DigiPen-CS393: Geometric Algebra Independent Study  
Ari Surprise - Spring 2023 (Faculty mentor: Prof. Herron)

Purpose  
At the start of this term, I set out to improve upon developing an understanding for Geometric Algebra: I'd heard a number of accolades as to what it can do, how it's easier to understand than many other comparable, modern approaches: most famously touting we should eliminate quaternions from modern engines in favor of these objects called rotors and bivectors. It's claimed to be more computationally efficient than conventional solutions like Linear Algebra and matrix math at performing those transformations: those having n^2 members to do multiplication and addition for each row and column, and being able to skew, shear and scale space undesirably, it seemed reasonable that there could be a form of computation which avoided some of those possiblities and optimized the process. It claims it can deliver on a number of applications amidst all this, and mitigate interpolation woes (which matrix math is famously bad at handling, to be fair), and do all of these things with n-dimensional solutions, solving many bottlenecks to applications and opening avenues to new ones: I see applications claiming to be demonstrating it showcasing transformations with these improved interpolations and purported framerate improvements with those advantages, I see collision detection, intersection and resolution which claims to be coordinate free and branchless, I see mesh hull computation algorithms being done, seemingly in realtime, ik solvers, etc: the list goes on and this is honestly only one or two contributors' applications coming to mind. It also claims to take a great deal of physics and other fields, and unify and simplify their equations using this new way of re-contextualizing them.

If this is a fad which doesn't prove to have staying power or live up to these claims (through whatever mitigating terms and conditions), I wanted to appraise the field to at least know its limits to know when it isn't applicable as relevant, should someone suggest it (a soon to be graduating student, it seemed the perfect opportunity to tackle it). If however, it *could* serve as any level of replacement for everything I've spent the last several years learning, regarding how to use Linear Algebra to transform geometry, I certainly couldn't *afford* to *not* study it: so better sooner than later. I'm used to seeing a large degree of skepticism and pessimism at new things which buck the trends, before they prove themselves, but over the years I've been studying (a relatively small timeframe to sample, I realize), I've seen the resources to learn it grow, and the number of applications proudly using it (optionally), likewise grow.

I'm also happy to say after this semester that most of what I had to study in the basics is hardly the bucking of trends, and far more to do with recontextualizing what we already largely know and are familiar with, using a lot of new nomenclature, and so is a language barrier to sift through, but that shift in framing serves to unify many fields like the claims say, and like any language, it's a barrier worth overcoming to learn if it can connect you to new communities, or enrich your schemata and deepen your understanding of things, even the ones you already thought you knew well enough. As such, this turned out to be more of a language class than a whole lot of new math per se. The lessons I have to give insofar as what I learned in covering the basics are, consequentially, mostly vocabulary. I do feel it's worthwhile recontextualization of material to learn the lingustic changes: it has deepend my intuitive understanding of subjects like quaternions, rather than the claims that it might 'displace' them, and moreover, this field is a very deep rabbithole with some very studied professionals doing a number of things with it. While those graduate, post-graduate and doctorate level studies are often beyond my abilities as an undergrad, I feel far more empowered to understand a vast wealth of subjects more easily now than I had imagined before studying it. I do expect covering these materials to be worth the time to cross that lingustic hurdle presented.

# Annotated Glossary

Geometric Algebra comes in many flavors, and the term in itself is intended to be an umbrella term for each of those categories. For example, the term itself may refer to the most basic, entry level forms of geometric algebra over a vector space (aka VGA), or GA may refer to that umbrella term instead of simply VGA. While learning the terms there will be some ambiguity, both for lack of an authoratative standard (people studying it, apt to formalize their own, which make the most sense to their needs and applications), and a hazard for conflicts arising between those standards' terms.

I do my best to address that ambiguity as much as I have found in my research over the semester, and detail what data provenance I can here, in terms of the origins. Where glossaries might come at the end of most documents, here it makes more sense to lead with the terms early in on, in order to be able to succinctly articulate the more novel concepts, and recontextualize otherwise familiar ones. To ensure common ground, some basic terms are given to describe mathematical literacy expected (assuming most arithmetic & algebra known, unless relevant as later becoming exceptional). Perhaps banal to rehash basics, but I re-introduce terms only in light of new context.

Operator: a function mapping elements in a space to produce elements of another space, usually a common enough arithmetic process to get a symbol to succinctly refer to the process, the process carried out under an operator being an operation (inclusive of the elements being mapped into the process). We will certainly need to be defining a number of new operators for the purposes here.

There are several places that this creates conflicts, and what operator to use for which operation becomes more subjective and a matter of personal taste regarding ambiguity and legibility. Because the field is burgeoning, it is not yet standardized, and so people create their documents with the slant which focuses on the operations most relevant to their work as adequately distinct. There aren't realistically enough operators to go around before conflicts come up: it's inevitable. I hope to cover as many of these conflicts as possible, but also can only cover conventions research I've come across used, so this can't be considered comprehensive. Still, it's best to address and be up front about, for further study and records, to lend what insight and guidance I might offer.

Precedence: Conventions to evaluate several operations with consistent results. Some must be performed categorically throughout, before others, to achieve this: sufficing to say PEMDAS holds. In terms of adding new operators, this is another layer of concern: many new operators listed don't carry explicitly, or commonly known orders of operations. In my practices, I have had few issues with the added operators being ambiguous, but many programming languages offer operator overloading in a limited capacity, and especially there precedence can be a very relevant issue.

Few languages which offer more than a mapping of a list of pre-existing operators with pre-defined precedence lists to newly defined callback functions when given the right operands. Some of those operators may carry a visual facsimilie to the new operators introduced, or interesting parallels to the functions of those operators in other contexts, but if precedence of those operators is not considered, it can present other issues of confusing results. Wedge (∧) for example looks like (^), and the latter is available to type on a keyboard / as an operator. Most languages use ^ as bitwise xor, which comes very low on the precedence list though, as it is more conventional to use on mask in / out of values within a given binary range, after other operations. Using ^ for a wedge, "A ^ B" has visual similarity with sources using "A ∧ B", but will not have the expected results without a great deal of parenthesis to correct for the order of operations.

In a field still developing a great deal of applications, most of the contributors are mathemeticians and physicists (though applications for the math do cross a strikingly wide range of disciplines). Still lacking rigidly formalized patterns, these translation woes are not always the foremost concerns over being read consistently and applied unambiguously: limitations of code languages are frequently held as inconvent contrivances. Still noteworthy to highlight, given the the ripple effects of a standard, as well as the tools available to computer science forcing us all to conform to those limitations to the extent code is used to describe, perform, or even understand math.

Product: I won't detail every arithmetic operation, but GA entails a number of multiplications. As such, the symbol (\*) is to imply the general product of two objects, multiplied together (or "a times b"). Other symbols like (⨯) or such will have other meanings, which may be familiar to some.  
I.e. a \* 4 = a + a + a + a, or a \* b = b added to itself a times = a added to itself b times. Also notably, it may be common to find that ab = a \* b in many places, as a shorthand.

Associativity: The property that operational quantities may be evaluated in an equation from left to right (under operators of equal precedence), without affecting precedence / results.  
I.e. a + b + c = a + (b + c) = (a + b) + c, but a \* b + c ≠ a \* (b + c).

Commutivity: The property that operational quantities may be swapped / transposed from left to right in an equation (under equal precedence operators), without affecting precedence / results.  
I.e. a + b = b + a, or a + b \* c + d = c \* b + d + a, but a + b \* c + d ≠ a \* c + b + d.

Distributivity: The property that operational quantities distribute across / into a set of lower precedence operations equally to achieve the same results. I.e. a(b + c) = a \* (b + c) = a \* b + a \* c,  
or a + b \* (c + d) = a + b \* c + b \* d, but a + b \* (c + d) ≠ (a + b) \* c + (a + b)\* d

Sets: Any collection of elements, objects (collections of elements), functions, operations, names, sets, etc. May be mathematical terms, or not. GA must use set theory insofar as subsets (partial collections of larger sets), supersets (inversely, coming from a subset to a larger set), intersections (partial or full sets defined from multiple sets where the component sets both contain the elements), and unions (where those component sets define the new set if either contain the elements). Being comfortable with these terms should suffice to follow these materials.

Vector math needs to use subsets and supersets, especially to discuss how new vs. old dimensional shifts relate, as to how the elements of one dimension relate to the other. Some GA operations also give new insight on geometric interpretations of unions and interscetions in relation to their elements as dimensions intersect and how those interactions convolute / simplify as a shared set.

Groups: Terms relating to set theory over operations, insofar as our concerns, a field generally has possible inputs matching possible outputs (I.e. rational numbers are a closed field under arithmetic operations; won't produce irrational, complex numbers, etc), rings generalize to unrestrict fields from needing to adhere to commutivity and multiplicative inverses, and groups generalize to say that the operation is associative, every element has an inverse and a common identity element exists for any element and that inverse. More could (and arguably should), be said regarding group theory, but this could be quite long winded and a subject in itself depending on the scope of operations. More rigorous definitions than that for distinguishing between these terms and their qualifiying criterion compared to the others aren't by and large going to be deeply important, but there are very clear parallels of concerns to code and function notation.

It becomes important to note these concepts in set theory when expanding the number range to include multidimensional algebra, especially n-dimensional ones, adding new operations that need to be implemented, etc. We need to revisit if these properties hold under an operation in more simple concrete terms insofar as, say, determining a function's return type, being able to perform unit tests to confirm the operation conforming to expectations, etc. It isn't always important to keep at the fore of concerns, but it can be relevant to remember if an operation isn't a field, what subset, superset or disjunctive set it might map things to, etc. This is afterall why exponentiation is not a basic operator we keep available, to the consternation of new CS students everywhere.

One could define exponentiation of the field for the naturals or wholes over the wholes, but not integers since they expand to rationals and change our numeric modeling, and rationals ironically can't map to a field either (but we settle and call it good enough to approximate: very tangentially, I'm unconvinced fixed point numbers & radicals don't just deserve their own classes myself, and I've encountered far more people interested in numeric modeling, and expressing specificic dissatisfaction with the peculiarities of IEEE floats, comparable to my own, active in this field than any other study I've taken up so far). It's just important to note the differences in terms of why our numbers are working the way they are, to be able to better know what we can expect from them.

Vector: A quantity along one or more dimensions which consequentially carries both magnitude and direction along those dimensions. More generally, we may prefer 'orientation' to direction, as this continues to better describe a higher dimensional host of objects' directions should they be non-linear (as is typical, past conventional vectors). Still, direction is not wrong to use either. Some GA flavors will work with different geometric interpretations of a vector, so while it is good to have a familiarity with the concept of a visualization of vectors as arrows showing displacement, and starting with that understanding is fine, flexibility towards that interpretation will be required.

Scalar: Any numeric quantity for which terms are reducable to comparable real numbers, having only magnitude, but no direction (as distinct from vectors which have both). Often synonymous with a real number, but given the label to set a number apart, as a distinctly non-vector quantity.

Vector Space: Defined most centrally here as a set that is closed under addition, scalar multiplication, and containing a unique null element. It can be more of a semantic issue to pull apart in terms of one space versus another or a shared one in GA, and much more could be said regarding vector and scalar axioms, but further stipulations need not be given to suffice here.

Binomial coefficient: Combinatorics, probability and statistics denote the count of possible of ways to choose k items from larger set of n items. Pascal's triangle demonstrates . Abstract geometric math often entails combinatorics in weighting, and GA lends strong contextual insight into that relevance. Consequentially, it will help a great deal to already at least be very comfortable with Pascal numbers on a recursive basis of calculation. Being familiar with the closed form and logic language is good for following proofs with deeper takeaways, but isn't strictly necessary.

Span, Linear Indepencence: Vector groups are said to be linearly independent when no vector or combination of vectors in the group is representable as a scalar multiple of another. Spanning means a vector group contains every element of vector space over a given dimension, and is said to occur for a dimension equal to the number of linearly independent vectors in that group. GA operations maintain linear independence among other properties, and do much more to make sub-operations / state for linear dependence, significantly simplifying those tests.

Orthogonality: When an element stands perfectly perpendicular to all other basis elements. We will be defining elgebraic constants to stand in for a new enumeration of basis elements, as well as some new layers of thinking of orthogonality here, but the same linear algebra based trigonometric tests and logic apply. That is a basis element is orthogonal when it has a cosine of 0 compared with the other elements (which can continue to be determined easily through vector dot products). The decomposition of parallel and perpendicular elements intrinsic to the algebra is really so much of what makes this path unifying across so many fields.

Basis: A set of linearly independent vectors such that V vectors have a span of V. An orthonormal basis additionally guarantees that each vector is length 1 and orthogonal to all others. GA gets to guarantee orthogonality because its operations can't distort space (at least away from unity: that is, beyond inversion), and normalization is emphazised more in basis elements than coefficients.

Tensor: Broadly defined as mathematical objects used to describe physical properties, arguably few things in math are not tensors, if we wanted to be very tongue in cheek about it. I know of many useful definitions of tensors, and many places where GA communities find it relevant to discuss vectors as subsets of tensors. I will not be so bold as to try to narrow down what exactly a tensor is / isn't, or which definition is the most helpful or accurate, but it helps to have encountered the term / concept. I find it helpful to think of a tensor in terms of the tensor product, and how once entangled (as with quantum states), the geometric representation of that state can be it's own thing, difficult to pull apart again as component pieces, such that it is better to think of that product as a new related object, unto itself: the elements defining it bundled together as state co-effectors.

For these kinds of reasons, you are likely to read complaints in introductory GA materials, wherever one uses oversimplifications of vectors to regard them as lists of numbers which can be pulled apart or manipulated individually. This is largely also why memberwise multiplication is a discouraged definition of any manner of vector product, in GA or beyond. It can be a bit like walking on eggshells to get anyone to give useful definitions due to the foundational nature though.

You certainly don't need to have a firm grasp on what exactly a tensor is to work with or in GA, but it will be a theme which runs throughout it, to run into discussions on what one is. The deeper down the rabbithole of materials, sciences and maths you go using it, the more you will invariably gain a literacy in it also, by exposure, gradually. This basic terming however should suffice to start, and lend credence as to why precise yet broad definitions can be important or relevant.

Wedge: Given under the operator (∧). The first exterior product defined, built upon a foundation of complex number theory, from a geometric lens: this is the product of the exterior amount or 'rejection', orthogonal / perpendicular to any parallel source elements' for any dimensional space. This was the foundation for all GA, originally defined in the 1840's by Gustav Grassman, "Exterior Algebra" or "Grassman Algebra" referring primarily to the definition of this product type. Retaining only that geometric context, we can define the foundational clause for any vector A, that A ∧ A ≔ 0 (read "A wedge A"), which should be intuitive as there can be no quantity of an object which is exterior to itself. From a vector space perspective however, it can also be reasoned the span of a set of vectors is the number of linearly independent vectors described, but the span of 2 linearly dependent vectors is 0. From there and the assumption that distributivity holds, we find:  
(A + B) ∧ (A + B) = A ∧ A + A ∧ B + B ∧ A + B ∧ B = 0 + A ∧ B + B ∧ A + 0 = A ∧ B + B ∧ A.

Dot: Given under the operator (⋅). While A ∧ A + B ∧ B = 0, more generally A \* A and B \* B need not: they have no exterior, vector quantity to their own dimensions, but they may have a scalar amount. By contrast to a wedge product, the dot product defines the scalar or 'inner' product of a set of vectors. Geometrically this represents the 'projection' of the two vectors down into their shared quantities in the parallel space between them. It's useful to think of the dot and wedge products together, hence 'inner' and 'outer' suggesting part of a whole 'geometric product' as their sum. It has many useful properties such as capturing the magnitude of the vector for any given dimension, where for a vector A, squares of all real elements within A squared (I.e. "A ⋅ A") is the squared scalar, euclidean length ≔ (norm(A) \* norm(A)), so √(A ⋅ A) = magnitude(A) = norm(A).

This can be viewed as a generalization of the Pythagorean Theorem in 2D, and one may extend the result to 3D: the triangle formed by the first 2 dimensions covers the span of projection in their plane, and acts as the base of the next triangle, using that former triangle's hypoteneuse (squared), adding the next side length squared gets the square of that hypoteneuse. This relationship clearly visualizes how the sum terms squared continues to be equal to the square of the norm in higher dimensions (albeit not convenient to graph past 3 dimensions). This also lends exposition to why the Pythagoeran theorem holds in higher dimensions as a geometric process of going up a dimension and projecting back down again (like the derivative of an integral), as opposed to any intuitive expectations that 3D might need something cubic, and so on upwards. GA also has far more utility in the squared result than the magnitude (a computational win).

It also becomes useful to see the dot and wedge product as connective in terms of trigonometry, where the more two vectors commute into a shared subspace, the more they fall into the basis of aligning to their inner product, which may be simplified to a scalar multiple of the cosine of their terms, whereas the more the elements stand perpendicular to one another, the more that tends towards 0 and approaches the sine of the two as their identity. Therein, we get lamber's cosine law, that is: A ⋅ B = norm(A) \* norm(B) \* cos(𝝷), where 𝝷 is the angle between A and B. So much value coming to unit vectors, where if norm(A) = norm(B) = 1 their dot products are too, and A ⋅ B = cos(𝝷), simplifying a great deal of math in determining their relative quantities. That property of perpendicular and parallel elemental decomposition is integral to the geometric product's utility.

Anti-commutivity: Complex numbers suffered eons of rejection. From that taste to see imaginary components disappear, many insights such as the complex conjugate have been born. Despite not being restricted to complex numbers, we have a similar phenomena here with vector and scalar, shared quantities, so to find this result, it should follow that for the product of any given terms, if (from above, simplified), (A + B) \* (A + B) = (A⋅A + B⋅B) + A ∧ B + B ∧ A, then for this type of resolution to occur, it must hold: A ∧ B + B ∧ A = 0. It follows: A ∧ B = -B ∧ A.

This results in a net neutral state, where only the scalar identity remains, returning us to a space without those convolutions, giving us a stable, shared identity and null value term, where operations are closed onto the scalars. In so doing, we see permutations of exterior vector spaces evidence a violation of commutivity (A ∧ B ≠ B ∧ A), yet a symmetry does persist. For any inversion of vector elements the product is negated: so it's not unique to linear algebra or matrix math in 'anti-commutivity' holding that commuting two elements should invert the result. It's a much more fundamental property of vector spaces: the inversion of elements with a perpendicular component reject one another, inverting the whole. Like much of physics suggests (Newton's 3rd law among others): geometrically, the inverted vector wedge has equal magnitude, but opposite orientation.

The 3D wedge can be partly articulated with this as well: (ax, ay, az) ∧ (bx, by, bz), using only distributivity and anti-commutivity as ( (aybz - azby), (azbx - axbz), (axby - aybx) ), familiar from linear algebra as the cross product. It's only unique to 3D that one gets this result because it's only in 3D that 1 element is orthogonal to the other 2 and more needs to be defined to get to the product's proper derivation. GA uses the wedge product to define a more broadly useful set of terms, applicable in any dimensions, whereas Josiah Willard Gibbs was only interested in 3D due to his focus on thermodynamics, and sought to popularize the cross product to simplify the process.

Quotient: The inverse of a product, that is to say the result of division, equivalent to multiplication by a reciprocal, i.e. the multiplicative inverse defined by terms such that a \* a-1 = 1. Conventionally vector math leaves the idea of a quotient behind past the scalars, and it is very easy to look up results decreeing the idea of a vector quotient undefined as an undisputed fact. Despite that popular concept, this is one of the major things defined in geometric algebra. From the founding concept of a multiplicative inverse being defined by that which produces the identity when taken as a product with the original, we solve for this in vector math by finding the object which produces that result, i.e. a scalar value of 1 and null vector component coefficients, uniformly = 0.

However much sense it makes to give credence to the rhetorical posit commonly spouted by mathematics professor "what would that even mean: so we leave it as undefined", alongside concepts like point addition, the wrinkle which GA introduces which makes this noteworthily different is actually not in that definition, but in the anti-commutivity. That property means a few things: for one, A/B ≠B/A, but more importantly, there is ambiguity in the multiplicative inverse: does A/B = A\*B-1? does A/B = A-1\*B? if AB ≠BA these will not be equivalent statements.

Consequentially, if we are to define quotients under the terms of definable equivalence through a multiplicative inverse we have to introduce a distinction of operators between adding left division, where (\) ≔ A\B = A-1B, and (conventional) right division, where (/) ≔ A/B = AB-1. It may be difficult to get used to, but it is a reasonably necessary convolution to be introduced: it is generally possible to avoid and stick by one as a default to simplify and reduce room for error, and clearly right division is more conventional to fall back on, but there are times when left division is the more succinct alternative, so it is better to simply be prescient towards the possibility that this notation will be encountered somewhere, and (assuming that preference), find the use of \ as exceptional to be notably apart from the norm of being used to seeing /, keeping track of which is which that way.

e# basis elements: to define coefficients for variables in any given number of dimensions, a new convention is generally helpful to adopt. As with defining the cross product above, I needed to skip arithmetic and shorthand the mapping of one element onto the other in the resulting vector, elementwise, and how cryptic the algebraic processes may have been confusing, hard to follow or offputting to some. We need an algebraic shorthand to refer to these elements and their capacity for anti-commutative state relative to their counterpart, consistently and as apart from those coefficients. For these purposes, it is common to see { e1, e2, e3, ..., en } used in place of the more standard { x, y, z, w }, which extends more inconsistently in higher dimensions (even by the 4th dimension, really), and can more often conflict with the wants of many coefficients for many terms as well, besides. It also alows us to be ordering / chirality agnostic, so we don't expressly value left or right handed coordinates as more valid than the other: it is a matter of choice of notation. This allows referring to (e1 ∧ e2) = -(e2 ∧ e1) = -e2 ∧ e1 = e2 ∧ -e1 with less conflict or confusion.

Shorthand of the en ∧ em ∧ el element notation is also used: referring to (e1 ∧ e2 ∧ e3 ∧ e4), or more simply for ease of typing, even "e1^e2^e3^e4" can get longwinded. In general, we see this reduce to (e1 ∧ e2 ∧ e3 ∧ e4) = e1234, a pattern which works very well up to 9D (10 if allowing for e0 too, but that carries specific meanings in various conventions). This many suffices for most needs and should present no name overlaps, but if one wants 10 or more dimensions, borrowing from CS conventions for hexidecimal can stave off those concerns (that is using { e1, e2, ..., e9, eA, eB, eC, ...}). If keeping to capital letters (even without subscripting, equivalently as { e1, e2, ... e9, eA, eB, eC }), this pattern poses no particular issues. We can now reiterate the former more articulately:  
a ∧ b = (axe1 + aye2 + aze3) ∧ (bxe1 + bye2 + bze3)  
first we apply distributivity: =  
(axe1 ∧ bxe1 + axe1 ∧ bye2 + axe1 ∧ bze3) +(aye2 ∧ bxe1 + aye2 ∧ bye2 + aye2 ∧ bze3) +(aze3 ∧ bxe1 + aze3 ∧ bye2 + aze3 ∧ bze3)  
anti-commute vectors to matching orders & commute scalar quantities to front: =  
(axbxe1 ∧ e1 + axbye1 ∧ e2 **- axbze3 ∧ e1**) +(**-aybxe1 ∧ e2** + aybye2 ∧ e2 + aybze2 ∧ e3) +(azbxe3 ∧ e1 **- azbye2 ∧ e3** + azbze3 ∧ e3)  
re-group like terms & contract: =  
(axbxe11 + aybye22+ azbze33) + (aybze23 - azbye23 + azbxe31 - axbze31 + axbye12 - aybxe12)  
factor vectors & simplify squares: =  
(axbx \* 0+ ayby \* 0+ azbz \* 0) + ((aybz - azby)(e23) + (azbx - axbz)(e31) + (axby- aybx)(e12))  
re-group: =  
0 + ((aybz - azby)(e23), (azbx - axbz)(e31), (axby- aybx)(e12))  
factor vector terms; simplify: =  
((aybz - azby), (azbx - axbz), (axby- aybx))T \* (e23, e31, e12)

Almost the same as the cross product, except for the different elements which are all the exclusions of the other basis element in the original (cross products simplify this to skirt introducing wedge products: a simpler incantation for the result, but with issues conflating 'normal' vectors and the inputs as the same as opposed to 'duals'). Other convention ordering reasons may be less obvious.

As noted, elements don't begin with e1 ∧e2. Here that was just to substitute in the order to make it match elementwise missing 3D element to match a cross product. The other key change is using e31 as opposed to the sequential alternative e13. When a basis forms a cyclic group as (e23, e31,e12) the convention holds the elements as connected under a (right-handed), chirality (for ascending ordered terms, as opposed to left-handed / clockwise with descending ones). When possible to form such a hamiltonian circuit, this can be done by ensuring each basis element matches the neighbor to form a loop (by 3 in 31 after the trailing 3 in 23, 2 in 12 preceding the loop back to 2 in 23, etc). This is possible in an odd number of dimensions, while even ones by definition have an odd number of basis elements to pair with by exclusion, as well as d+1 binomial coefficients ot choose amongst yielding an odd number of 'graded' groups (i.e. equal numbered basis wedges). That medial combinatoric group out will be unable to reconcile the odd partner: it won't have a free basis element pair to be placed adjacent to, and so must have d-1 discontinuities. Elsewise, this keeps the products of any two in a reliable, consistent relationship to the others.

Also noteworthy, in conforming to an anti-commutative property and adhering to operations which sum to 0 (no remainder factors), the geometric objects which result are coordinate-free. Unlike {x, y, z, ... } labels denoting cartesian axes dividing space, the en labels are across not all of space, but only tying all objects compared under each direction as parallel & orthogonal to the other basis elements, per object. This fact enables a great deal of reliability in conformant operation results.

Grade: An operation, but not generally denoted with a given operator (save perhaps for the 'multivector' part extractor). Where vectors or matrices have a given 'rank' denoting how many elements exist in that dimension it spans (rows or columns: irrelevant here), the 'grade' of an element refers to how many orthogonal elements are wedged together to create that product quantity, and while a given vector may contain many elements of a given grade, it is common to divide them into subvectors, ordered by their common grade if multiple grades exist. So for the case of e1 ∧ e2, grade(e1 ∧ e2) = 2, but we can take the wedge products of many vectors as well, I.e. grade(e123) = 3, grade(e1457) = 4, etc. It should be noted that the grade of an object may not simply be the sum of all elements included where redundancies occur, like grade(e1214) ≠ 4. e1 ∧ e2 ∧ e1 ∧ e4 = -(e1∧e1∧e2∧e4) = (e1∧e1)∧ -(e2∧e4) = -(e2 ∧ e4)= e4 ∧ e2 = e42, and grade(e42) = 2. Here we can highlight that while e1 ∧ e1 = 0, that doesn't imply the sum value of wedge products necessarily will.

An object grade never exceeds the dimensions of the space it spans, and the number of elements a dimension can have for a given grade will be a factor of the binomial coefficients for that dimension and grade. In 4D for example, there are 1 ways to choose 0 elements, 4 ways to choose 1, 6 ways to choose 2, 4 ways to choose 3, and 1 way to choose all 4. This corresponds to a row of Pascal's triangle for any dimension equal to the nth row, which will always have 2n basis elements in total for any dimension, and may make sense combinatorically since we are reducing permutations via anti-commutivity, thus nCr(d,k) elements exist for a given dimension d & grade k. Interestingly, this reduces all the way to the base case of 0, where we have 0 dimensions, i.e. the scalar, i.e. the real numbers, which may seem odd since it implies 1D numbers have 1 scalar grade-0 element and 1 grade-1 element. If one has been considering these vector terms as decidedly unique or apart from complex numbers this would be stranger, but in fact the connection between the two is stronger than it may at first appear.

Reverse: Most often given by the dagger symbol (†), occasionally in some places, tildes (~), either prefized postfixed, or over the object's symbol name(s), some also use asterisk, but of course there are complaints about that with regards to (\*) meaning the general multiplication of 2 objects, let alone all the extra meanings a unary variant can take on in code, in terms of dereferencing, and several other, more GA exclusive operations like the dual (later still), where unary (\*) is still the preferred representation besides: dagger looks comparable and distinct enough compared to asterisk, so it is the most common and least confusing, and therefore generally most accepted. Geometrically, this typically represents the opposite orientation of the object, but it may look identical to the original depending on the object's grade. The reverse of a given object / multivector is to take the multivector entailed, and per object / globally (however easier to conceptualize), take those wedge products in reverse ordering. Depending on the grade of the object and the dimension of the sapce, this may result in the negation, or the original element. For examples, with 3D mutivector M (presumably with uniform coefficients of 1, omitted for brevity), vs 4D N:  
M ≔ { 1 + e1 + e2 + e3 + e23 + e31 + e12 + e123 },  
reverse(M) ≔ { e321 + e21 + e13 + e32 + e3 + e2 + e1 + 1 }  
or equivalently: { 1 + e1 + e2 + e3 - e23 - e31 - e12 - e123 }  
N ≔ { 1 + e1 + e2 + e3 + e4 + e43 + e31 + e32 + e24 + e41 + e12 + e243 + e314 + e421 + e132 + e1234 },  
reverse(N) ≔ {e4321 +e231 +e124 +e413 +e342 +e21 +e14 +e42 +e23 +e13 +e34 +e4 +e3 +e2 +e1 +1}  
= { 1 + e1 + e2 + e3 + e4 - e43 - e31 - e32 - e24 - e41 - e12 - e243 - e314 - e421 - e132 + e1234 }

Per grade, this develops an ongoing pattern of (+,+,-,-,+,+,-,-,...). Generally k-grade parts transform consistently. The reverse is known for each, up to a sign, which is a factor of (-1){k(k-1)/2}. To better intuit this, it helps to think in terms of recurrence relations and subsets within a total number of swaps to incur inversions for a given grade. what matters is if the swaps required to invert a value is odd or even, and the k-1 grade object will have taken k fewer than the k-grade element will to re-order properly, as a subset, and the k grade must then move that extra kth digit through (k-1), sorted elements. In the base case, a 0 or 1 grade object takes no inversions, grade 2 takes 1, grade 3 then taking another 2 to move the third element back across the other 2, which will do nothing as even numbers of flips have a net neutral effect, grade 4 will similarly take 4-1 or 3 additional flips which will flip things having an odd number of transpositions, and they will alternate so on as numbers rise from these base cases. All that is to say it will take the kth triangular element of Pascal's triangle to flip, and an even count of total flips will revert the sign, while an odd total will invert it, so since (-1) rasied to powers of odds retain while evens revert, dividing by 2 gets the domain to match consistently to only flip signs after each other grade uptick. This observation further reduces programmatically to bitwise operation (k&2): the alternate odds produce exactly the right inversion, so !(k&2) or (2 \* !(k&2) - 1) if we want a -/+1 result for scalar multiplication. A bitwise xor used inplace with that and the coefficient c and knowing the appropriate width w to shift by to move a value to the MSB can also work: (((k&2)<<w)^c), if a 1 in the MSB also negates the value for your platform (as generally true with IEEE floats which do not use 2's complement). If following implementation details are outside of your concern or understanding, don't be concerned for your ability to continue following: it's a tangent for some, but helpful for others.

Roots of Unity: A root of unity can be taken quite literally in that it must be a number for which some multiple of itself exponentiates to the identity. The principle of the identity element is in its multiplicative idempotence: it can multiply anything, including self-exponentiation to no continued change regardless of iterations, which is an important element to be able to reach as it stands to be a benchmark in normalizing any group of operations. For this reason we care greatly about roots of unity, and those are conventionally namely numbers which square to 1 and -1. Any number and it's inverse should get us to 1, while -1 would be reached by taking both the negation and reciprocal of a scalar value. In this context, it makes more sense why we might value the given square root of -1 conventionally known as the imaginary unit i (or perhaps j for electrical engineers). It is however less common that this is seen as equally valuable to hold for positive numbers as well. Roots of unity are introduced for complex numbers, and it is one thing to have an imaginary unit, given rise from x2 + 1 = 0. After all no real number can square to -1, so there must be some other quantity being lost in translation, but what about +1? Is it reasonable to assert that half of this equation carried a factor which wasn't a real number, and the other half that restored us to a null state wasn't? Was +1 a scalar, was it another squared value: how could you tell one way or the other?

Perhaps despite +1 appearing to have a real root one can define, it's not so unreasonable after all to think that maybe both +1 and -1 had some unit to them which was non-real, and measured their exponentiation in some way: evaluating their dimensional span and whether that squared back into the rest to cancel out. It is at least helpful to be able to measure both parts, regardless of whether it is accepted as reasonable, and at least see what benefits apply.

For any basis element discussed insofar, we are looking at such roots of unity which we call geometric numbers (though not to say quite all the geometric numbers). It can seem strange for some that such numbers which collapse down in the exterior product due to self multiplication are defined as 0, yet don't collapse the entire value where others persist. This is because the geometric numbers collapse to span 0 dimensions, that is to say they reduce to scalars, but are taken in product with the other basis vectors. If no other basis vectors remain, then this does reduce to a scalar, but if other elements remain, a vector span does too. The scalar it reduces to however need not be 1, and it is not unconventional to see other roots of unity used: this is the entire premise of complex numbers, and those have been shown to have a great deal of use in modeling physical phenomena. So too do these numbers which square to 1 (albeit modeling different ones).

Clifford numbers: Clifford numbers take on the notational form Cl(p,q) where the p and q are subscripts which refer to the number of basis elements for dimensions which square to +1 (p), and ones which square to -1 (q). Using this notation, we gain a set of Algebra which unify many dimensional spans into a more universal set to work with. Applications of exterior algebra was popularized in the 19th century by many: Hamilton used them to derive quaternions, Gibbs popularized the cross product, William Kingdon Clifford isn't known nearly as well, in large part due to his relatively early demise at only 34 years old. His work was in this algebraic group notation, and brought together Grassman's exterior numbers and the Complex or Quaternions to work together in one treatment, equivalent to the the en notation used above, by using e- and e+, and while it can be somewhat cumbersome to refer to as more dimensions subscripted by signs and numbers, if (+) is assumed unless (-) is stated, the same e-n / en names can continue the same use.

Sadly, Clifford's early demise came prior to unifying these groups with the euclidean case to provide an algebraic superset, and his work was largely forgotten for some time without his ability to stand behind it, work it through that idiosyncracy and evangalize it himself, but he laid significant groudwork all the same, and under the current e-n / en framing, it seems almost obvious what might have been the missing dimension to link them now, in hindsight.

Geometric Mirrors: The full set of geometric mirrors include not only the roots of unity, but also the indeterminate, and are represented most commonly with an extension of clifford notation, that is G(p,q,r), with p real basis numbers, q imaginary ones, and r dual ones. Called mirrors because these geometric numbers multiply with their inverses to model reflections, as one may be loosely familiar with if having studied quaternions and having used the sandwich product.

Accepting the utility of x2 + 1 = 0 being able to cancel out more sensibly in considering measuring both numbers needing to be squared values to be more safely considered null, it may also make more sense that we are comparing apples to apples in all terms, that is to suggest: x2 + 12 = 02: a squared null result as well. Both -1 and +1 are produced by valid roots of unity (e- and e+ respectively), and can reasonably square to cancel one another, but that also suggests another non-real root may be in the result: a number which squares to 0, but perhaps has a square root which is more just the scalar value 0, like with 1. A given dimensions G(p,q,r) are referred to as an algebra.

Dual numbers: This additional value squaring to 0 alongside the scalars is what we call the dual numbers. Although William Clifford introduced them (and so honorifically, you will see the notation Cl(p,q,r), or 𝓒𝓵(p,q,r) used equivalently to G(p,q,r) in some places at times), and Eduard Study later used the concepts as a measure of the relative position of skew lines and dual angles, giving them their common usage under the symbol (ε) akin to the imaginary unit (i), it was not until much later that abstract algebra started to see these numbers in parallel with the roots of unity as the root of the indeterminate it is now.

The newest addition to any Clifford algebra group, some sources will cite the inclusion of the dual numbers as a misnomer in their not being a root of unity, and many fields having been developed without, it is therefore mostly in that reason that it is an addendum, and not a medial subscript to take (+1, 0, -1) basis element values sequentially, perhaps more sensibly.

It is rare that an algebra has need of more than one dual dimension element, as they serve to translate other basis elements into a reference frame. The promotion of one dual dimension is ultimately what homogeneous coordinates and promotion are about doing, and the abstraction from dual numbers being given explicitly is why clip coordinates require normalization, or a homogenously promoted matrix from linear algebra may seem so fragile and finicky to get desired results.

1D exponentiation: Multipication in GA models transformation via series of reflections. Geometric mirrors' exponentiation is highly useful to showcasing how each models the transform to which it correleates. It is most simple to graph in 2D with the 1D variants: the complex, the duals and the hyperreals. It can be demonstrated well with the simple orthogonal cases with 1 basis element and 0 scalar, and then with the (√2/2) variants with equal member measure, each taken to incrementally higher powers: ((√2/2) + (√2/2)R)n, ((√2/2) + (√2/2)i)n and ((√2/2) + (√2/2)ε)n each having a unit length, they will exponentiate to the identity, and each power of n progressively moves the resulting point graphed in their planes with different patterns / rates.

Complex numbers trace rotations, hyperreals trace hyperbola and duals trace translations. This may make some sense: imaginary numbers keep flipping back and forth, circling a scalar number, hyperreals keep adding to it, and dual numbers keep cancelling out without effecting that scalar's growth, but multiplying their own magnitude all the same (and proportionately / inversely at that).

I like to link the 3 by thinking of them not just around how they relate to the unit circle as they move via graudal exponentiation, where the tendency forms for a baseline for the complex to pull back onto itself, the hyperreals shooring off the other way, and the duals extending to form a straight vertical line between them, but also how that tendency extends along a temporal axis vs the real value. The value {1, 0} for any among them stay neutral, but the more of a non-real component, the more the tendencies of these transformations slowly start to form.

Thinking of how the non-real part effects the sum value over time and regular exponentiation, we see the complex pull back from shooting too far away to circle it and form a sin wave, hyperreals curving off to map spatial distortion with time, and duals just continuing forward. For these reasons the complex, hyperreal and dual numbers are generalized in a spacetime algebra under another set of aliases where basis vectors v are (v² > 0): timelike, (v² = 0): lightlike, (v² < 0): spacelike ones. I'm not going to get into the menutia for spacetime algebra, but these concepts can still be helpful ways to frame the geometric mirrors to group them and generalize their tendencies.

Euler form: In the study of complex number theory, one may learn the representation R\*ei𝝷, where the notation expands to a complex number's complex coordinates R(cos(𝝷) +i\*sin(𝝷)), and corresponds with the rotation over exponentiation, and by considering the sums of geometric interpretations of Taylor series with the imaginary unit in the numerator, getting exponentiated, we see the sum geometric series (equal to ex), to be decomposable into the sum of odd powers (equal to sin(x)) and even powers (equal to cos(x)), the notation makes sense given their sum being equal to ex. Then, over a temporal axis this produces a sine wave with the sin value in the imaginary axis thusly modeling the transformation of the rotation performed to model periodic physical phenomena in waveforms like light and sound. Also since i squares to -1, this gets substituted into x^2 + 1 = 0 to famously produce ei𝝿 + 1 = 0 (which I find actually shortcuts a lot of the beauty).

This parallel goes beyond complex numbers to cover all geometric numbers equally well under similar measure though. Where ei𝝷 makes sense in being a complex composition, comparable eR𝝷 for the hyperreals and eε𝝷 for the duals highlight the same effects relative to their transformations. As with complex numbers where r1ei𝝷 r2ei𝞅 = r1r2ei(𝝷+𝞅), so too do the other geometric mirrors have their transformations modeled comparably under those terms, being so much of why the geometric product is so useful, in producing the scalar product of the two object's magnitude, as well as the sum of their operational change relative to the shared baseline that is their scalar quantity. This also gives light to understanding how the total quantity of a larger dimensional object like a quaternion (modelable as G(0,3,0) among others), could have total magnitude and cosine quantities inform the axis being rotated about more concretely, with geometric understanding.

Notationally, it is likely that Euler form has a relation ship to why the basis vectors are given by starting with the letter e, and the overlap is a convenient one, but William Clifford having died young and being unavailable to ask if these were his motivations for the notation, this is purely speculative. Some consider if it would be better to have multiple letter prefixes or the like to denote which geometric mirror a basis element corresponds to instead of uniformly using e. However, this reference to the Euler constant exponential form, the fact that the basis elements can be abbreviated to subscripts of {-q, ..., -2, -1, 0, 1, 2, ..., p} are some of the stronger arguments against the change (apart from such attempts further fragmenting standards being something to minimize).

Exp: Using Euler form to represent exponentiation of geometric objects. For the light it sheds on transformations, exponentiatial form is arguably the preferred read of them, so as the polar form has some bearing on understanding the transformation and the relationship between two objects, and it can be surprisingly easy to understand conversions succinctly, it is used for conversion often.

K-Vector: The sum / list of non-zero coefficients matching a given grade-k are a k-vector. It's also uncommon to use for 1-vectors or 0-vectors (as opposed to vector or scalar respectively), but k-vector labels allow for a more uniform framing of terms and may be understood regardless.

Blade: Geometrically, a multivector which corresponds to a particular geometric object for a given dimension is a blade. This is to say they represent the outer product of vectors, and the resultant multivector will therefore carry some geometric representation (not to necessarily say capable of decomposition: the product is ambiguous / lossy). This stands as somewhat of a counterpart in subtext to a k-vector, where you may refer to a k-blade as a more geometric body of terms, so 2-blade would be a more dimension agnostic way of referring to a full set of bivectors.

On some personal notes, I found many sources which (early in my studies when still formalizing nomenclature for myself), led me to believe the relationship between blades & k-vectors was in reference to any given coefficient for a wedged term within the k-vector, or perhaps the full group of all coefficients for a given grade, and equivalent to 〈M〉k operator notation for grade extraction, but these misconceptions slightly miss the point: some k-vectors are unable to be written as the outer product of 2 vectors, like e12 + e34. So k-blades are a subset of k-vectors, and a k-blade is the outer product of k-vectors, which does more necessarily carry geometric meaning than a k-vector.

I also am sad to see blades exist with the convention for extracting a given grade k-vector from a multivector M used being the operator 〈M〉k to get the k-grade part. I take no personal issue with that notation specifically, except that the operator (†}, spoken "dagger", exists for this opportunity, and is used as a 'reverse' operator for such trivial reasons as looking alike enough to the already overloaded (\*) to suffice in that role. My only real issue with 〈M〉k usage is in the pattern that people use 〈M〉to anonymously refer to extracting 〈M〉0. That is to complain that 〈M〉should be obvious to anyone as meaning not the entire multivector, but only the scalar / 0-blade. It's technically fine to define this way, but it is not intuitive to beginners that this notational omission should mean that. I suppose it makes it more elegant to write the most formal definition of magnitude = 〈A†A〉(meaning "the scalar part of A reverse times A"), but that hardly seems to adequately justify comparatively, and adding a 0 there isn't much. Regardless, I say that less to suggest my own standards on top of so many others, and more as a caution for your own further study, should you research more beyond this document.

Bivector (et al): A label for the collection of wedging of a number of vector objects together to produce a new vector object. So a vector wedged with another vector produces a new type of geometric object called a bivector. Generally we use latin prefixes to refer to the grade of the object being described, so in the case of the cross product (before taking a complement), we have bivectors since there are 2 terms: this would extend to define trivectors for grade 3 objects, quadrivectors for grade 4, etc. This means grammaritcally that a bivector can be synonymous with a 2-vector, a 3-vector is a trivector, etc. It can also refer to a single coefficient within a k-vector however, and as opposed to those more algebraic labels, can imply gemetric subtext of intended interpretation. Which you use can carry that implication, as to whether your focus is on the geometric object being represented, versus the collection of like-graded variable+coefficient pairs.

Where a vector has direction and magnitude, a bivector has orientation and magnitude, and is an object which circumscribes the plane of the vectors that went into it. The norm of the bivector still represents magnitude, but as area of the plane between the two vectors that created it, but notably also has no shape. As many may have learned from the cross product, the magnitude of is equivalent to A ∧ B = norm(A) \* norm(B) \* sin(𝝷), where 𝝷 is the angle between A and B. You may also have learned that is equivalent to the area of a parallelogram area where norm(A) \* norm(B) is proportionate to that scale, but this does not imply the parallelogram area is the shape of the bivector either: 12A ∧ ½B = ½A ∧ 12B = 2A ∧ 3B = 3A ∧ 2B = 6(A ∧ B), it really doesn't enter into any k-vector how much of the product came from which source as much as 30 isn't any more 5\*6 than 3\*10, or any other non-integer factorization pair, or set larger than a pair at that.

Taken in conjunction with Lambert's cosine law though, the product of 2 component magnitudes (or a desired result magnitude), can be used in conjunction with the sine & cosine of a given 𝝷 to create the geometric representation of arc displacement in a plane, where r1 (≔ the norm of A) \* r2 (≔ the norm of B) = R (≔ the combined / desired magnitude), and <R\*cos(𝝷), R\*〈AB〉(≔ the unit plane around which A^B run parallel)\*sin(𝝷)> can continue to work for the bivectors of higher dimensions, not merely polar coordinates of the 2D plane or complex numbers. We can even give this as the definition for a quaternion or the GA equivalent of a rotor, which can articulate this displacement from A to B by the given 𝝷 and not simply as the abstracted transformation, which even many who are literate in quaternion math continue to feel it necessary to treat as a black box, in terms of its geometric implications.

Similarly as more successive wedge products are taken, the properties remain in having an orientation and magnitude, and geometric representation becomes that of an oriented volume with magnitude of the parallelipiped of the component vectors sweep (the complement in 3D being equivalent to the scalar triple product), and so on upwards for hyper-volumes past that, likewise.

Pseudoscalar: aka basis blade, capital I or Ed, etc. Within a given dimension, there is always 1 way to choose 0 basis elements (the scalar, 0-vector part), and reciprocally, 1 way to choose all d elements, but that name will vary by dimension, so often it's simpler to refer to it's mirrored element nature by calling it a pseudoscalar. Another way of framing these terms besides k-vectors and blades or latin prefixes is also to consider the mirrored nature of a set of basis elements, the pseudoscalar is the basis element which represents something geoemtrically distinct from the scalar (that measuring the shared cosine of angles moved around the local axis the sine and cosine are perpendicular to), as opposed to the determinant of the transformation. In a matrix this relates to the scale of the transformation, here it offers comparable insights but how one refers to the scale of a geometric transformation in a tensor is different, and relates more to the transformation of the whole vector space under a given orientation. So where linear algebra would lead one to think in terms of scale, geometric algebra would be more apt to consider the pseudoscalar as a change of reference frame for the space (or hyper-space in higher dimensions). It stll relates to the scalar like other k-vectors and (depending on the dimension), may not produce the identity after squaring.

A given grade will have an equal number of elements as one continues inward along basis elements too, and so too can this mirroring continue to aptly refer to those (d-k)-grade objects: i.e. pseudo-vectors, pseudo-bivectors, etc. There are some advantages in certain flavors of GA in preferring these terms, mostly in terms of the way they respond to dimensional expansion or contraction vary inversely in comparison to the originals (more details on that in duals however).

The use of the 'psuedo' prefix has been criticized as connoting 'lesser' status unduely onto these objects, but the alternative suggestion of so called 'anti'-scalars, 'anti'-vectors and so on, promote conflation with the later introduction of concepts like a 'vector inverse', which is certainly worse than mere unfortunate subtext. The other alternative seen is to discuss these objects as 'dual' to their counterparts, but dual is a highly over-loaded term in GA, and as related as most usages may be, pseudo-k-vectors or (d-k)-vectors are both terms which have far fewer areas for conflation.

Multivector: the sum of all objects resulting from a geometric product, which (including wedge products of elements), are apt to not be restricted to objects with grades equal to the input grades. The cross product needing to take a complement is an excellent example of this, but one could take a k-vector and product of a prior wedge product and get another grade object. In general, a multivector is the object which allows for any combination of any / all grades of objects. From a programming perspective, this may need to be a different object per dimension if one wishes to keep to certain organizational pattern, but more remains to be said before settling on conventions there. It is entirely admissable for any number of elements to have 0 coefficients, and if only non-zero coefficients exist for a certain grade, that may allow the multivector to simplify to another geometric object (or comparably promote a given bivector or the like to a multivector with many 0 coefficients in the other grades). The given d dimension of multivector will have 2d elements.

Complement: Typically noted by a macron (I.e. comp(A) = Ā): the operation gets the excluded, orthogonal basis elements for an object within the given dimension. For 3D, the complement to each of the elements above would result in the cross product, so we see that the actual definition of the cross product of cross(A, B) isn't simply (A ∧ B), but rather comp(A ∧ B) (which could also be noted in more proper LaTeX or the like by an overbar / macron on the group). An interesting property is that A ∧ Ā will by definition assure that each element will have the full basis element set, and therefore can be factored out per element, and therefore the complement of that is a pure scalar: each element will have been a sum of memberwise multiplications of A with itself, i.e |A|2, or a manner of proof for the dot product (however notably, this is more difficult to parse in other flavors of GA than VGA, and the comparable definition given later using the reverse is prefered).

One issue with this definition is in the ambiguity of orientation for the omitted elements. Where taking the complement of one grade object and another similar one wedged together, there can be a significant difference between the result of Ā vs A (right vs left complements), when the span of dimensions is even, and the sign flips can be difficult to keep straight (when dimension d is even and grade k is odd, they invert, that is: A = (-1)k(n-k)Ā). Otherwise they are equivalent and actually can be commutative, but in this case we need to be careful to take right complements on the right of a wedge, and likewise for left ones. This isn't to say signs don't flip then either, but that the right complement flips signs such that A ∧ Ā = each sequential basis element (i.e. e12345 in 5D). This means that as grades increase, each other odd grade inverts (which is notably also the same as the reverse). Taking either complement twice also producing a different result than one may expect (an involute not the original, where each odd grade is negated: again, for even dimensions not odd), but taking each complement once does undo the effect of the other to restore the original.

Norm: Magnitude becomes a peculiar issue in geometric algebra, even before introducing the geometric mirrors / numbers when it gets more complex. It seems like I've discussed it enough with the dot product, but in the face of multivectors and k-vectors, there are some additional factors to consider. With scalars and vectors, it is conventional to rely on the dot product's square root, but when it comes to bivectors, there is a new issue with these basis element factors. With scalars, there is no basis element, and with vectors, no swapping takes place as basis elements are already adjacent, but with bivectors, there begins to be an issue: (e1 ∧ e2 ∧ e1 ∧ e2) = -(e1 ∧ e1 ∧ e2 ∧ e2). In order to get alike elements to become adjacent, an index swap must take place, which is a fascinating turn of events because this means a bivector acts like an imaginary number, as will a trivector present this issue, but not a quadrivector, just like with the reverse.

The reverse holds compelling products taken with the original, i.e. A†A, as well however, and can serve to correct for this. In taking the product of the reverse and the original, much like with the complement, the result is a sum of square components placed into the scalar, so by extracting the grade 0 element, 〈A†A〉0 gets us the dot product, i.e. the squared norm on a more general basis. Therefore, √( 〈A†A〉0) is the most general way we can calculate the euclidean norm of an object.

Also interestingly I read somewhere (I want to say Lengyel's book, maybe Macdonald's in vol. 1, but haven't found it again since), that the ever increasing k-vectors prove to reduce to the 0-vector identically as getting the vector norm, or for higher k-vectors, all equivalently to an absolute value, suggesting there is no notational necessity to use double bars as distinct from single bars when taking the absolute value versus the norm as they are equivalent (except perhaps with matrices an their bars signifying the taking of a determinant). I was convinced to adopt that notation change myself more broadly, but I hope if it upsets anyone, this aside might at least help to assuage concern in that it isn't done absent mindedly or flippantly: I'm sorry to not cite the specific reference though.

Ideal Norm: The ideal norm differs from the (Euclidean) norm in that it represents the magnitude taken "at infinity". When dealing with dual numbers, it is possible that some basis elements are included in the lengthwhich might not be appropriate, and the above norm takes care of this to only give us the euclidean length with the numbers of other dual basis vectors or complex ones made to square and tend towards the appropriate scalar values, but while typically the length we want, that is not always the desired property of the norm either. Sometimes the Euclidean magnitude is 0, and it helps to know if a non-zero vector is at play, or if you take the product of two elements which were parallel, the result will be dual, and in this case we might want to ideal norm in order to get the distance between those objects. The only difference in an ideal norm is what that sounds like: we take the dual before taking the norm, that is √( 〈A\*†A\*〉0), or just ( 〈A\*†A\*〉0) if the squared magnitude might still suffice in those cases, as conventional in CS when comparing square lengths.

Involute / Grade Involution: like the reverse, but instead of swapping signs every 2 grades, it swaps each grade (so just (+, -, +, -, +, -), resulting in bitwise !(k&1) equivalence. No great uniform symbol defines this operation, varying in what signifies it: some use carets (so Â, but of course that conflicts with unit vectors: it's related in terms of a sandwich product being simpler with unit vectors, and case sensetivity can keep things less confusing, but problematically ambiguous regardless), some use postfix \* (so A\*, but as discussed in reverses, \* already means plenty of important things). Some have taken to using a postfix caret for it (so A^), which seems the least ambiguous so far, at least. Like the reverse, it has some similar properties related to inversion, but geometrically can't be taken in isolation to be equivalent to that either.

Inverse: Geometrically, a given object's inverse is that mutiplicative reciprocal which restores to the identity when taken as a product with the original value. Though linear algebra conventionally holds this to be a meaningless non-sequitur of a term in relation to vectors, in geometric algebra we have a great use for vector inverses, as they provide the basis for normalizing transformations. The identity in GA continues to be what it is in the scalars: a scalar value of 1. For any number of dimensions, it is possible to multiply a scalar 1 plus any number of k-vector zeroes to result in the same output (going back to the extended scalars remaining closed under scalars so long as the vector parts are null. By creating a vector inverse, we can create a restorative force which may cancel out the distortative effects of multiplying geometric objects together into different objects to produce a net effect of only a transformation about that object using a sandwich product.

So the inverse is important to transforming things, but it must be reasonably involved to compute, surely. As luck would have it, not at all: similarly to how unit vectors are reliable because they are on the identity and their own inverse, stable, etc, the inverse is generally merely a sign change on getting to that point. Where a unit vector is calculated by dividing by length (n1 \* |n|-1 = |n|0 = 1), here we hope some version of n will multiply to that, so if n1 \* |n|2 = |n|-1 and n1 \* |n|-1 = 1 as earlier. The only difference is in negating the vector elements such that the grades complement and cancel out in the second half, which is exactly what the reverse taken with itself does to achieve the dot product. It therefore makes sense that the vector inverse for a given k-vector A is A† / A ⋅ A ("A reverse divided by A dot A"), as A ⋅ A = |A|2, and A†Agives the norm in its scalar part, so it follows that in sum, the operations should give the reciprocal magnitude object which yields a scalar result.

Since we have a definition for a vector inverse in GA, this opens a number of potentials as it stands that multiplying by the inverse not only acts as the reciprocal when taken with the original, but even generally, this means the A \* B-1 = A \* (1/〈B†B〉0) \* B† = A / B. Vector division opens a number of operational possibliities.

Hodge Dual: An operation resulting in an object with the same magnitude and orientation, but as an object with all the excluded basis elements included and the included ones excluded. Also a highly overloaded term with many similar meanings. Much simpler take on the complement to get the orthogonal basis elements compared to the original mapping: for multivector A, dual(A) (most commonly A\*) = A/I (that is, the pseudoscalar).

For a basis element with magnitude 1 (by its definition), dividing by length squared to get the value of I-1 is just I† (which may be I or -I depending on the dimension, as well as the basis handedness). Just like the complement, this gets the elements which are orthogonal to the original object, but because the complement has the same issues as the reverse, and we are multiplying by a reverse for the given dimension, that means the dual cancels out those issues to give a more consistent result.

It is notably very easy to conflate a number of "duals" in GA, but the Hodge dual is the most pivotal, and most likely to be meant by 'dual' or 'dualized'. There is also topological Poincaré and Lefschetz duality ( but usually mentioned in GA by contrast to the Hodge dual), complements (over-simplified terms), inverses (also a bit of a misnomer), dual numbers (an unrelated matter entirely), especially as "the duals", dual numbers may be easy to misinterpret where "a dual" is called for instead. It is also possible however that the hodge dual could mean another closely related subject.

In some GA flavors vector interpretation (and indeed all k-vectors) are subject to change. Better detailed in it's own terming, PGA for example defines vectors geometrically as the first dividing space subset (that is a 2D line of the plane, a 3D plane of the volume, a 4D volume of hyperspace, etc), the bivectors then 1D intersections winnowed objects from there (2D points, 3D lines, 4D planes, etc), etc down the line. There, that means the geometric representation of an object changes with each dimensional promotion: the duals are the ones which remain consistent as the (d-1) grade, (d-2) grade, etc. Obviously not always desirable, but the behavior highlights that this was reciprocally true in VGA for pseudo-scalars, pseudo-vectors etc. The geometric PGA result is that as one adds dimensions, points become lines, lines become planes, and planes become volumes, while the psuedo-objects remain d-k grade. While some objects may be better expressed as ones which promote with dimensions (like a blueprint), others would evidence these changes as bizarre (more like a schematic view of an object representation), and we may very well want both to occur.

In order for this to work, the dualized object must be written explicitly as a d-k vector, and in that distinction we really are splitting hairs on what kind of dual one means by the term 'dual' when said, but it is a relatively novel concept and doesn't yet have a distinct term from the hodge dual as to when one means to take the dual (get a k-vector that is d-k grade equivalent to the input grade k), or to convert to the dual (get a d-k grade object equivalent to the k-vector input), let alone which of those operations being "dualized" might refer to.

Antiwedge / Vee: The operation denoted by an inverted wedge (∨). Closely related to the complement in conjunction with the wedge product (these are the jokes, folks), the antiwedge product (a bit of a mouthful, so called a vee product too, read "A vee B" for the visual proxy: sadly (v) isn't a definable operator in most languages however), represents the exterior combination of elements absent in the operands, i.e. comp(A ∨ B) = Ā ∧ B̄, or equivalently, comp(A ∧ B) = Ā ∨ B̄, and also equivalently dual(A ∨ B) = (A ∨ B)\* = A\* ∧ B\* (the dual and complement closely related, especially when taken symmetrically).

This correlates perfectly with logical disjunction and DeMorgan's law, and begins to demonstrate compelling parallels with set unions and intersections. Geometrically, the 'regressive' product has the opposite effect of the wedge product, getting the lower grade object from between the two, that is .

Join &| Meet: Usually given by (&) and (|) respectively: operations very much akin to boolean union (&&) intersection (||) (respectively), between two objects in regards to their spans when multiplied together, and conventionally, generalizations of the wedge and vee products as well.

Where like grade objects are taken together, the result is a multivector of even grade objects, but where GA starts to show its applications is in the products of more types of objects than that. The join is a generalization of the outer product A ∧ B = 〈AB〉gr(A)+gr(B), producing an object which has the sum of the grades of (uniquely orthogonal), input basis elements, 'joining' their spans. The meet is a generalization of the inner product, so (A ∨ B)\* or equivalently A\* ∧ B\*= 〈AB〉|(gr(A)-gr(B))|, which is to say the lower grade part of the result. For either (presuming mono-grade object inputs, or else, per grade in each input), one will always end up with these grade sum & differences for the output (plus a third grade potential in the commutator), and those resulting objects will be in one multivector, and so must be calculated together by taking their geometric product, but that can be split up into the component grade objects by the join (∧) and meet (∨), and perhaps even split into separate function calls the geometric product calls, as the grade objects resulting have no common factors and no wasted re-computation to be concerned with (conventionally: other, interpretative inputs could violate this).

Also notably, using bitwise operators (whether &, |, or even ^), are the areas which make for precedence woes. It is a valid point in mathematics that there is a distinction between the set intersection or union, and the meet or the join, but in terms of markup and translating to code, it is a matter of pedantics why the join and the meet are not held as fully "equivalent" to boolean set intersection, as there is no better operator (that's why they are using the bitwise ones), and there is no other operation which makes more sense in their places. In fact the meet and join are very geometrically apt to describe intersection tests more than anything, as it is what they do to the geometry, so having A && B of two planes give their line of intersection is quite wholly appropriate, and additional boolean methods to test from there can determine if the inputs were intersecting by the state quite easily, which makes much more sense than using low precedence operators requiring parentheses, and adding more confusion when one means & or ^ for the exterior or wedge product over a given set of inputs.

It makes some more sense in light of the mixed grade inner products relating to | though, as the need to take absolute value of grades isn't always preferable, so we introduce (⎦) and (⎣) operators to distinguish between taking the left and right contraction respectively to take the higher grade object when the contraction operator is facing it, the scalar resulting otherwise being 0. These more situational inner products are useful in taking less commutative inner products to have more consistent results in terms of using that to produce projections via consistent left ro right hand arguments (when mixed grade objects are used, such that if the lower grade object is not faced by the operator, the result is 0 (the higher grade object projects nothing onto the lower grade object), which can be preferable in some cases.

Subalgebra: Generally, meaning that we are working with a subset of a given algebra. GA working in n-dimensions, this could mean a few things: it may mean that the entirety of a set of numbers for a given dimension's numbers which is normally clsed unto itself and it's own internal operations is also comprised within a larger dimension's terms. We see this with 1D or 2D being contained in 3D, and it continues as dimensions rise: for example complex numbers are a subalgebra of quaternions, as quaternions contain all the 1D math a complex number does in it's 3D rotations, and this makes intuitive sense from the notation G(0,1,0) ⊆ G(0,3,0), given that 1 < 3 in the only non-zero basis.

The other place subalgebra is used that is not always as obvious is as with terms like "even subalgebra" or "odd subalgebra", and in these cases it is to highlight the grade of objects of a given algebra. So when you hear talks complain that dual or regular quaternions make less sense because they are only the even subalgebra of PGA, this is what it is in reference to. As with how bivectors square to -1 despite being real basis vectors that square to 1, the quaternions and dual quaternions map the i, j and k elements equivalently to the bivectors, ε to the quadrivector, ε(i, j, k) being the other 'dual' half of the bivectors for a dual quaternion, and the scalars to the dot products.

Where this is less than obvious is that conventionally, one thinks of say the complex i as being tied to the y axis of the complex plane, or aliased to the complex basis part of rotation around the x axis, j to y and k to z, and so on: as we know by now a rotation happens in a plane, and 'around' a normal axis it holds neutral and doesn't rotate the elements of (the eigenvector of the plane), but this correlates to bivectors, not their dualized vector equivalent. This implies i is not actually associated to the x axis, but the plane it rotates around (the yzplane: that is e23), which means if we want struct layout to match existing (dual) quaternion conventions, we have some messier looking layout decisions to map duals onto the other duals memberwise. This ceases to be a convention to match or an issue beyond 4D, but produces some odd idiosyncracies in the lower dimensions.

The reason we value talking about the even subalgebra is that when a product of two odd grade objects is taken, the result is always defined over just the even subalgebra, and when the product is taken for two even grade ones, it still is the even subalgebra, so once taking the geometric product, the result of geometric products remains the geometric product, and if we want to keep our multivector data structure relatively small / sparse / compact, we can just not store the odd grade elements and make the geometric product store the even subalgebra alone. In 3D or equivalently in the promoted / projective G(3,0,1), both the even and odd grade objects have geometrically useful properties as physics objects as well, so modeling them separately makes some sense: the even PGA subalgebra forms a screw motion and contains a spinor, while the odd subalgebra makes a flector.

Because these objects are useful, and G(3,0,0) is a subalgebra to G(3,0,1), it may make sense to consider a less obvious progression of grade ordering for the multivectors as well, as we can map them to 4 member subsets that mean several certain objects if carefully mapped to be memberwise equivalent in order. That is to suggest, one might want to map the { 0, 1, 2, 3, 4 } grades in as { 1, { 0, 2, 4 }, 3 } grade ordered objects so the even subalgebra is contiguous and can be read as a screw motion, or perhaps { { 0, 2, 4}, { 1, 3 } } to also group flectors together. Going the next step and grouping all duals is less beneficial in these dimensions though, because taking the { 0 , 4 } breaks the 4 member alignment boundaries, makes the scalars discontinuous with the quaternion spinor components, omits the duals from having a pseudoscalar likewise, and pushes it less helpfully in with the quaternion which as a subalgebra wouldn't include it.

Isomorphism: Many forms of a geometric phenomenon may map to the same result in several algebra: an isomorphism literally means same shape, and implies the arithmetic entailed in manipulating one algebra versus another produce the same result. Quaternions give an excellent example of this, even beyond the above statements. It is true that the real bivectors take on the behavior of complex numbers, but in that sense it is also true that the even subalgebra of the dimension above are isomorphic to the full algebra of the dimension below.

William Rowan Hamilton didn't merely happen upon quaternions: he spent a long time, arduously vexed by the idea. It seemed to him that it was unreasonable that complex numbers worked, but that one couldn't have another set of complex vectors in another direction. He tried to formulate these so called 'couplets' of his and how they would multiply together, but for a very long time it didn't make sense how they would do so. He could add and subtract them, or perform scalar multiplication, but it was unclear what the result would be if he took this i value and j value and they were to both multiply, what would happen which would be a reliable or useful system. Eventually his epiphany came, and he realized they could multiply to a third vector k, equal to i times j, and each other likewise to form the cycle and i \* j \* k all equalling -1 worked out, and he rejoiced for his discovery, but if we think about it under the terms of GA, what he did was much more troubling than that (although it may have worked). In our terms, he saw that he had the scalar, e1 and e2 forming his basis, and that he could form e12 to form a complete algebra, but he did not yet grasp that he had concocted this third object k equivalent to e12 as an entirely different object from what e1 and e2 had been in the rest of his couplet basis.

He had invented 'triplets' as he began to call them, before the name quaternions, but he hadn't thought that what he had done in turning the third to a bivector was that he had also equated the other 2 to bivectors to make those basis elements form the cycle he had envisioned. This is to say he had wanted to work in G(0,2,0), and to make it work, he had promoted the basis to the even subalgebra of G(0,3,0), thinking of them not as a grade (1,2,1) element basis, but (1, 0, 3, 0) ones. The two sets form an isomorphism (within these operations), but one matches the framing and nomenclature consistently, and the other less so: algebraically, they are the same thing for any component, but one requires an explicit understanding of object grade. Further, it is an isomorphism to use the even subalgebra of G(0,3,0) or G(3,0,0) for this, since if one uses scalars, it is the same, and for bivectors only, there is no difference between the basis elements needing an inversion once (for either 030 or 300), to get the bivectors' basis element pairs aligned, and then the two squaring from e1212 = -e1122 of one basis to either -(1)(1) for G(3,0,0), or -(-1)(-1) for G(0,3,0): both equal -1, one has to go through 1 inversion to get there, the other does 3. Ultimately the two choices are isomorphisms to one another (all operations having the same result).

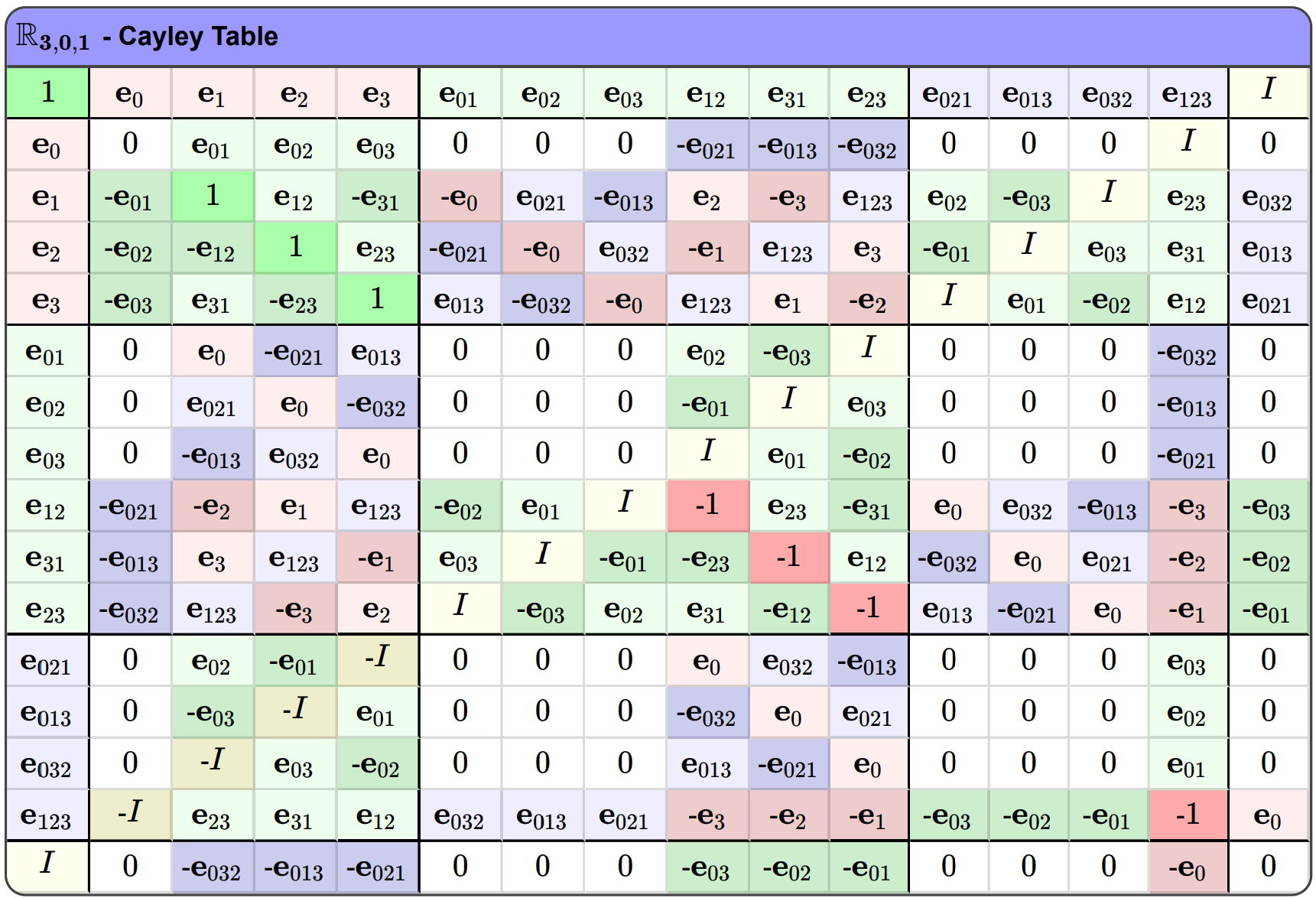
Commutator: Given by the cross symbol (⨯). In general this is the operation in mathematics which measures the portion of the operation which fails to be commutative. Where the inner and outer products get the higher and lower grade objects relative to the inputs, the GA commutator gets the equal grade part. Especially around the medial basis elements for a dimension, one is apt to see elements combine to exterior rejection and cancellation in equal measure in some places, resulting in a portion which doesn't commute to the inner or outer products' grade. In using the commutator product as well, it is possible to cover all parts of the product as fitting neatly into one part versus the other 2, simply by the grade of the resultant element. The definition ⅟₂ (AB - BA) is most often the only one to offer, as it gets at the point of what data it carries.

It can perhaps be vexxing / frustrating attempting to define the component operation of the geometric product in terms of the product itself: while true, it can come off as a circular definition. This is simply to highlight an important property however, but that this part of the definition might justify the namesake, why it sometimes exists, what it algebraically and geometrically represents, and so forth. It is quite possible that a given product will have no commutator components, but when one does take the geometric product, and the anti-commutative complement, it should result in objects whose only difference is in their components which did not commute.

The average of those 2 products which did not cancel out indeed must be the component which remains outside of commutation, hence the commutator which completes the product. The fact that it is defined in this circular way need not be a concern, as it isn't always entirely helpful to attempt to compartmentalize the geometric product into it's component parts: it is a tensor and not to be looked upon in that manner. The fact that this too is a part of the geometric product is simply a way of understanding parts of the whole operation, in geometric terms.

Geometric Product: The core operation of GA, performed through exhaustive distributive factor products over all input basis elements in a pair of multivectors A \* B is AB = A ⋅ B + A ⨯ B + A ∧ B. One may note I have (intentionally), avoided expressly defining the geometric product up until this point: not because it should be obvious or already known, but because it is necessary to get so far as defining the commutator before being able to define the full product.

Without the commutator, it is difficult to see the general case of the even subalgebra of the staple case of G(3,0,1), whether the bivectors which are 3/4 of the product should belong to which seeming more ambiguous, because that is where they fall: from 2 grade 2 inputs to a grade 2 output. Although the shorthand definition AB = A ⋅ B + A ∧ B is mostly what the point of GA is about, and what one is apt to find as the definition in most places, it is not the whole picture which can make matters confusing to learn, and it is important to have the full definition if that is to be prevented.

Caley table: As we lose commutativity and have non Abelian results, it is often best to draw up charts for how the basis elements relate to one another in a given algebra, which besides serving as a point of reference or potential programmatic tool to automate the multiplication, also helps to illustrate how the basis elements invert. If you aren't familiar with Caley tables, they are just the algebraic extension of charts as simple as multiplication tables, but where we lose commutativity symmetry stops being guaranteed, and the result is more of a look into the basis elements' matrix:  


Referring to a table for how each basis element interacts with each other basis element and seeing where it falls can be a good way to ensure transcription and arithmetic errors are mitigated, and can serve as a lookup into one vector element and another to see where the result gets mapped in the output, and whether that result should be inverted.

The basis elements chosen as indices need only represent the sum blades of multivectors entailed in the representative ordering which has been chosen, but there is no right or wrong ordering to those elements (aside perhaps for adhereing to a convention of grouping by grades or progressing in ascending order for legibility, but even that is merely convention, not right or wrong). In the above example, the product of e23 ∧ e021 is given as e013, while e021 ∧ e23 is -e013, but this is to say it could be just as correct to say e23 ∧ e021 = e23021 = e30221 = e30122 = e301 = e013 = e130 = -e310 = -e031, stopping at any ordering once a minimal grade 3 result is extracted (as long as a consistent order is chosen, and redundant, non-spanning squared basis elements are simplified to scalar coefficients). You may also note the steps taken are single digits double transposed over 2 digits to simplify sign reversals: I haven't heard others explicitly articulate doing this in their arithmetic, but I do find it helpful to minimize iterative sign flip errors-by-hand, by relying on 2 digit jumps where possible.

Isometry: Defined as any mapping from one space onto another which conserves both distances and angles: that is operations which do not distory the space by squashing, stretching, skewing shearing or so many of hte operations that are largely undesirable products we can get from linear algebra operations, and have to work to avoid. Linear algebra would also lend insight on an isometry by defining them as the operations with either determinant 1, or -1 (which may say something about why interpolation through reflections can be so bad looking interpolated over traditional matrix transforms, having to go down to 0 medially between those reflections). Much of the power in the geometric operations is in the fact that they adhere to this constraint.

Isometries may not sound like much, but they are a full enough list by inclusion of the identity, reflections, rotations, translations, transflections and rotorotations. These are almost all the transformations we actually desire, and isometries can model any rigid body transform with the added benefit of guaranteeing that the basis that does in maintains a preserved relationship. That is to say if an orthonormal basis goes in, you get one out with no extra work, guaranteed.

Sandwich: Taking the inverse of a transformative operand with another geometric product after, to balance a geometric product taken with an object out, the net result of ABA-1, or A-1BA is the means by which we can use geometric algebra to perform a reflection about that object. By taking the inverse and creating the object which cancels out the geometric distortion of the operation, we get something different than just undoing the transformation: the first part AB will get the grade sum object of the comingling of A and B, a plane times a another getting the line of their intersection (in 3D), a vector times a vector getting a bivector, etc, and then taking that product and dividign out by A-1 again causes a similar transformation again, a normaization / cancelation of magnitude change, and a grade restoration. As such, we get a doubled transformation through A around the gauge of th intersection between A and B (exception free: even parallel objects meeting at infinity still do this, coming out with valid results). It may be familiar to linear algebra students that a reflection is a very similar process with matrices and / or dot products, getting the normal vector to do twice the displacement and subtracting the original to get twice the perpendicular distance in the opposing side by normalizing the parallel ones to cancel out from the doubling.

Reflection: Sandwich products model a reflection across a geometric object, which is fairly significant for a few reasons. For one, it is much more succinct computationally than a matrix, for another, the data gotten out of it is exception free and more geometrically meaningful across the resulting product, beyond that, it can be interpolated as geometrically intended unlike with linear algebra, and perhaps most importantly, not only is it an isometry, but it is a fundamental isometry which can be composed serially to produce any other isometry.

The AB of the sandwich got us the gauge of their intersection to reflect around (either in the space, or at infinity if parallel), but if we then use that as the basis of our sandwich instead (i.e. (AB)v(AB)-1 = ABvB-1A-1), we get a 2-reflection which is a rotation (or translation if rotating about infinity), around that gauge, which is again capable of interpolation (with some caveats on time: technical animator's note, this alone will still have issues with direct linear interpolation of values leading to non-linear temporal interpolation, with quadradic falloff on slow movement speed at the beginning and end, and a very quick jump through the middle, but workarounds can be used with what we've already obvserved in this solution for to solve for that in comparable ways, perhaps even easier as a 1D interpolation geometrically for your parameterization, and more comutationally practical than many slerping methods). This process of serialized reflections can be used to create any isometry in the dimensions available for the given vector space: in 3D a 1-reflection performs a plane reflection 2-reflection performs a line reflection, and a 3-reflection creates a point reflection. These are physical phenomena which may not be familiar, but can be observed by juxtaposing real mirrors orthogonally and looking at the image that is produced cumulatively through them.

As 2 mirrors become parallel, we can see infinite hallways form in the images between them, as they become perpendicular, the image across the 2 becomes a rotated copy of the original, 180 degrees across from the reflectee. You can even read the writing on your shirt in the mirror this way, which is significant as well, as that illustrates how a reflection (an operation with determinant -1, inverting the space), squares to one which restores the image orientation (a rotation being a 2-reflection which (-1)(-1) = 1, gets the space back after the journey into the mirror realm and back a comparable distance). A point reflection gets a third mirror under that to get 3 orthogonal mirrors, and rotates around that point: an inverted cube can be seen across the cumulative image which is upside-down from the original, and has parallels with less contrived physical phenomena perhaps most strongly in how our eyeball forms an image on the back of the retina.

As odd numbers of reflections are taken, some inversions of space will remain, which can be seen in a point reflection in a 180 degree flip of it's cumulative orientation. In segue and by extension (but not to conflate the two), the part that people find less intuitive about rotation is that a full revolution gets up back to this kind of inverted identity, and a second iteration is required to form a complete cycle. Rotations are modeled mathematically in e-i𝝷/2, where dividing by 2 is intuitively necessary through the lens of being thought of as a double reflection, doubling the displacement. This also means that a full revolution is e-i(2𝝿)/2 = e-i𝝿 = ei𝝿 = cos(𝝿) + sin(𝝿)i = -1 + 0 = -1. It seems very backwards on the surface that a full cycle should do anything more than get us back to where we started, but although it seems like a flaw in this system, it actually models reality.

Often the exemplar used to illustrate this fact is to have someone hold an object and rotate it about only one axis, and to observe that you can physically achieve this, but it brings a kinked state up-chain to your arm 'entangling' it, yet through the right additional movements, carrying through this to another revolution sorts the kink out. It is dismissed as a playful, less rigorous example, but may carry more important symmetry than it would seem in how this phenomena mirrors an inversion in the reference frame. It also isn't isolated to the mechanics of a human arm (or how that parallel is driven in guiding the piloting of a robotic arm for that matter), but also mirrors how entangled neutrons will interact differently after a 360 degree revolution, but not after 720 degree spins.

Projective Model: Equivalent to using homogeneous coordinates, using one extra dimension of dual numbers in the basis (i.e. G(3,0,1) instead of G(3,0,0), the most common algebra for 3D PGA, or G(2,0,0) -> G(2,0,1) for 2D). Sometimes a frowned upon namesake, seen as a misnomer in some lights or confusing some terms, this is more in reference to the way a projector carries light and brings it out to a different dimension, and less the mathematical meaning of a projection where we look at how the 3D data is transformed by being mapped to one of 1 lower dimensional span, but this is called projective space as the added dual dimension makes the other dimensions that can relate to each other in a linear space about the origin to an affine one where the dual number transforms the space of those dimensions in their displacement from the origin with no other distortion, which can be described as "projecting" that coordinate space into a common reference frame by translations, and is perhaps one which can be viewed comparably in that the geometric data is a dimension higher than the places wehre the data is ultimately being mapped, it isn't entirely a misnomer, if confusingly so to some.

The geometric representation of this algebra also converts the flavor of k-vectors to k-reflections, which can seem less intuitive than the VGA explanations of k-vectors. Perhaps this can be seen to make sense in the same way a revolution or odd number of reflections invert something intrinsically, similarly (though not to imply a causal link), PGA uses 1-vectors as VGA (d-1)-vectors, 2-vectors are (d-2)-vectors, etc. This is accurate in modeling the number of reflections being used, which is more geometrically related to the resultant objects of the operations matching the data expected. This recieves some degree of ire however, from students already learning a lot who see this apparent inversion as "doing things backwards". It isn't impossible to do PGA without this inversion, and it is understandable why it seems like a step making things harder than necessary if poorly explained. All I can really say in justification is that modeled one way vs. the other, the algebra which results appears more backwards, and apart from the expectations if taken in the VGA flavor. It is also a good means of inroduction to becoming comfortable with the tendency for new flavors of GA to cause subtle shifts in our understandings of terms in general.

The homogeneous promotion by 1 dimension typically sees e0 used to represent the dual addition. Provided the duals aren't used for multiple dimensions of duals, this allows the consecutive integers to act as labels for all basis elements so simply e# suffices for any other G(p, q, 1) combination of basis terms, which covers most applied purposes adequately and servicably.

Still this can be frustrating for some, as it means what we are modeling in the affine space is somewhat inverted to the objects we are interested in representing: a plane-reflection is the 1-blade representing its normal, the line-reflection is the 2-blade representing its plucker cooridnates, the point reflection is the 3-blade representing the spatial constraints which map to holding that point invariant. These are ideal terms describing how an object is defined by exclusion, but not how to represent that object. All this means however is that the result is closely related to the dual of the object, and we can use matrix math to convert back to basis terms we're used to in order to be represented in conventional geometric sensibilities after we're done with the basic transformations we want to render upon them first.

Rotors in G(3,0,0): GA quaternions, modeling a rotation between a pair of input vectors, their product forming the even subalgebra which represents the gauge of rotation around their intersection to get the angle between them, and can be composed as any number of rotations by continuing to take the product of one rotation with another to produce their cumulative rotation. To visualize the rotation, the plane and orientation formed by the resulting bivector plane of a pair of input vectors sweeps the oriendation of rotation.

Those bivectors can be viewed as the normal to that plane, andif unitized with the scalar, their product represent the cartesian coordinates where that scalar continues to represent the cosine of the angle about which the plane sweeps the rotation, while the euclidean length of the other 3 combine to from the sin of that angle, their proportion between each other defining the normal direction itself. this explains how the values can interpolate the rotation, and why linear interpolation would result in the values not sweeping the unit circle of those coordinates consistently: the magnitude doesn't stay the same (the inverse takes care of that), and more importantly the angles subtended don't evenly sweep across the arclength of that displacement.

Much like with quaternions, a so called 'pure' quaternion where the scalar value is 0, rotors with only bivector coefficients have only a sine value and a 0 in cosine. In this case, not only do the 3 components represent the normal axis about which the rotation spins (therefore it's eigenvector), in having a magnitude with no cosine, either one represents an orthogonal rotation around that axis. This also means that one quirk of either rotors or quaternions is that as the rotations fall to squares that are pure cosine values, they lose precision as to what axis defines the rotation and as the rotation becomes approximately null (or an inversion if negative in the scalar field).

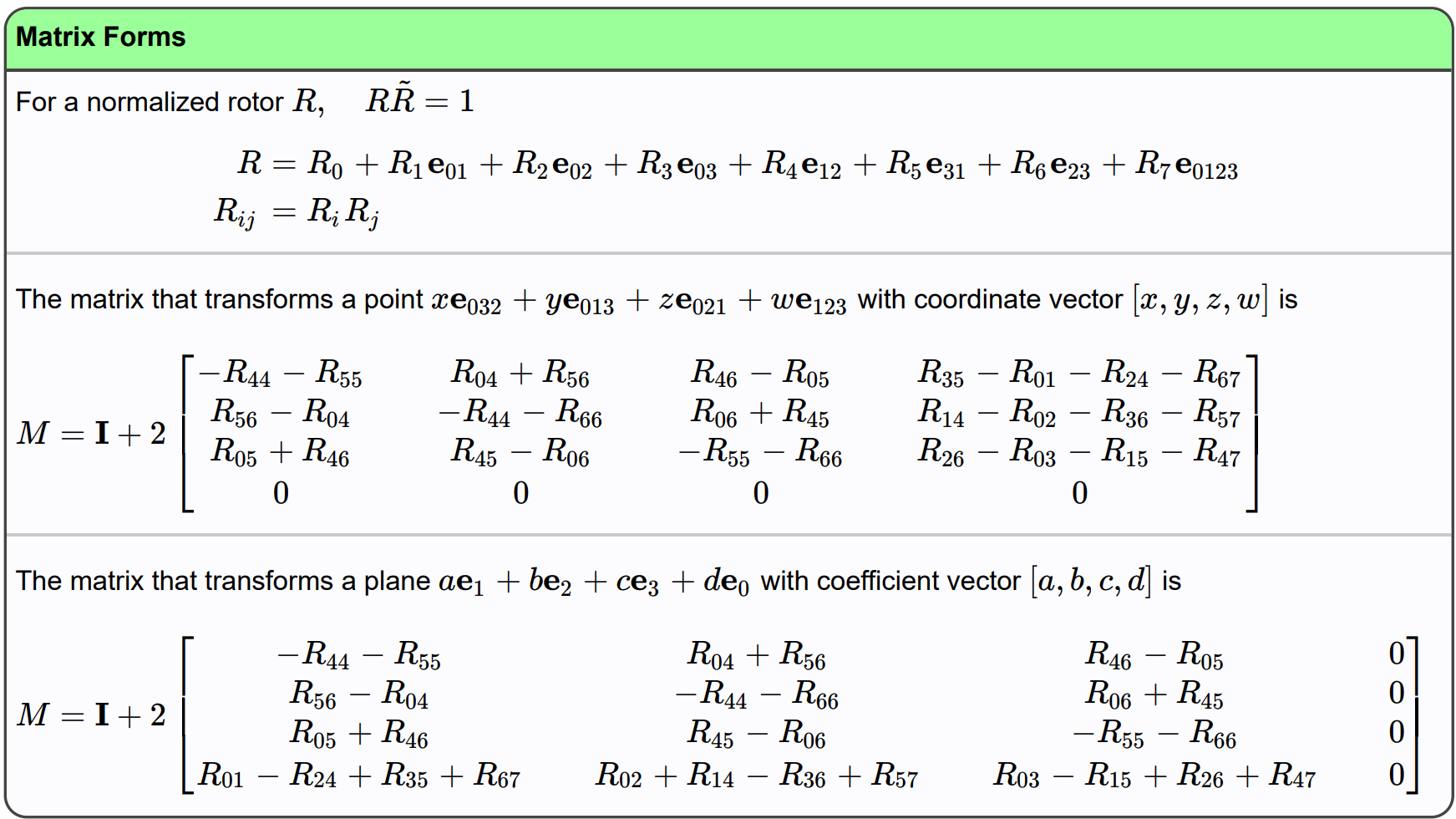
Both memberwise and operationally, the conventional quaternion, the even subalgebra of G(0,3,0) and the even subalgebra of G(3,0,0) are isomorphic to one another, and mostly a matter of labeling convention differences, at most depending on an order change (w = scalar, i = e23, j = e31, k = e12). It isn't so much that GA advocates have any desire to 'remove' quaternions from game engines per se, it's just that the lables chosen give very little explanatory power to what operations are being performed, while the GA labels the explanations given above make a more geometric interpretation viable and promotes ideas for how to visualize thosegeometric interpretations relatably.

Rotors / Motors in G(3,0,1): GA dual quaternions modeling a screw motion, the geometric cumulative representation of the even subalgebra, modeling the motion's axis, it's angle, and the 'screwiness' (or the thread sizing of it). Using the 6 coefficients of the bivectors models the plucker coordinates which may be familiar to physics students, giving us the direction bivectors in the coefficients without the dual dimension, and the moment vector in those bivectors which do contain it. In tandem, the norm of the bivectors will have the direction vector length only, and if normalizing as a set, the length of the moment will give the distance which the direction vector is from the origin (i.e. the dualized radius of the screw motion), as well as the axis the screw translates along. So called the moment as it related to the moment of inertia perpendicular to the direction vector, and between them, describing the linear combination of all points which fit this constraint the same way a plane may familiarly describe all the points which fit a given sum by the normal coefficients. The scalar describes the amount of rotation taking place, so a 0 scalar signifies an orthogonal rotation amount. The pseudoscalar then remains signifying the scale of screw motion, the rate at which the motion takes place, hence screwiness / thread size.

Where the input planes are parallel, the 'rotor' formed will be entirely dual, and represent a pure translation, while a 'pure' rotor is as a 'pure' quaternion: all the non-dual bivector values will be the only non-zero ones. Conventionally, pure translations are called a 'motor', and some combination of the two is a screw motion, or more formally perhaps: a rotoreflection.

Much the same as quaternions and rotors don't stand in opposition to one another, neither do the dual quaternions stand in opposition to the homogeneous rotors / motors /screws: the two cooperate and their differences are also a matter of labels and ordering.

Representation: Converting a rotor to matrix form can be less obvious in PGA, dealing with a transformation on a point or plane still fairly integral to working with traditional shaders and outputting at some point. I don't know what I can say to add to my preferred from cheat sheet from De Keninck's definitions on the matter, except to say it is noteworthy that the linear 3x3 elements are uniform in both conversion of a given point or plane, and the plane being the one with non-standard homogeneous promotion may make some sense between the two. a point having an affine column transform to relate the coordinate transform is fairly conventionally straightforward, and the plane reflection to normal vector is a dualization process, so it isn't that strange to see non-zero bottom row values, just as normal matrix's linear inverse transpose may have these characteristics.



Conformal Model: The conformal model adds 2 dimensions unlike homogeneous promotion, but neither is a dual one: conformal modeling adds G(+1, +1, 0), and the two define the origin and infinity, with vectors taking on the representation of circles in this flavor, doubling back on themselves and coming between the origin and infinity. Due to these additions, it allows the additional features of dialating space. The origin o = ½(e- + e+) while ∞ = (e- - e+). Both ∞ and o square to 0, while a given point p = o + p + ½p2∞. I am still working on my understanding of PGA, and CGA is fascinating, but given the 2 additional dimensions of promotion, and the fact that each dimensions shows an O(n!) growth rate (adding a factor of n in basis vectors which have factorial additional elements each dimension), this means that PGA is 4 times more computation than VGA, but to produce affine math which is still less than the rate of 4x4 matrices extra work, but CGA is 5 times more work arithmetically than PGA, and so is usually not seen as the favorable path forward in modeling simulations in realtime.

It is important to learn in an eventual, and CGA is foundational to understanding spacetime algebra, so if the phenomena one wants to model is more astronomical in nature / scale, it is useful to learn more about. I will not pretend to be more studied in this than having looked at it briefly, but it is worth knowing what is worth looking for in CGA, what it might offer, where it may be able to better lead someone, etc. These are the fields of study where becoming PhD level makes more sense, and while it is learnable, it isn't necessarily in scope or within the needs of graphics rendering.

Geometric Calculus: The geometric derivative unifies much of physics, and while it was outside of my scope to study, I can say that the geometric product continues to relate to it. Continued derivatives and integrals are somewhat simplified as well with the way the product unifies linear and angular forces into one object, making the applications simpler as well. I certainly hope to study more into geometric calculus, but insofar it has been outside of my expertise to research.

# Summary

The rabbithole of geometric algebra is quite deep, and for all it unifies and simplifies, not casting anything away per se, it can simultaneously serve to upend a great deal of feelings of familiarity with the basics, and I certainly felt a degree of backsliding in order to take two steps forward later over this term. I have undoubtably learned a great deal of the fundamentals, especially insofar as VGA, and somewhat PGA. Each flavor has its own hurdles to overcome, and adds geometric capabilities to merit learning about, but it is also important to have a mindset of taking on only what is necessary to learn right now to not get lost in the weeds of it all.

I have tried to cover the basics and what has gone into the study that took the whole of the semester to get my project up to where it now is in the vocabulary section. As long as it goes on in detailing history and giving exposition to what can be confusing or helpful around a given term, I hope that much suffices in validating how much research has taken place this term on the subject.

# References

Reviews added for context on utility as well

* Alan Macdonald <http://www.faculty.luther.edu/~macdonal/> (note: no trailing d)  
  *Linear and Geometric Algebra: Volume 1* **978-1453854938**  
  *Vector and Geometric Calculus: Volume 2* **978-1480132450**  
  <https://www.youtube.com/@AlanMacdonald1/playlists>  
   A staple resource for getting into GA, the first volume is designed to serve as a classroom textbook for an integrated linear and geometric algebra syllabus with a focus on geometric algebra, without coming at the expense of staple linear algebra literacy to so many academic accredidation standards. The chapters do a good job of explaining the basics, but do go through a pattern of frequently suggesting it is the reader's responsibility to prove a principle to be true using the given statements. It is usually reasonable given the materials, but also a lot to ask for an undergraduate course on linear algebra without also supplying answers and steps, and risks losing the less mathematically inclined or already literate readers in the process. A good textbook for a class which plans to introduce these subjects with the guidance of a teacher to aid when a student gets lost and needs a hand, but not the ideal material for independent reading and enrichment to cover these materials. Still passable for these purposes, and covers the fundamentals for LA, VGA, PGA and CGA in good order without getting hung up on details along the way, and an excellent segue to volume 2 if one is to continue these studies in pursuit of a more unified understanding of physics and thegeometric derivative. He has supporting playlists on his youtube page for either textbook which also do a great job of explaining the foundational principles.
* Eric Lengyel  
  *Foundations of Game Engine Development: Volume 1, Mathematics* **978-0-9858117-4-7**  
  <https://projectivegeometricalgebra.org/> <http://www.terathon.com/lengyel/>  
   Very accessible introductory material for integrated linear and geometric algebra, but mostly from a linear algebra basis with geometric algebra introductions, not quite sufficient to have fully working implementation (despite included PGA C++ code and supportive text). Lengyel's contributions are easy to follow for a wider audience due to incusion of code, and his background in computer science and games makes the flavor very easy to onboard to new GA terms with fewer translation issues. His notation for doing GA are however some of the most non-conventional popularizations compared to the rest of the field, and some of those easier to follow examples can prove to be a crutch if the lessons he offers don't go deep enough and one needs to look elsewhere for more study. Lengyel has tools and papers for GA, but the FGED volumes are mostly LA materials, and while he purports in volume 1 (from 2016), that he plans to say more in volume 3 on exterior algebra and PGA, dual quaternions, etc. and I have been eagerly awaiting that for some time. His tweets during this year's GDC (relating to being fed up with students unabashedly using pirated copies of his first textbooks in front of him in his own class, and not wanting to proceed with digital printings of his future books), were not inspiring confidence however that these texts might come sooner than later. I found the parallels drawn very useful and easy to follow, but also got hung up by his choices at many points this semester: recommended if taken with a grain of salt, and in tandem with other sources. His operations are also controversial because some like his antiproduct are technically not geometric, and introduce remainders without clarifying the fact, so they can especially frustrate new learners. Enki took the time to elucidate why they are controversial very articulately here:  
  [https://enki.ws/ganja.js/examples/coffeeshop.html#Ee6ips2i4](https://enki.ws/ganja.js/examples/coffeeshop.html%23Ee6ips2i4)  
  I still find his materials compelling, accessible and helpful, but it should definitely not be one's learning material in isolation, or it could be easy to be led astray, without good idea why things don't always seem to work as expected. Game articles, CG talks & the C4 engine also on his terathon site
* **Leo Dorst, Chris Doran, Joan Lasenby**  
  *Applications of Geometric Algebra in Computer Science and Engineering,* **978-1461266068**  
   Likely a good, rigorous textbook for a CS class akin to this independent study attempt: more thorough than most, and edited fairly thoroughly yet still over 500 pages long, having some of the most up to date work compiled I've found in a traditional math textbook on the subject, but, focused on GA applications directly, so perhaps less ideal to introduce the math of linear algebra concurrently
* **David Hestenes**  
  *Clifford Algebra to Geometric Calculus*, **978-9027725615**  
   Hestenes has been very active in pioneering and evangelizing the use of geometric algebra since the 1960's, but with a very strong focus on the conformal model & a slant more focuses towards the use of geometric calculus to unify physics. Robust and noble goals, but not always lightweight enough to be useful for realtime applications. A good name to bear in mind as ones studies advance.
* **Steven (Enki) De Keninck, Charles Gunn, Leo Dorst**  
  <https://bivector.net/> ; <https://www.youtube.com/@bivector/playlists>  
  <https://www.youtube.com/watch?v=tX4H_ctggYo> Siggraph '19 presentation introducing PGA math  
  <https://www.youtube.com/watch?v=ichOiuBoBoQ> Game '20 talk explaining Dual Quaternions  
  <https://www.youtube.com/playlist?list=PLsSPBzvBkYjxrsTOr0KLDilkZaw7UE2Vc> PGA tut. series  
   Forums, research documents, operational crib sheets, discord server, links to starter videos, links to github pages for libraries PGA in several code languages being actively developed and updated, etc. Likely the largest single repository for finding help in learning anything to do with GA or GC: a very active and helpful community which has been instrumental to the success I did have this term. I didn't get as far as I'd hoped, mostly because I was so below the bar of starting literacy at the beginning relative to where the average community newcomer might be, let alone the experts. People there are educators formalizing their understandings and lesson plans, and digging into less introductory materials themselves: many mathemeticians and physicists concerned with formalism, as well as CS folk who are looking for optimized math alternatives. Just following discussions there in real time helped tremendously to build literacy. Even if the help offered when I had questions often went over my head, a number of people are better at simplifying there as well. Most are interested in more unified physics as it relates to more applications, and geometric calculus is a major focus for much of the research being organized / studied there, but that deeper ability to use the basics redoubled interest from both parties more than splitting focus. Enki's PGA talks focus on GA applications are are the most prevalent materials cited in evangelizing the applications of GA, but as useful as all of these resources are, and seeing live code demo from his coffeeshop or perusing ganja.js can be illuminating or inspiring, he tends to be very all or nothing in breaking algebra down to component arithmetic, so his work can be difficult to follow when you try to get into how it is working: either having pure multivector arrays taking all products by sequential subscripts, or pure symbolic simplification in an attempt at unification, which may be less useful at early stages of learning due to being too abstracted to implement explicitly, leading to unimplementable, circularly defined function definitions in terms of the others through various identities...which can be frustrating and serve to make people not believe in the material actually working
* **Matt Ferraro**  
  <https://mattferraro.dev/posts/geometric-algebra>  
   A good explanation of the basics of VGA with rotatable visuals live for enhanced clarity. A very easy to follow resource to get comfortable with basic terminology. Bear in mind VGA basics can be misleading in the face of learning PGA k-vector interpretations
* **Martin Roelfs**  
  <https://www.youtube.com/@ArrowofEntropy/playlists>  
   Linked on the head of Bivector and well recommended. Only a few videos, and some of the most relatively short ones I've seen, but the core conceptual differences are highlighted succinctly and (mostly), covered well. But one does need to keep on top of his algebra, to follow where he mis-speaks in reference of vector u in place of v during part 2, so that's with a dash of correcting for your math teacher's whiteboarding mixed in. At cumulatively under an hour, a very good whirlwind tour
* **Sudgylacmoe**  
  <https://www.youtube.com/@sudgylacmoe/playlists>  
   Excellent video playlists for GA introductions. Miscellaneous Math over Zero to Geo recommended currently, only since Misc. Math is all deep dives and 'swift introductions', while Zero to Geo is still in early stages of content development and the basics may be covered elsewhere better. He preferred I use his handle, but if
* Hamish Todd:  
  <https://hamishtodd1.github.io/> Personal site   
  <https://hamishtodd1.substack.com/p/how-to-picture-the-pga-pseudoscalar> Visualization article  
  [­https://hamishtodd1.substack.com/p/a-clifford-algebra-encompassing-rotations](%1fhttps:/hamishtodd1.substack.com/p/a-clifford-algebra-encompassing-rotations) Alternative CGA  
  <https://gdcvault.com/play/1029233/Math-in-Game-Development-Summit>  
   Math teacher and GPU programmer, Todd's presentations strike a very good balance on a combination of foundations, rigor, clear visual exemplars, and code examples. He is also exceedingly friendly and eager to help during my conversations on the bivector discord with him, and his GDC talks from this year were excellent (the last link, with part 2 available in time or with a vault pass)