes come true, geometric algebra does away with the *internal interface problem* between geometric computational modules (typically arising when solving part of one's application by techniques of linear algebra using matrices, and then having to translate them to differential forms to treat other aspects, all proceeded by the projective geometry of processing and interpreting actual visual observations). It will replace all this with a common language in which all these specialized modules can communicate, and in which algorithms can be specified and developed. The *modeling problem* ('which geometric model for which application') remains, but the choices are limited (one of the Clifford algebras) and can all be implemented in advance, in a standard manner, with generic data structures. We can then focus on *what* we need to compute in our applications, rather than on *how* to compute it.

2.3 The contraction: a better inner product

The Clifford product is the unambiguous basis of all geometric algebra, and from it are constructed derived products which are useful for 'orthogonality algebra'. Using such products, we would expect to prove lemmas which are universally valid 'total identities', into which we can plug any element of the geometric algebra. The currently used inner product of [4], however, is riddled with lemmas containing conditions, mostly on the relative grade of its operands. These problems were recognized (see [4] pg.20), but not resolved until recently, when Lounesto [9] called attention to a powerful different way of introducing an inner-product-like operation into geometric algebra. He calls this the *contraction* and denotes by '⌋'; his suggestion does not seem

to have been followed in the applied literature. Yet the contraction may be a great improvement to geometric algebra, since it simplifies the algebraic structure without sacrificing any of the geometric meaning – as will now be shown.

Here is the definition. Assume that you have already defined the Clifford product based on a bilinear form $\langle , \rangle$ on vectors, and have based on that the outer product, as in eq.(2.1): by means of what it does on vectors, and demanding bilinearity and associativity. Now extend the bilinear form $\langle \, , \, \rangle$ to arbitrary multivectors, as follows.

1. For scalars:
$$\langle \alpha, \beta \rangle = \alpha\beta \quad \text{for } \alpha, \beta \in K. \tag{2.3}$$
2. For two multivectors of the form $a = \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \cdots \wedge \mathbf{a}_k$ and $b = \mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \cdots \wedge \mathbf{b}_\ell$:
$$\langle a, b \rangle = \begin{cases} \det\left(\langle \mathbf{a}_i, \mathbf{b}_j \rangle\right) & \text{if} \quad k = \ell \\ 0 & \text{if} \quad k \neq \ell \end{cases} \tag{2.4}$$
Here $(\langle \mathbf{a}_i, \mathbf{b}_j \rangle)$ denotes the matrix of which the (i,j)-th element equals $\langle \mathbf{a}_i, \mathbf{b}_j \rangle$; its determinant is just used as a convenient shorthand for the anti-symmetric construction of the bilinear form.
3. The bilinear form is to be linear in both arguments.

Note that this is symmetrical, i.e. $\langle a, b \rangle = \langle b, a \rangle$. As a consequence of the imposed orthogonality of this extended bilinear form, a set of equalities $\langle x, a \rangle = \langle b, a \rangle$ for all a in (a basis of) $\mathcal{C}\ell_n$ implies $x = b$.

With this bilinear form, define the contraction as *adjoint* to the outer product:
$$\langle u \rfloor v, w \rangle \equiv \langle v, \widetilde{u} \wedge w \rangle \quad \text{for all} \quad u, v, w \in \mathcal{C}\ell_n \tag{2.5}$$
(where the reversion $\widetilde{u}$ of u is used to absorb some inconvenient signs). Now one can prove the following properties (see also [9]):

(a) $\alpha \rfloor \beta = \alpha\beta, \quad \alpha \rfloor \mathbf{x} = \alpha\mathbf{x} \quad \text{and} \quad \mathbf{x} \rfloor \alpha = 0, \quad \text{for } \mathbf{x} \in V^n, \alpha, \beta \in K$

(b) $\mathbf{x} \rfloor \mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle \quad \text{for } \mathbf{x}, \mathbf{y} \in V^n$

(c) $\mathbf{x} \rfloor (u \wedge v) = (\mathbf{x} \rfloor u) \wedge v + \widehat{u} \wedge (\mathbf{x} \rfloor v) \quad \text{for } \mathbf{x} \in V^n, \quad u, v \in \mathcal{C}\ell_n$

(d) $(u \wedge v) \rfloor w = u \rfloor (v \rfloor w), \quad u, v, w \in \mathcal{C}\ell_n$

Property (a) shows that a contraction involving scalars is not symmetric, as opposed to their inner product for which [4] explicitly demands $\alpha \cdot \mathbf{x} = \mathbf{x} \cdot \alpha = 0$. Property (b) shows that for vectors the contraction corresponds with the inner product. Property (c) shows that it is like a derivation, and the common inner product satisfies it as well. Property (d) is a duality between outer product and contraction, valid for *all* multivectors; the corresponding statement for the inner product has much more limited validity (more about this below).

These properties, together with *linearity in both arguments*, are sufficient to compute the contraction of *any* two multivectors, more conveniently than by the formal definition eq.(2.5). An important difference with the inner product '$\cdot$' is that the contraction '$\rfloor$' is *not* symmetric in its arguments (property (a) is one example). This means that many of the useful constructions and proofs of [4] which use the inner product need to be redone. When we do so, we find that the *conditions* under which the proof worked for the inner product (for instance on the relative grades of arguments) are now elegantly absorbed in the contraction operator (outside the range of the conditions on $\cdot$, the expression with $\rfloor$ then automatically produces 0). Thus the useful results from [4], ch.1 and its sequels are not only 'rescued', but also expressed more concisely. And many results obtain an expanded range of validity, due to the nice algebraic properties of the new inner product. We give some examples.

Examples:

1. *Duality statements.* In [4](1-1.25b) we find for *homogeneous* multivectors a_r, b_s, c_t of grades r, s, t, respectively, the property:

$$a_r \cdot (b_s \cdot c_t) = (a_r \wedge b_s) \cdot c_t \quad \text{for} \quad r + s \leq t \text{ and } r, s \geq 0 \tag{2.6}$$

   With the contraction rather than the inner product, we can prove the much stronger:

$$u\rfloor(v\rfloor i) = (u \wedge v)\rfloor i, \quad \text{for} \quad u \in \mathcal{G}(i), \quad v \text{ arbitrary.} \tag{2.7}$$

   Here u and v are general (*not just homogeneous!*) multivectors, i is a pseudoscalar (and therefore homogeneous), and the only condition is that u is in the geometric algebra of the subspace with pseudoscalar i. Note that it is permitted to have v in a space exceeding $\mathcal{G}(i)$; if it is, both sides are 0 and hence still equal. Thus this structural property in 'interior/exterior algebra', and the algebras built on it, has a *much enlarged scope of validity*. Other duality statements from [4] generalize similarly, we will prove an example in section 2.4.2. The most extreme is property (d) above (which is similar to [4](1-2.17b), but now valid for *all* multivectors u, v, w).
2. *Expansion of geometric products.* In [4](1-1.63) we find for the expansion of a geometric product bivector $\mathbf{B}$ with a multivector u:

$$\mathbf{B}\,u = \mathbf{B} \cdot u + \tfrac{1}{2}(\mathbf{B}\,u - u\,\mathbf{B}) + \mathbf{B} \wedge u \quad \text{if} \quad \mathbf{B} = \langle \mathbf{B} \rangle_2 \text{ and } \langle u \rangle_1 = 0$$

   Note the demand $\langle u \rangle_1 = 0$: this formula does *not* work for vectors. So we have an identity of which the validity depends on the grade of an operand. In (subtle) contrast, using the contraction, we can prove:

$$\mathbf{B}\,u = \mathbf{B}\rfloor u + \tfrac{1}{2}(\mathbf{B}\,u - u\,\mathbf{B}) + \mathbf{B} \wedge u \quad \text{if} \quad \mathbf{B} = \langle \mathbf{B} \rangle_2,$$

   a formula that is now valid for *all* u, since $\mathbf{B}\rfloor u = 0$ for the scalar and vector parts of u. As before, the contraction operator automatically takes

care of the conditions. This formula is part of a series of expansion formulas, for a scalar α, vector $\mathbf{x}$ and bivector $\mathbf{B}$ we get:

$$\alpha\, u = \alpha \rfloor u = \alpha \wedge u, \quad \mathbf{x}\, u = \mathbf{x} \rfloor u + \mathbf{x} \wedge u$$

$$\mathbf{B}\, u = \mathbf{B} \rfloor u + \tfrac{1}{2}(\mathbf{B}\, u - u\, \mathbf{B}) + \mathbf{B} \wedge u.$$

Each higher order obtains one more term. In [4], the statement for bivectors takes the exception stated above, while that for scalars reads $\alpha\, u = \alpha \cdot u + \alpha \wedge u = \alpha \wedge u$ since $\alpha \cdot u = 0$ by definition.

3. *Etcetera...* Lack of space precludes more examples; but *all* geometric constructions from [4], chapter 1, can be reproduced using the contraction. This demonstrates that the contraction can also be used as a basis for a full geometric algebra.

In summary, an equally or more powerful structure is created by using $\rfloor$ rather than $\cdot$, *in which known results are simultaneously generalized and more simply expressible*, without conditional exceptions. This cleaner algebraic structure will lead lead to simpler geometric software, since no exception handling is required.[3]

2.4 The design of theorems and 'filters'

Since the contraction operator reduces conditions in expressions, it becomes possible to develop a technique for designing 'geometric filters', i.e. expressions in geometric algebra that perform certain desired tasks. I have dubbed this technique the 'index set method', since it designs the filters based on which independent orthogonal basis vectors (characterized by indices) occur in input and output of the filter. Such indices may get passed, they may be cancelled, or they may lead to a 0 result. For instance, in $\mathbf{e}_1\mathbf{e}_2$, both indices 1 and 2 occur in the result; in $\mathbf{e}_1\mathbf{e}_2\mathbf{e}_1 = -\mathbf{e}_1^2\mathbf{e}_2 = \alpha\, \mathbf{e}_2$, index 1 has been absorbed in a scalar α; and in $\mathbf{e}_1 \rfloor \mathbf{e}_2$, the combination of index 1 for the first argument and index 2 for the second results in 0 (remember that the $\mathbf{e}_i$ are orthogonal). We denote the index set of a simple multivector a by $\mathcal{I}(a)$. Despite the index-based nature of this procedure, linearity guarantees that the final results are coordinate free, independent of the basis on which they were derived.

Figure 2.3 and 2.4 present the different filters of two and three terms, using only geometric product, outer product and contraction between terms (some reduction of the full range was made using the symmetry of geometric product and outer product on index sets).

[3] Moreover, the extended bilinear form permits us to repair some inelegancies in the common definitions of basic concepts, such as the use of grade operators to define elementary concepts like the norm of a multivector u by $\langle \widetilde{u}\, u \rangle_0$. Having defined the extended bilinear form we can simply define: $|u|^2 = \langle u, u \rangle$; the same in value, but arguably more elegant.

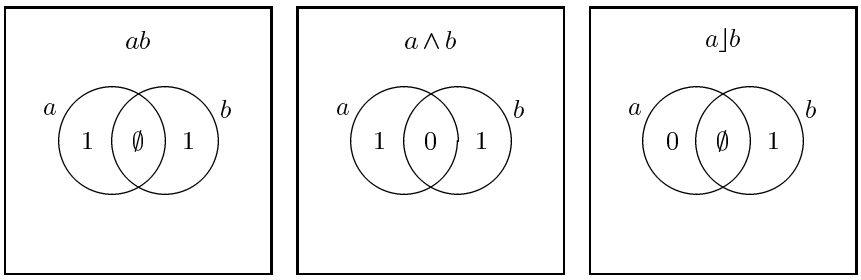


Fig. 2.3. The index sets of the basic products. Notation: '1' denotes that indices in this subset appear in the result, '0' denotes that indices appearing in this set make the whole result 0, and '∅' denotes that indices in this subset do not appear in the result (but neither do they make the result zero).

2.4.1 Proving theorems

We can prove identities and their conditions by the following method:

1. First assume that the arguments are simple multivectors.
2. Draw up the outcome diagrams of both sides (using figures 2.3 and 2.4 to compose them quickly).
3. Make a composite diagram retaining only those subsets in which no conflict exists between outcomes.
4. In the areas with outcome 0, the identity obviously (but rather trivially) holds. Construct general simple multivectors for each of the arguments, taking a representative from each non-zero area, taking care to satisfy the containment relationships of the diagram's construction in step 3. (Details below.)
5. Evaluate both sides of the identity for these sample multivectors. If it holds the identity has been proved for all simple multivectors; if it does not, this computation shows which scalar factor to introduce.
6. For those arguments in which the identity is linear, extend it to general multivectors, within the derived preconditions of step 3.

The method most clearly saves work in step 5, since the exceptional cases messing up the computations have already been taken out in steps 3 and 4. This is best illustrated by an example.

2.4.2 Example: proof of a duality

Let us investigate the validity of the identity $u \wedge (vw) = (u \rfloor v)w$, a form of duality between $\wedge$ and $\rfloor$.

1. We focus first on the identity for simple multivectors a, b, c, so on the identity $a \wedge (bc) = (a \rfloor b)c$.

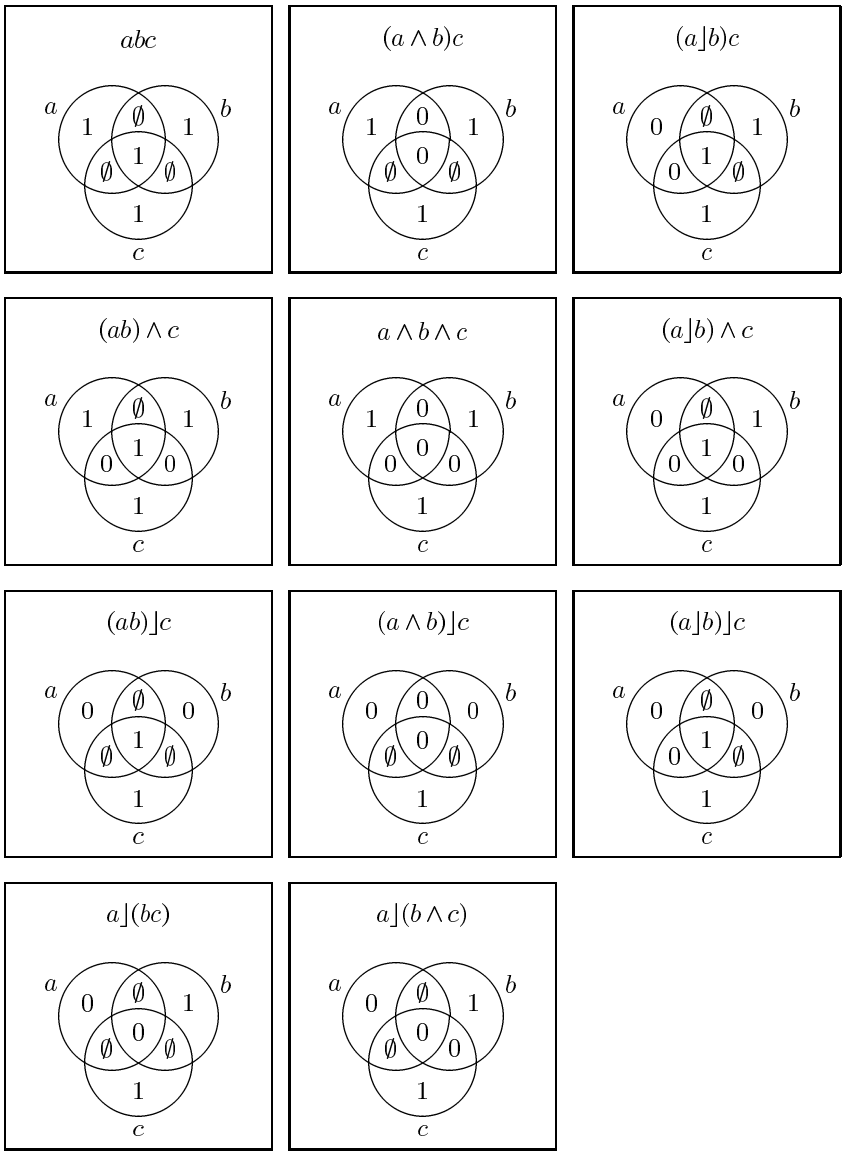
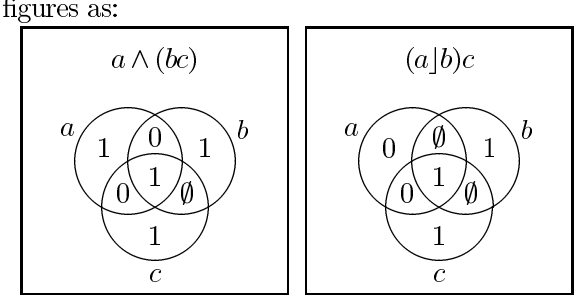
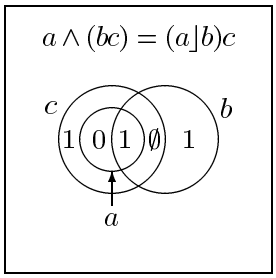


Fig. 2.4. The complete catalog of different three-term products. Notation as in figure 2.3.

2. The diagrams for both sides of the possible identity can be gleaned from the earlier figures as:



3. The composite diagram must contain the consistent parts of both. We observe that the parts where $\mathcal{I}(a) \not\subseteq \mathcal{I}(c)$ are not consistent, and therefore redraw the diagram to exclude this, noting the condition $\mathcal{I}(a) \subseteq \mathcal{I}(c)$:



4. To establish the full identity, we now have to inspect the scalar factors. To do so, take a representative simple multi-vector in each of the subsets of the diagram that do *not* lead to a zero result, to compose a typical example. In our diagram this implies, for instance: $a = a_k$, $b = \widetilde{a}_k b_\ell d_n$, $c = a_k \widetilde{d}_n c_m$, with $a_k = \mathbf{e}_1 \cdots \mathbf{e}_k$, $b_\ell = \mathbf{e}_{k+1} \cdots \mathbf{e}_{k+\ell}$, $c_m = \mathbf{e}_{k+\ell+1} \cdots \mathbf{e}_{k+\ell+m}$, $d_n = \mathbf{e}_{k+\ell+m+1} \cdots \mathbf{e}_{k+\ell+m+n}$; the index of a_k, b_ℓ, c_m, d_n indicates the grade, and the components of each are orthogonal basic vectors. The reversions in the expressions for b and c were merely put in for convenience in the computations below; since they only involve a scalar factor ± 1 on both sides of the identity, this is permitted.
5. With these sample multivectors, we obtain:

$$\begin{aligned} a \wedge (bc) &= a_k \wedge (\widetilde{a}_k b_\ell d_n a_k \widetilde{d}_n c_m) = a_k \wedge ((-1)^{kn} (d_n \widetilde{d}_n) \widetilde{a}_k b_\ell a_k c_m) \\ &= (-1)^{k(n+\ell)} (d_n \widetilde{d}_n)(\widetilde{a}_k a_k) a_k b_\ell c_m, \end{aligned}$$

and

$$\begin{aligned} (a\rfloor b)c &= (a_k \rfloor (\widetilde{a}_k b_\ell d_n)) a_k \widetilde{d}_n c_m = (a_k \widetilde{a}_k) b_\ell d_n a_k \widetilde{d}_n c_m \\ &= (-1)^{k(n+\ell)} (a_k \widetilde{a}_k)(d_n \widetilde{d}_n) a_k b_\ell c_m. \end{aligned}$$

This establishes that the two results are indeed identical under the condition found in step 3:

$$a \wedge (bc) = (a\rfloor b)c \quad \text{if} \quad \mathcal{I}(a) \subseteq \mathcal{I}(c). \tag{2.8}$$

6. The two sides in eq.(2.8) are linear in all arguments. The precondition assumes that all indices in $\mathcal{I}(a)$ are in $\mathcal{I}(c)$. The *simplest* linear extension is obtained by keeping c simple, making it in effect the pseudoscalar i of the space in which a and its linear extensions u reside. Then the precondition $\mathcal{I}(a) \subseteq \mathcal{I}(c)$ extends to $u \in \mathcal{G}(i)$. Thus we have proved an identity for two general multivectors u and v and a pseudoscalar i of the u-space:

$$u \wedge (v\, i) = (u \rfloor v) i \quad \text{if} \quad u \in \mathcal{G}(i) \tag{2.9}$$

   By carefully keeping track of indices, further extensions of the result for simple multivectors may be possible, but they are hard to phrase and are less useful because of that.

It should be clear that we can use the method also to come up with new theorems – this now becomes a routine exercise for any practitioner of geometric algebra (as it should be).

2.4.3 Filter design to specification

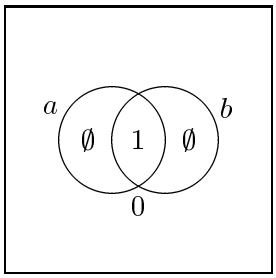
Since 'filters' are merely 'useful expressions', their method of design is very similar:

1. Focus first on simple multivectors.
2. Specify the desired outcome set in terms of a diagram.
3. Identify this diagram in an exhaustive table of outcomes (such as figure 2.3 or 2.4). It may be a *sub-diagram* of an entry.
4. Identify the conditions on the arguments that select this (sub-)diagram.
5. These conditions, applied to the equation defining the diagram, give the desired 'filter' expression.

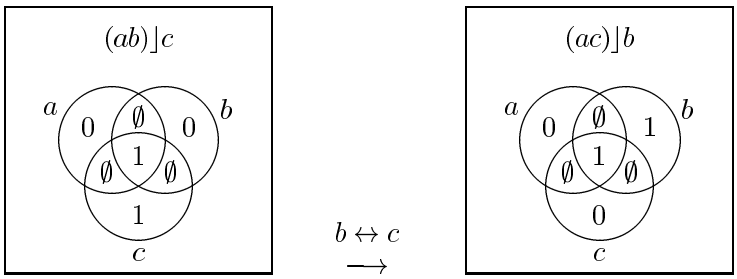
2.4.4 Example: the design of the meet operation

We illustrate the design of the important *directed intersection operator* in the 'algebra of directions'.

1. The desired outcome of the intersection on two index sets $\mathcal{I}(a)$ and $\mathcal{I}(b)$ is that indices 'pass' when they are in the intersection of the index sets, are indifferent when in either of them, and zero outside. This is thus:

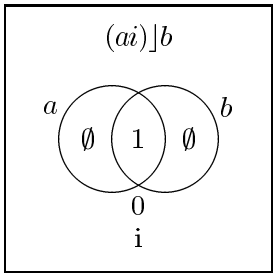


2. Such a filter cannot be made by simply combining the two index sets; all possibilities of that were indicated in figure 2.3, and it is not among them. Thus we look in the three-argument filters of figure 2.4. We find the desired possibility as a subset of the diagram of $(ab)\rfloor c$:



(We interchanged the dummy filter parameters b and c, to make the parameters and diagram correspond with our choice under step 1.)

3. ¿From this diagram, we produce the desired result by demanding: $\mathcal{I}(a) \subseteq \mathcal{I}(c)$ and $\mathcal{I}(b) \subseteq \mathcal{I}(c)$. So c must contain both a and b in this sense; the simplest is if c is a pseudoscalar for the space containing both a and c. The new diagram is:



4. This shows that a *non-trivial result* (i.e. non-zero) is only achieved when i is a pseudoscalar of the *smallest* space containing both a and b. Then $(ai)\rfloor b$ is a pseudoscalar for the subspace common to $\mathcal{G}^1(a)$ and $\mathcal{G}^1(b)$, since it contains only indices from $\mathcal{I}(a) \cap \mathcal{I}(b)$.

The operation we have constructed in this example is proportional to the *meet* of subspaces, conventionally defined by $a \vee_i b = (ai^{-1})\rfloor b$ (which differs by an admissible scalar sign from our filter); more about this, its geometrical interpretations and the importance of scalar factors in section 2.6.

2.5 Splitting algebras explicitly

As we stated in section 2.2, 'splitting' is a generic operation that helps in translating geometrical structures in an application to an appropriate Clifford

algebra. A split is based on the following fact: the space of k-vectors in a Clifford algebra $\mathcal{C}\!\ell_n$ contains $\binom{n}{k}$ elements. The identity

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k} \tag{2.10}$$

suggests that k-vectors of the Clifford algebra $\mathcal{C}\!\ell_{n+1}$ could be mapped onto $(k-1)$-vectors and k-vectors in the algebra $\mathcal{C}\!\ell_n$. We can indeed make this explicit. An important example is the split of a k-vector a of $\mathcal{C}\!\ell_{n+1}$ relative to a fixed vector e_0 in $\mathcal{C}\!\ell_{n+1}$, which can be considered as a decomposition according to the identity:

$$a = e_0^{-1} \wedge (e_0 \rfloor a) + e_0^{-1} \rfloor (e_0 \wedge a). \tag{2.11}$$

In this equation, $e_0 \rfloor a$ is a $(k-1)$ vector in $\mathcal{C}\!\ell_{n+1}$, which is moreover constrained to the n-dimensional subspace $\mathcal{G}(e_0{}^*)$, perpendicular to e_0 (Proof: $e_0 \rfloor (e_0 \rfloor a) = (e_0 \wedge e_0) \rfloor a = 0 \rfloor a = 0$). The second term $e_0^{-1} \rfloor (e_0 \wedge a)$ is a k-vector in this same subspace (Proof: $e_0 \rfloor (e_0^{-1} \rfloor (e_0 \wedge a)) = (e_0 \wedge e_0^{-1}) \rfloor (e_0 \wedge a) = 0$). Thus if we identify this subspace with the vector space generating the lower-dimensional Clifford algebra $\mathcal{C}\!\ell_n$, then we have explicitly constructed a mapping from $\mathcal{C}\!\ell_{n+1}$ onto $\mathcal{C}\!\ell_n$. We follow custom in denoting the elements of $\mathcal{C}\!\ell_n$ or its isomorphic subspace in **bold** font, the other elements of $\mathcal{C}\!\ell_{n+1}$ in normal *math* font.

Doing the split for $k = 1$, we see that a vector x of $\mathcal{C}\!\ell_{n+1}$ corresponds to a vector $\mathbf{x}$ in $\mathcal{C}\!\ell_n$ of the form $e_0^{-1} \rfloor (e_0 \wedge x)/\alpha$, with α a scalar or scalar-valued function.[4] This gives, conversely:

$$x = (e_0^{-1} \rfloor x) e_0 + \alpha \mathbf{x} = x_0 e_0 + \alpha \mathbf{x} \tag{2.12}$$

(defining $x_0 \equiv e_0^{-1} \rfloor x$). By choosing different α, we can implement different splits. A particularly useful way is the *projective split* obtained by setting $\alpha = x_0 = e_0^{-1} \rfloor x$. This gives:

$$x = x_0(e_0 + \mathbf{x}) \quad \text{and} \quad \mathbf{x} = e_0^{-1} \rfloor (\frac{e_0 \wedge x}{e_0^{-1} \rfloor x}) = e_0 \rfloor (\frac{e_0 \wedge x}{e_0 \rfloor x}). \tag{2.13}$$

In the projective split, any x representing a vector $\mathbf{x}$ can thus be written as a member of a 1-scalar-parameter family of representatives, namely as some scalar multiple of the vector $e_0 + \mathbf{x}$. It is called 'projective' since it is useful in doing projective geometry (see [7]), but it has other uses too. For some of those, we must perform the projective split embedding in a *canonical* way, representing $\mathbf{x}$ by $x = e_0 + \mathbf{x}$, taking the arbitrary constant x_0 in eq.(2.13) equal to 1; since $x_0 = e_0{}^{-1} \rfloor x$, we can view this as an embedding to the hyperplane $e_0{}^{-1} \rfloor x = 1$ in $\mathcal{C}\!\ell_{n+1}$. For some other embeddings, we require normalization of higher-order pseudoscalars.

[4] There is a second way to embed vectors, using $k = 2$: out of x, construct a bivector $e_0^{-1} \wedge x$, then map that according to the first term of eq.(2.11) as $e_0 \rfloor (e_0^{-1} \wedge x)$. The result is equal to $e_0^{-1} \rfloor (e_0 \wedge x)$, and thus the same; this second way may seem more indirect, but it is actually sounder algebraically, see [7].

The basic usefulness of the projective split is that the embedding of $\mathcal{C}\ell_n$ into $\mathcal{C}\ell_{n+1}$ simplifies computations. Take, for instance, the equation for the set of points on a directed line from $\mathbf{p}$ to $\mathbf{q}$ in V_n of $\mathcal{C}\ell_n$. This set is

$$\{\mathbf{x} \in V_n \mid (\mathbf{x}-\mathbf{p}) \wedge (\mathbf{q}-\mathbf{p}) = 0\}. \tag{2.14}$$

(for this is the set of all points $\mathbf{x}$ such that $\mathbf{x}-\mathbf{p}$ and $\mathbf{q}-\mathbf{p}$ have the same direction). Under the projective split, this set is represented by the bivector $\ell = p \wedge q$ (with p and q the representations of $\mathbf{p}$ and $\mathbf{q}$, respectively), in the sense that $\mathbf{x}$ is on the line iff its representative x is in the space of this pseudoscalar ℓ:

$$(\mathbf{x}-\mathbf{p}) \wedge (\mathbf{q}-\mathbf{p}) = 0 \quad \Longleftrightarrow \quad x \wedge (p \wedge q) = 0, \tag{2.15}$$

as is easily verified. This result extends to higher order affine linear subspaces: they are *simple* multivectors of $\mathcal{C}\ell_{n+1}$, to be interpreted by the projective split. It generalizes the 'trick' of *homogeneous coordinates* (well-known in applied linear algebra) from mere vectors to all of the higher-dimensional subspaces.

In the literature on projective splits (such as [7],[12]), the actual embedding is hidden in isomorphisms stating the *equivalence* of certain bivectors in $\mathcal{C}\ell_{n+1}$ and vectors in $\mathcal{C}\ell_n$. As a consequence, the grades of the various elements get confusing (a bivector can be 'equal' to a vector!), and the operation $\wedge$ acquires a meaning which subtly changes with the operands (if you write $\mathbf{x} = e_0 \wedge x$ and $\mathbf{y} = e_0 \wedge y$, then formally $\mathbf{x} \wedge \mathbf{y} = 0$ for any $\mathbf{x}$, $\mathbf{y}$!). The above shows that there is no need for this: we can make the mappings totally explicit. The resulting cleaner algebraic structure leads to cleaner software.[5]

2.6 The rich semantics of the meet operation

We are now ready to discuss the meet operation from the 'algebra of directions' in more detail and to apply it to the intersection of directed affine linear subspaces by combining it with the projective split interpretation.

In literature, the meet is often treated as a 'qualitative' operation. The reason is probably that its most useful application is when a and b in $a \vee_i b$ are pseudoscalars, and that these in turn have their most useful application when they are considered the representatives of affine subspaces in the projective split. Since the projective split contains an arbitrary scalar for the embedding (such as x_0 in eq.(2.13)), this then leads one to the neglect of all scalar factors (or, when done more carefully, all *positive* scalar factors) [12]. This qualitative approach, however, is also necessarily *binary*: subspaces either intersect or they don't, and there is no measure of the *relevance* of the intersection. This is a problem in applications where geometrical data

[5] This corresponds in spirit to Stolfi's careful treatment of the implementation of oriented projective geometry [14]. He also recommends the introduction of a 0 for every grade k (denoted $\mathbf{0}^k$), to make all theorems on grades universally valid.

has an associated *uncertainty*. For instance, when intersecting two observed planes that are almost co-directional, the location of the intersecting line is ill-determined and this should be expressed in the error margin; it may even require the observed planes to be considered as two observations of the same plane, making the intersection line physically meaningless. We thus need a way to express 'intersection strength' as well as the intersection result.

Traditionally, the 'meet' operation is just taken as providing the intersection subspace, and not the intersection strength. We now show that it can give both, with the *magnitude of the meet* giving such diverse measures of intersection strength as the *distance measure between subspaces* (known from numerical linear algebra), and even (in the explicit projective split interpretation of subspaces of $\mathcal{C}\ell_{n+1}$) the *Euclidean distance between non-intersecting affine subspaces* in $\mathcal{C}\ell_n$!

2.6.1 Meeting pseudoscalars

We first need to understand the meet in more detail, especially being more careful about scalar factors (including signs) than is common in literature.

For pseudoscalars a and b, the meet $a \vee_i b$ is a pseudoscalar of their intersection, with a sign and magnitude that depends on those of a, b and i. For the standard definition $a \vee_i b \equiv (ai^{-1}) \rfloor b$, involving the *inverse* of the pseudoscalar, this is as follows.

> *Let a and b be simple multivectors with a common factor c. Then defining i through:*
>
> $$i = (b\,c^{-1}) \wedge c \wedge (c^{-1} a) \tag{2.16}$$
>
> *we have:*
>
> $$a \vee_i b = c \tag{2.17}$$

(The proof is straightforward using the methods of section 2.4.1; the use of i^{-1} in the meet causes the somewhat unfortunate reversion of the arguments in the definition of i.)[6] Note that there is no such thing as '*the*' meet of a and b; replacing c by $-c$ gives an opposite sign. It is therefore necessary *always* to denote the pseudoscalar relative to which the meet is taken, and any suggestion that it can be omitted or defined objectively from a and b (such as found in [12]) is wrong.

There is less confusion about the *join* $\dot{\wedge}$ of two spaces, an operation that is used to give a pseudoscalar for the common subspace spanned by two pseudoscalars a and b. If a and b have *no common factors* (so the corresponding subspaces have only the point 0 in common), then the join is given by:

[6] The above can be used to correct an error in Pappas [12], who uses a pseudoscalar and decomposition that should have made his meet equal to $(-1)^{\mathrm{grade}(A')(\mathrm{grade}(C)+\mathrm{grade}(B'))} C^{-1}$ rather than C, in his notation.

$$a \dot{\wedge} b = b \wedge a \tag{2.18}$$

(the reversal of the arguments is done to prevent stray signs when using this in combination with the meet, and is again due to the use of i^{-1} in the definition of the meet.) The join is then a *directed union* of the subspaces.

If a and b *do* have a common subspace, then an objective definition of their join can *not* be given ([6],pg.34): there is an ambiguity of sign which can *not* be resolved explicitly, as in the case of the meet (see [14] for a clear explanation of this counterintuitive issue). Thus a directed union can then not exist, and eq.(2.18) correctly yields 0 (the only pseudoscalar representing a non-directed subspace).

For readers familiar with the wonderfully illustrated book on oriented projective geometry by Stolfi [14], note that his meet (denoted there by $\wedge_i$, presumably following [1]) differs from the above standard definition in geometric algebra. He defines it [14] pg. 50 (modulo a positive scalar) through equations which in our notation would effectively read:

$$a \wedge_i b = c \iff i = (ac^{-1}) \wedge c \wedge (c^{-1}b) \tag{2.19}$$

Thus Stolfi's meet $a \wedge_i b$ is identical to our meet $b \vee_i a$ (same i!) – and similarly, his join $a \vee b$ is identical to our join $b \dot{\wedge} a$. His delightful graphic constructions are therefore applicable to the 'algebra of directions' with a simple interchange of the operands.

2.6.2 Meets of affine subspaces

Affinely translated subspaces of $\mathcal{C}\ell_n$ are represented by pseudoscalars of $\mathcal{C}\ell_{n+1}$ under the projective split; the meet of these pseudoscalars can then be interpreted in terms of quantities of $\mathcal{C}\ell_n$ as signifying the directed intersection of affine subspaces.

When we compute the meet of two non-homogeneous linear subspaces of a space $\mathcal{G}(\mathbf{i})$, represented as $a = (e_0 + \mathbf{a}) \wedge \mathbf{A}$ and $b = (e_0 + \mathbf{b}) \wedge \mathbf{B}$ of $\mathcal{G}(e_0\mathbf{i})$ in the homogeneous projective split representation, we obtain:

$$\begin{aligned}
&a \vee_{e_0\mathbf{i}} b \\
&= \left(((e_0 + \mathbf{a}) \wedge \mathbf{A})\mathbf{i}^{-1}{e_0}^{-1}\right) \rfloor ((e_0 + \mathbf{b}) \wedge \mathbf{B}) \\
&= e_0(\mathbf{A} \vee_{\mathbf{i}} \mathbf{B}) + \left(\widehat{\mathbf{A}} \vee_{\widehat{\mathbf{i}}} (\mathbf{b} \wedge \mathbf{B}) + (\mathbf{a} \wedge \mathbf{A}) \vee_{\mathbf{i}} \mathbf{B}\right) \\
&= \left(e_0 \left(\widehat{\mathbf{A}} \vee_{\widehat{\mathbf{i}}} (\mathbf{b} \wedge \mathbf{B}) + (\mathbf{a} \wedge \mathbf{A}) \vee_{\mathbf{i}} \mathbf{B}\right) (\mathbf{A} \vee_{\mathbf{i}} \mathbf{B})^{-1}\right) \wedge (\mathbf{A} \vee_{\mathbf{i}} \mathbf{B}),
\end{aligned}$$

where the last step assumes that $(\mathbf{A} \vee_{\mathbf{i}} \mathbf{B})$ is invertible; which is the case if $\mathbf{i}$ is at most a pseudoscalar for the smallest common subspace of $\mathbf{A}$ and $\mathbf{B}$. Under this condition, the projective split interpretation of the result of the meet is thus an affine subspace with directional pseudoscalar $(\mathbf{A} \vee_{\mathbf{i}} \mathbf{B})$, translated over the position vector $\left(\widehat{\mathbf{A}} \vee_{\widehat{\mathbf{i}}} (\mathbf{b} \wedge \mathbf{B}) + (\mathbf{a} \wedge \mathbf{A}) \vee_{\mathbf{i}} \mathbf{B}\right) (\mathbf{A} \vee_{\mathbf{i}} \mathbf{B})^{-1}$.[7]

[7] These results correct an error in the published version of this paper in *Geometric Computing with Clifford Algebra*, eds. G. Sommer and E. Bayro-Corrochano,

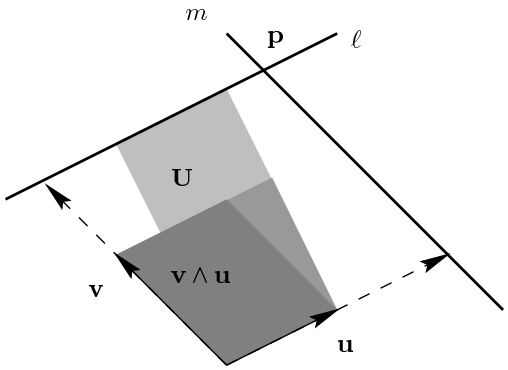


Fig. 2.5. The directed intersection of two lines in $\mathbb{R}^2$

Example: The directed intersection of two lines in $\mathbb{R}^2$: $\ell = e_0 \wedge \mathbf{u} + \mathbf{U}$ and $m = e_0 \wedge \mathbf{v} + \mathbf{V}$ in the projective split representation, see figure 2.5. We compute, with $\mathbf{i}$ taken as a pseudoscalar for $\mathbb{R}^2$:

$$\begin{aligned} p = \ell \vee_{e_0 \mathbf{i}} m &= -e_0(\hat{\mathbf{u}}^* \rfloor \mathbf{v}) + \hat{\mathbf{u}}^* \rfloor \mathbf{V} + \mathbf{U}^* \rfloor \mathbf{v} \\ &= e_0(\mathbf{v} \rfloor \mathbf{u}^*) - (\mathbf{u}^* \wedge \mathbf{V}^*)\mathbf{i} + \mathbf{U}^* \mathbf{v} \\ &= e_0(\mathbf{v} \wedge \mathbf{u})^* - \mathbf{V}^* \mathbf{u} + \mathbf{U}^* \mathbf{v}. \end{aligned}$$

Since $(\mathbf{v} \wedge \mathbf{u})^*$ is scalar, this corresponds to the intersection *point*:

$$\mathbf{p} = \frac{\mathbf{V}^*}{(\mathbf{u} \wedge \mathbf{v})^*} \mathbf{u} + \frac{\mathbf{U}^*}{(\mathbf{v} \wedge \mathbf{u})^*} \mathbf{v}$$

if $(\mathbf{v} \wedge \mathbf{u})^*$ is non-zero. Figure 2.5 graphically demonstrates the correctness of this result.

The geometric algebra framework validates the intersection results in any dimension, and in a computational representation that does not require exceptional data structures: points, lines, planes, etcetera are all admissible outcomes.

2.6.3 Scalar meets give distances between subspaces

We have seen in eq.(2.17) that the meet $a \vee_i b$ normally gives a *pseudoscalar* as its result, and that this is interpretable as the space of intersection of a and b. There is also an interpretation when the result of the meet is a *scalar*. The subspaces then intersect in the origin only; i.e. they are complementary in the smallest common space (with pseudoscalar i), though not necessarily orthogonal. When the meet is a scalar, we can rewrite it as:

Springer 2000; an error of simplification was made in the last two lines of the derivation of $a \vee_{e_0 \mathbf{i}} b$ above.

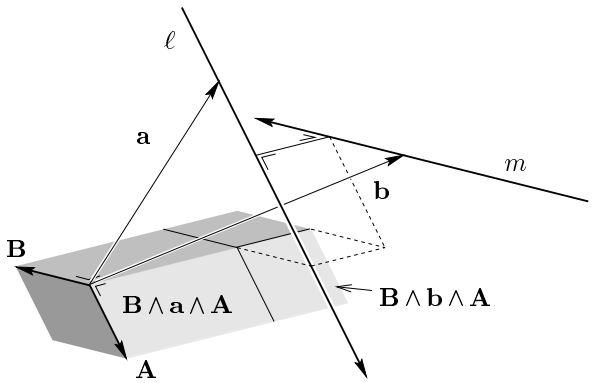


Fig. 2.6. The Euclidean distance between two affine subspaces in $\mathbb{R}^n$ whose ranks add to $n+1$.

$$a \vee_b = a^* \rfloor b = \langle a^* \rfloor b, 1 \rangle = \langle b, \widehat{a^*} \rangle = \langle \widehat{a^*}, b \rangle = \langle a^*, \widehat{b} \rangle = \langle b \rfloor a^*, 1 \rangle = b \rfloor a^* \\ = (b \wedge a)^*,$$

Thus in such a case, the meet equals the *volume* of the commonly spanned space, relative to the standard pseudoscalar i. This is a useful measure if we take all pseudoscalars involved (a, b and i) to be *unit* pseudoscalars. Then the meet varies continuously from 1 to -1, and is zero when the two subspaces have some subspace in common (they do *not* necessarily coincide: *any* common factor in a and b makes $b \wedge a$ equal to zero). We can interpret the values ± 1 as: the subspaces are *orthogonal* in the embedding space $\mathcal{G}^1(i)$. *The magnitude of a scalar meet of unit pseudoscalars is thus a measure for the 'parallelism' of the spaces they represent.*

> **Example:** Consider two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^2$ with pseudoscalar i. Then $\mathbf{x} \vee_i \mathbf{y} = (\mathbf{y} \wedge \mathbf{x})^* = (-|\mathbf{y}|\,|\mathbf{x}|\, i \sin\phi)\, i^{-1} = |\mathbf{x}|\,|\mathbf{y}| \sin\phi$, with ϕ the angle from $\mathbf{x}$ to $\mathbf{y}$ in i. If both are unit vectors this yields $\sin\phi$. Thus the meet has the largest absolute value, 1, when $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal* (with $+1$ when $\mathbf{y}$ is in the positive direction from $\mathbf{x}$, so $\mathbf{y} = \mathbf{x}i$, and -1 for the opposite direction), and goes *continuously to zero* when $\mathbf{x}$ and $\mathbf{y}$ become more and more parallel.

In linear algebra, a commonly used distance measure between subspaces is the sine of the angle between them, see e.g. [3]. It can be shown by generalization from the 1-dimensional example above that this is indeed what $(b \wedge a)^*$ is, for unit a, b and i. Thus the meet contains this common practice in numerical linear algebra, casting an interesting light on its essential nature.

2.6.4 The Euclidean distance between affine subspaces

If we are in a projective split representation, a and b in $\mathcal{C}\ell_{n+1}$ represent affinely translated linear subspaces of $\mathcal{C}\ell_n$. If these subspaces are complementary in

$\mathcal{C}\ell_{n+1}$ as in the previous section, then their meet is scalar; this complementarity means their ranks in $\mathcal{C}\ell_n$ add up to $n+1$. For two such spaces (a point and a line in 2-D, two lines in 3-D) we can define their (directed) distance in $\mathcal{C}\ell_n$ as the length of the (directed) mutual perpendicular connecting them. This turns out to be proportional to *the meet of their projective split representatives*!

Let us write the affine subspace represented by the pseudoscalar a as the translation by a vector $\mathbf{a}$ of a subspace with unit pseudoscalar $\mathbf{A}$, i.e. the set $\{(\mathbf{x}-\mathbf{a})\wedge\mathbf{A}=0\}$. Its canonical projective split representation is $a=(e_0+\mathbf{a})\wedge\mathbf{A}$, and similarly for b we have $b=(e_0+\mathbf{b})\wedge\mathbf{B}$. Let $\mathbf{i}$ be the unit pseudoscalar of $\mathcal{C}\ell_n$, and $I=e_0\mathbf{i}$ the pseudoscalar for $\mathcal{C}\ell_{n+1}$ (note the order!). Then with the assumed complementarity of a and b in I, their meet relative to I is:

$$\begin{aligned} a\vee_I b=(b\wedge a)I^{-1} &= ((e_0+\mathbf{b})\wedge\mathbf{B}\wedge(e_0+\mathbf{a})\wedge\mathbf{A})I^{-1} \\ &= (e_0\wedge\mathbf{B}\wedge\mathbf{a}\wedge\mathbf{A}+\mathbf{b}\wedge\mathbf{B}\wedge e_0\wedge\mathbf{A})I^{-1} \\ &= e_0(\mathbf{B}\wedge\mathbf{a}\wedge\mathbf{A}-\mathbf{B}\wedge\mathbf{b}\wedge\mathbf{A})I^{-1} \\ &= (\mathbf{B}\wedge\mathbf{a}\wedge\mathbf{A}-\mathbf{B}\wedge\mathbf{b}\wedge\mathbf{A})\mathbf{i}^{-1} \end{aligned} \tag{2.20}$$

This is a quantity that is entirely computable in $\mathcal{C}\ell_n$. It is proportional to the *orthogonal directed Euclidean distance* between the two subspaces represented by a and b, by a proportionality factor of $(\mathbf{B}\wedge\mathbf{A})\mathbf{i}^{-1}$ (which is the 'distance' between the directional elements in the sense of section 2.6.3). This is depicted in figure 2.6: the expression in brackets is a difference of two volumes in **i**-space, which can be viewed as being spanned by $\mathbf{B}$, $\mathbf{A}$ and a vector in the direction of their perpendicular connection; the difference relative to is the directed length of this vector.

We thus find yet another classical distance measure embedded in the intersection operation of geometric algebra. Note that eq.(2.20) is valid in any finite-dimensional space, and coordinate free. It is thus well-suited for implementation in a generic geometric software package!

2.6.5 The use of geometric algebra

The meet was designed as a straightforward 'directed intersection operation' for geometric algebra in section 2.6. The examples show that it has a semantics that depends on the modeling step which translates an application to appropriate geometric algebra. This is an instance of an important principle: there is no *unique* interpretation of Clifford algebra or geometric algebra.[8]

This is not a weakness of geometric algebra, but rather a sign of its strength: a limited number of generic constructions in the mathematically consistent theory of geometric algebra suffices to implement what used to

[8] Hestenes frequently points out that the use of a bilinear form in the Clifford algebra does not automatically imply that it *only* applies to metric spaces: it all depends on how you use it.

be seen as disparate geometrical tricks in different applications. If you view geometric algebra as giving an exhaustive library of advanced computational techniques, then once you have made the mapping between your application and geometric algebra, the meaning of these techniques is automatic, and gives a complete set of internally consistent operators in the application. This seeming restriction will prevent you from going astray (not just anything is permitted) and can help to inspire you (since it gives advanced and consistent constructive techniques). Also, since these can be defined in generic terms of Clifford algebra, they need only be implemented once – the only responsibility of the practising computer scientist is then the explicit implementation of the mapping between the application and this generic body. After that, computations are automatic.

Having said that, we need to show that the library of techniques in geometric algebra is indeed sufficient for such purposes, and extend it where possible. The challenge is not necessarily to do new things using geometric algebra (though that is always nice!), but rather to show that a single framework encompasses all previously known results, and does so compactly. To mathematicians, this is not a very exciting task; to computer scientists and physicists, its completion would be immensely gratifying. It would give us a box of integrated geometrical power tools which we could use to perceive, describe and direct objects in the world without being hampered by interface problems between incompatible sets of mathematical instruments (as we now so often are).

2.7 Geometrical models of multivectors

We have seen how we could understand some of the formulas coming out of our 'meet' computations by drawing a picture of the situation, and representing the multivectors involved by directed lines, directed areas, directed volumes, etc. Once you have done this for a while, you will find that this tends to reverse: the pictures soon become a natural construction tool for the design of formulas and algorithms. Unfortunately, few authors using geometric algebra appear to find a need for such pictorial explanations and constructions (an exception is [5]). Why? Any explanation of a powerful framework for 'doing geometry' that does not contain pictures must be less than convincing to the intended audience! In my experience, pictorial constructions such as figure 2.2 immediately instill a desire to learn more about geometric algebra in an audience of novices, and they are therefore immensely helpful.

It may indeed be possible to give a proper grammar for the construction of these pictures, which would turn this into a sound design procedure, and one that could be taught to the graphically inclined. There is some work to be done, though, to find a proper pictorial model: are vectors better viewed as emanating from the origin (i.e. as positions), or should we treat them as 'free vectors' (i.e. as directions)? Should we represent a bivector as a reshapeable

homogeneous plane element of a certain magnitude (as in [10]) or as a stack of planes with a stacking density (as in [11])? Do the answers depend on the 'model' of the geometry we are using (e.g. the 'free vector' image for affine directions, the 'fixed vectors' for their projective split representation)?

Whatever the answers, they are worth some research: the use of pictorial representations by proponents of differential forms (e.g. [11]) has helped them in 'spreading the faith', since the pictures effectively convey the intuition behind the computations and instill confidence in their consistency. Geometric algebra and geometric calculus could and should use a similar route to speedy introduction to a wider audience.

2.8 Conclusions

Clifford algebra is not useful by itself; it is just a consistent mathematical structure. Its surprising power comes from the discovery that this structure can be used to represent very many geometrical phenomena; indeed, maybe even *all* of geometry. It does so in *geometric algebra* which reorganizes the structure of Clifford algebra at various levels, guided by geometrical significance (see figure 2.1). This provides a framework that is immediately computational, rather than an arcane abstraction (not to be confused with algebraic geometry!). This has clear advantages: it *unites* geometry, and this is very important to the computer sciences, for a unified framework minimizes conversion between modules. It also gives a richer conceptual structure to design geometric algorithms, mainly since we do not need to express everything in terms of vectors (or, worse, coordinates) before we can make it computable. This makes advanced geometrical techniques more accessible to non-geometers.

In this contribution I argued that the user-oriented development of geometric algebra requires some new approaches, or changes in emphasis:

- it is insightful to the novice to convey explicitly the ordering of geometric ideas involved in turning Clifford algebra into geometric algebra (section 2.2);
- the algebraic structure of geometric algebra should be cleaned up to make operators operand-independent; we demonstrated this principle in the substitution of the contraction for the inner product (section 2.3), and in the totally explicit formulation of the mapping implementing the projective split isomorphism (section 2.5)
- we need a convenient design strategy to construct geometric 'filters' tuned to specific purposes, empowering the users to develop their own 'theory' as needed (section 2.4);
- we need to map traditionally useful concepts to their counterparts in geometrical algebra; and conversely, we should interpret the basic operators in geometric algebra in classical terms (section 2.6)

– it would be helpful to have a standardized pictorial representation of the basic concepts (section 2.7)

A lot of the work that has been done in geometric algebra is immediately relevant to these goals; notably the work of Hestenes and his followers, who have focussed on spreading the faith among physicists and mathematicians. Similar work now needs to be undertaken to promote its application to the geometrical issues in such computer sciences as vision, graphics and robotics.

References

1. M. Barnabei, A. Brini, G-C. Rota, *On the exterior calculus of invariant theory*, J. of Algebra, vol. 96, 1985, pp. 120–160.
2. C. Doran, A. Lasenby, S. Gull, *Chapter 6: Linear Algebra*, in: Clifford (Geometric) Algebras with applications in physics, mathematics and engineering, W.E. Baylis (ed.), Birkhäuser, Boston, 1996.
3. G.H. Golub, C.F. van Loan, *Matrix computations*, Johns Hopkins University Press, 1983.
4. D. Hestenes and G Sobczyk, *Clifford algebra to geometric calculus*, D. Reidel, Dordrecht, 1984.
5. D. Hestenes, *New foundations for classical mechanics*, D. Reidel, Dordrecht, 1985.
6. D. Hestenes and R. Ziegler, *Projective geometry with Clifford algebra*, Acta Applicandae Mathematicae 23: 25-63, 1991.
7. D. Hestenes, *The design of linear algebra and geometry*, Acta Applicandae Mathematicae 23: 65-93, 1991.
8. J. Lasenby, W. J. Fitzgerald, C. J. L. Doran and A. N. Lasenby. New Geometric Methods for Computer Vision Int. J. Comp. Vision 36(3), p. 191-213 (1998).
9. P. Lounesto, *Marcel Riesz's work on Clifford algebras*, in: Clifford numbers and spinors, E.F. Bolinder and P. Lounesto, eds., Kluwer Academic Publishers, 1993, pp. 119–241.
10. P. Lounesto, *Clifford algebras and spinors*, London Mathematical Society Lecture Note Series 239, Cambridge University Press, 1997.
11. C.W. Misner, K.S. Thorne, J.A. Wheeler, *Gravitation*, W.H. Freeman, New York, 1973.
12. R.C. Pappas, *Oriented projective geometry with Clifford algebra* in: R. Abłamowicz, P. Lounesto, J.M. Parra, *Clifford algebras with numeric and symbolic computations*, Birkhäuser, Boston, 1996, pp. 233-250.
13. M. Riesz, *Clifford numbers and spinors*, E.F. Bolinder and P. Lounesto, eds., Kluwer Academic Publishers, 1993.
14. J. Stolfi, *Oriented projective geometry*, Academic Press, 1991.