Geometric, Wedge and Related Products

Kurt Nalty

July 31, 2015

Abstract

Geometric algbebra has decent consensus on the meaning of the geometric, wedge, symmetric and anti-symmetric products. However, there are conflicting definitions for various dot products, and dual forms, nicely detailed in Appendix B of Geometric Algebra for Computer Science [5]. This is my reference sheet for the three dimensional Euclidean geometric, wedge, symmetric, anti-symmetric, regressive (anti-wedge), join and meet products. Code is posted at `http://www.kurtnalty.com/Tests.GA3DE_3_0_0.ginac.cp`

3D Euclidean Geometric Algebra Basis

Three dimensional Euclidean geometrical algebra has a scalar (1), three vectors (e_x, e_y and e_z), three bivectors (e_xe_y, e_ze_x, and e_ye_z), and one trivector ($e_xe_ye_z$) defining the geometry. In multiplication table format, the order-sensitive multiplication among these elements, with prefactors on the left column and postfactors on top row, is

	1	e_x	e_y	e_z	e_xe_y	e_ze_x	e_ye_z	$e_xe_ye_z$
1	1	e_x	e_y	e_z	e_xe_y	e_ze_x	e_ye_z	$e_xe_ye_z$
e_x	e_x	1	e_xe_y	-e_ze_x	e_y	$-e_z$	$e_xe_ye_z$	e_ye_z
e_y	e_y	$-e_xe_y$	1	e_ye_z	$-e_x$	$e_xe_ye_z$	e_z	e_ze_x
e_z	e_z	e_ze_x	$-e_ye_z$	1	$e_xe_ye_z$	e_x	$-e_y$	e_xe_y
e_xe_y	e_xe_y	$-e_y$	e_x	$e_xe_ye_z$	-1	e_ye_z	$-e_ze_x$	$-e_z$
e_ze_x	e_ze_x	e_z	$e_xe_ye_z$	$-e_x$	$-e_ye_z$	-1	e_xe_y	$-e_y$
e_ye_z	e_ye_z	$e_xe_ye_z$	$-e_z$	e_y	e_ze_x	$-e_xe_y$	-1	$-e_x$
$e_xe_ye_z$	$e_xe_ye_z$	e_ye_z	e_ze_x	e_xe_y	$-e_z$	$-e_y$	$-e_x$	-1

3D Euclidean Geometric Product

In component form, the geometric product in 3D Euclidean geometry is

```
GA3DE operator*(const GA3DE &a, const GA3DE &b) {
    GA3DE c;

    c.q   = + a.q*b.q   + a.x*b.x   + a.y*b.y   + a.z*b.z
            - a.xy*b.xy  - a.zx*b.zx  - a.yz*b.yz  - a.xyz*b.xyz;
    c.x   = + a.q*b.x   + a.x*b.q   - a.y*b.xy  + a.z*b.zx
            + a.xy*b.y   - a.zx*b.z   - a.yz*b.xyz - a.xyz*b.yz ;
    c.y   = + a.q*b.y   + a.x*b.xy  + a.y*b.q   - a.z*b.yz
            - a.xy*b.x   - a.zx*b.xyz + a.yz*b.z   - a.xyz*b.zx ;
    c.z   = + a.q*b.z   - a.x*b.zx  + a.y*b.yz  + a.z*b.q
            - a.xy*b.xyz + a.zx*b.x   - a.yz*b.y   - a.xyz*b.xy ;
    c.xy  = + a.q*b.xy  + a.x*b.y   - a.y*b.x   + a.z*b.xyz
            + a.xy*b.q   + a.zx*b.yz  - a.yz*b.zx  + a.xyz*b.z  ;
    c.zx  = + a.q*b.zx  - a.x*b.z   + a.y*b.xyz + a.z*b.x
            - a.xy*b.yz  + a.zx*b.q   + a.yz*b.xy  + a.xyz*b.y  ;
    c.yz  = + a.q*b.yz  + a.x*b.xyz + a.y*b.z   - a.z*b.y
            + a.xy*b.zx  - a.zx*b.xy  + a.yz*b.q   + a.xyz*b.x  ;
    c.xyz = + a.q*b.xyz + a.x*b.yz  + a.y*b.zx  + a.z*b.xy
            + a.xy*b.z   + a.zx*b.y   + a.yz*b.x   + a.xyz*b.q  ;

    return c;
}
```

Note Alan Bromborsky [2] uses e_{xz} rather than e_{zx} for his canonical basis in the Sympy package. John Browne [3] also uses the e_{xz} basis in his book and Mathematica GrassmanAlgebra package. As $e_x e_z = e_z e_x$, there will be a sign difference between my c.zx versus their c.xz components.

Geometric Square Product

Given a generic multivector

$$G = g.q + g.xe_x + g.ye_y + g.ze_z + g.xye_xe_y + g.zxe_ze_x + g.yze_ye_z + g.xyze_xe_ye_z$$

The square of this multivector, in component form, is

$$\begin{aligned}
s &= GG \\
s.q &= g.q * g.q + (g.x * g.x + g.y * g.y + g.z * g.z) \\
&\quad -(g.xy * g.xy + g.zx * g.zx + g.yz * g.yz) - g.xyz * g.xyz \\
s.x &= 2 * (g.q * g.x - g.yz * g.xyz) \\
s.y &= 2 * (g.q * g.y - g.zx * g.xyz) \\
s.z &= 2 * (g.q * g.z - g.xy * g.xyz) \\
s.xy &= 2 * (g.q * g.xy + g.z * g.xyz) \\
s.zx &= 2 * (g.q * g.zx + g.y * g.xyz) \\
s.yz &= 2 * (g.q * g.yz + g.x * g.xyz) \\
s.xyz &= 2 * (g.q * g.xyz + g.x * g.yz + g.y * g.zx + g.z * g.xy)
\end{aligned}$$

Symmetric and Antisymmetric Multivector Product

The symmetric and antisymmetric products of A and B are

$$\begin{aligned}
S &= \frac{1}{2}(AB + BA) \\
A &= \frac{1}{2}(AB - BA)
\end{aligned}$$

In component form, we have

```
GA3DE Symmetric(const GA3DE &a, const GA3DE &b) // Symmetric product
{
    GA3DE S;
    S.q   = a.q*b.q   + a.x*b.x + a.y*b.y + a.z*b.z
            - a.xy*b.xy - a.xz*b.xz - a.yz*b.yz - a.xyz*b.xyz ;
    S.x   = a.q*b.x   + a.x*b.q   - a.yz*b.xyz - a.xyz*b.yz ;
    S.y   = a.q*b.y   + a.y*b.q   - a.xz*b.xyz - a.xyz*b.xz ;
    S.z   = a.q*b.z   + a.z*b.q   - a.xy*b.xyz - a.xyz*b.xy ;
    S.xy  = a.q*b.xy  + a.z*b.xyz + a.xy*b.q   + a.xyz*b.z ;
    S.zx  = a.q*b.xz  + a.y*b.xyz + a.xz*b.q   + a.xyz*b.y ;
    S.yz  = a.q*b.yz  + a.x*b.xyz + a.yz*b.q   + a.xyz*b.x ;
    S.xyz = a.q*b.xyz + a.x*b.yz  + a.xy*b.z + a.xyz*b.q
            + a.xz*b.y + a.y*b.xz + a.yz*b.x + a.z*b.xy ;
    return S;  }
```

```
GA3DE Antisymmetric(const GA3DE &a, const GA3DE &b) // Antisymmetric product
{
    GA3DE A;

    A.q   = 0;
    A.x   = a.z*b.xz - a.y*b.xy + a.xy*b.y  - a.xz*b.z  ;
    A.y   = a.x*b.xy - a.z*b.yz + a.yz*b.z  - a.xy*b.x  ;
    A.z   = a.y*b.yz - a.x*b.xz + a.xz*b.x  - a.yz*b.y  ;
    A.xy  = a.x*b.y  - a.y*b.x  + a.xz*b.yz - a.yz*b.xz ;
    A.zx  = a.z*b.x  - a.x*b.z  + a.yz*b.xy - a.xy*b.yz ;
    A.yz  = a.y*b.z  - a.z*b.y  + a.xy*b.xz - a.xz*b.xy ;
    A.xyz = 0;
    return A;
}
```

Wedge Product

In the wedge product, the directional vector basis anti-commute, while an overall scalar term commutes (reference David Hestenes [7] (p. 6))

The wedge product in component form (Alan Bromborsky [1] and Eric Langyel [9]) is

```
GA3DE Wedge(const GA3DE &a, const GA3DE &b) // wedge product
{
    GA3DE c;

    c.q   = + a.q*b.q;
    c.x   = + a.q*b.x   + a.x*b.q;
    c.y   = + a.q*b.y   + a.y*b.q;
    c.z   = + a.q*b.z   + a.z*b.q;
    c.xy  = + a.q*b.xy  + a.x*b.y  - a.y*b.x  + a.xy*b.q;
    c.zx  = + a.q*b.zx  - a.x*b.z  + a.z*b.x  + a.zx*b.q;
    c.yz  = + a.q*b.yz  + a.y*b.z  - a.z*b.y  + a.yz*b.q;
    c.xyz = + a.q*b.xyz + a.x*b.yz + a.y*b.zx + a.xy*b.z
            + a.z*b.xy  + a.zx*b.y + a.yz*b.x + a.xyz*b.q;
    return c;
}
```

The wedge product has the property of preserving reasonable units of measure in its product. For a pair of multivectors where the scalar is a pure number, the vector is measured in meters, the bivector is measured in square meters, and the trivector is measure in cubic meters, their product has

the same units for the corresponding components, meaning that the scalar product is still dimensionless, the vector portion is in meters, bivector portion in square meters, and trivector in cubic meters. (Inverse units, such as wavenumber in m^{-1}, m^{-2}, and m^{-3} work as well.) Consequently, multivectors using the wedge product and graded units of measure are closed under both addition and multiplication.

```
r = (a.q, L*a.x, a.y*L, L*a.z, L^2*a.xy, a.zx*L^2, L^2*a.yz, L^3*a.xyz)
s = (b.q, L*b.x, L*b.y, L*b.z, L^2*b.xy, L^2*b.zx, L^2*b.yz, b.xyz*L^3)
Wedge Product with Units (L)
u.q   =       + a.q*b.q
u.x   =   L*( + a.x*b.q + a.q*b.x )
u.y   =   L*( + a.y*b.q + a.q*b.y )
u.z   =   L*( + a.z*b.q + a.q*b.z )
u.xy  = L^2*( + a.q*b.xy + a.x*b.y - a.y*b.x + a.xy*b.q )
u.zx  = L^2*( + a.q*b.zx - a.x*b.z + a.z*b.x + a.zx*b.q )
u.yz  = L^2*( + a.q*b.yz + a.y*b.z - a.z*b.y + a.yz*b.q )
u.xyz = L^3*( + a.q*b.xyz + a.x*b.yz + a.y*b.zx + a.z*b.xy
              + a.xy*b.z + a.zx*b.y + a.yz*b.x + a.xyz*b.q )
```

While squared individual basis vectors zero out in the wedge product, the general multivector *does not* square to zero. Higher powers have an obvious pattern.

```
r = (a.q, L*a.x, a.y*L, L*a.z, L^2*a.xy, a.zx*L^2, L^2*a.yz, L^3*a.xyz)
Square Wedge with Units (L)
u.q   = a.q^2
u.x   = 2*L*a.q*a.x
u.y   = 2*L*a.q*a.y
u.z   = 2*L*a.q*a.z
u.xy  = 2*L^2*a.q*a.xy
u.zx  = 2*L^2*a.q*a.zx
u.yz  = 2*L^2*a.q*a.yz
u.xyz = 2*L^3*(a.q*a.xyz + a.x*a.yz + a.y*a.zx + a.z*a.xy)

Cube Wedge with Units (L)
u.q   = a.q^3
u.x   = 3*L*a.q^2*a.x
u.y   = 3*L*a.q^2*a.y
u.z   = 3*L*a.q^2*a.z
u.xy  = 3*L^2*a.q^2*a.xy
u.zx  = 3*L^2*a.q^2*a.zx
u.yz  = 3*L^2*a.q^2*a.yz
u.xyz = 3*L^3*(2*a.q*a.x*a.yz + 2*a.q*a.y*a.zx + 2*a.q*a.z*a.xy + a.q^2*a.xyz)
```

```
r = (a.q, L*a.x, a.y*L, L*a.z, L^2*a.xy, a.zx*L^2, L^2*a.yz, L^3*a.xyz)
Quartic Wedge with Units (L)
u.q   = a.q^4
u.x   = 4*L*a.q^3*a.x
u.y   = 4*L*a.q^3*a.y
u.z   = 4*L*a.q^3*a.z
u.xy  = 4*L^2*a.q^3*a.xy
u.zx  = 4*L^2*a.q^3*a.zx
u.yz  = 4*L^2*a.q^3*a.yz
u.xyz = 4*L^3*(3*a.q^2*a.x*a.yz + 3*a.q^2*a.z*a.xy + 3*a.q^2*a.y*a.zx + a.q^3*a.xyz)

r = (a.q, L*a.x, a.y*L, L*a.z, L^2*a.xy, a.zx*L^2, L^2*a.yz, L^3*a.xyz)
Quintic Wedge with Units (L)
u.q   = a.q^5
u.x   = 5*L*a.q^4*a.x
u.y   = 5*L*a.q^4*a.y
u.z   = 5*L*a.q^4*a.z
u.xy  = 5*L^2*a.q^4*a.xy
u.zx  = 5*L^2*a.q^4*a.zx
u.yz  = 5*L^2*a.q^4*a.yz
u.xyz = 5*L^3(4*a.q^3*a.x*a.yz + 4*a.q^3*a.y*a.zx + 4*a.q^3*a.z*a.xy + a.q^4*a.xyz)

r = (a.q, L*a.x, a.y*L, L*a.z, L^2*a.xy, a.zx*L^2, L^2*a.yz, L^3*a.xyz)
Hexic Wedge with Units (L)
u.q   = a.q^6
u.x   = 6*L*a.q^5*a.x
u.y   = 6*L*a.q^5*a.y
u.z   = 6*L*a.q^5*a.z
u.xy  = 6*L^2*a.q^5*a.xy
u.zx  = 6*L^2*a.q^5*a.zx
u.yz  = 6*L^2*a.q^5*a.yz
u.xyz = 6*L^3*(5*a.q^4*a.x*a.yz + 5*a.q^4*a.y*a.zx + 5*a.q^4*a.z*a.xy + a.q^5*a.xyz)
```

Symmetric and Antisymmetric Wedge Product

Due to the presense of the scalar terms, the wedge product is not purely antisymmetric. The means we have non-zero symmetric and antisymmetric wedge product combinations.

```
r = (a.q, L*a.x, a.y*L, L*a.z, L^2*a.xy, a.zx*L^2, L^2*a.yz, L^3*a.xyz)
s = (b.q, L*b.x, L*b.y, L*b.z, L^2*b.xy, L^2*b.zx, L^2*b.yz, b.xyz*L^3)
```

```
Symmetric Wedge with Units (L)
u.q   =       + a.q*b.q
u.x   =   L*( + a.q*b.x + a.x*b.q )
u.y   =   L*( + a.q*b.y + a.y*b.q )
u.z   =   L*( + a.q*b.z + a.z*b.q )
u.xy  = L^2*( + a.q*b.xy + a.xy*b.q )
u.zx  = L^2*( + a.q*b.zx + a.zx*b.q )
u.yz  = L^2*( + a.q*b.yz + a.yz*b.q )
u.xyz = L^3*( + a.q*b.xyz + a.x*b.yz + a.y*b.zx + a.z*b.xy
              + a.xy*b.z + a.zx*b.y + a.yz*b.x + a.xyz*b.q )

r = (a.q, L*a.x, a.y*L, L*a.z, L^2*a.xy, a.zx*L^2, L^2*a.yz, L^3*a.xyz)
s = (b.q, L*b.x, L*b.y, L*b.z, L^2*b.xy, L^2*b.zx, L^2*b.yz, b.xyz*L^3)
Antisymmetric Wedge with Units (L)
u.q   = 0
u.x   = 0
u.y   = 0
u.z   = 0
u.xy  = L^2*( + a.x*b.y - a.y*b.x )
u.zx  = L^2*( + a.z*b.x - a.x*b.z )
u.yz  = L^2*( + a.y*b.z - a.z*b.y )
u.xyz = 0
```

Pseudovector Product (Dual)

In three dimensional Euclidean geometric algebra, the trivector basis squares to negative one.

$$(e_x e_y e_z)(e_x e_y e_z) \quad = \quad e_x e_y e_z e_x e_y e_z = -1$$

The trivector also has the property of commuting with all multivector elements, and thus shares the algebraic properties of $i = \sqrt{-1}$. Consequently, we commonly see notations such as $I = e_x e_y e_z$ for the trivector basis.

In three dimensions, the pseudoscalar commutes with all elements, but in general, (for example, 2D and 4D space) the highest grade element does not commute. Consequently authors, such as Dorst [5], p 82, will have have differing definitions for the left side versus right side dual product. In three dimensions, left versus right does not yet matter.

Commutivity means $IM = MI$. However, $MI \neq M$. In component form,

$$\begin{aligned}
M &= m.q + m.x\ e_x + m.y\ e_y + m.z\ e_z \\
&\quad + m.xy\ e_x e_y + m.zx\ e_z e_x + m.yz\ e_y e_z + m.xyz\ e_x e_y e_z \\
MI &= -m.xyz - m.yz\ e_x - m.zx\ e_y - m.xy\ e_z \\
&\quad + m.z\ e_x e_y + m.y\ e_z e_x + m.z\ e_x e_y + m.q\ e_x e_y e_z \\
MI^{-1} &= M/I = -MI \\
&= m.xyz + m.yz\ e_x + m.zx\ e_y + m.xy\ e_z \\
&\quad - m.z\ e_x e_y - m.y\ e_z e_x - m.z\ e_x e_y - m.q\ e_x e_y e_z
\end{aligned}$$

David Hestenes, in [6] p.56 equation (3.8) uses IA as the definition of the dual operator on A. However, in other works, such as [8] he uses AI^{-1}. In yet another work, [7], page 25, Hestenes defines the dual (using a tilde) with the reverse of the pseudoscalar as $\tilde{A} = AI^{\dagger}$, which in three space is $\tilde{A} = -AI$. Other authors, such as Alan Bromborsky, [1] use the inverse of the pseudovector in the duality operation above. Consequently, we have a sign difference for the dual between these different authors. In my opinion, the concensus is currently to use $AI^{-1}(= -AI$ in 3D) as the definition for the dual. Notationally, Hestenes typically uses tildes $\tilde{A}$ to indicate duals, while Dorst and Broborsky use a raised asterisk A^*.

Regardless of sign convention, the dual of a multivector is naively orthogonal to the original multivector, meaning the sum of the pairwise component products is zero. Likewise, double operation of the dual in three dimensional Euclidean space inverts the sign of the original multivector. In three dimensional Euclidean geometric algebra, we can implement the dual as simple swapping of components, with sign changes as needed. Notice the change of scalar/pseudoscalar, vector and bivector components.

```
GA3DE Dual(GA3DE w) // Effectively post-multiply by negative pseudoscalar.
{   GA3DE v;
    v.q =  w.xyz;
    v.x =  w.yz;
    v.y =  w.zx;
    v.z =  w.xy;
    v.xy =  -w.z;
    v.zx =  -w.y;
    v.yz =  -w.x;
    v.xyz = -w.q;
    return v;
}
```

Grassman Complement

John Browne [3], working from Lloyd Kannenberg's English translation of Hermann Grassman's *Ausdehnungslehre* presents the function of the complement, which in three dimensions is like the dual, without the sign changes. (In other dimensions such as 2D, the complement has various components complemented.) Notice the change of scalar/pseudoscalar, vector and bivector components.

```
GA3DE Comp(GA3DE w) // Grassman's Complement
{
    GA3DE v;
    v.q =   w.xyz;
    v.x =   w.yz;
    v.y =   w.zx;
    v.z =   w.xy;
    v.xy =  w.z;
    v.zx =  w.y;
    v.yz =  w.x;
    v.xyz = w.q;
    return v;
}
```

John Browne points out that the complement (not), wedge (or) and regressive product (and) have mutual relationships matching DeMorgan's law in digital logic. Eric Langyel [9] call the regressive product the antiwedge product. Both authors use an overbar to indicate the Grassman complement. David Hestenes waxes eloquently on the value of the complement and regressive product in *Universal Geometric Algebra* [8].

Reverse

The reverse operator, typically indicated by a dagger exponent $M^\dagger$, changes the sign of multivector elements according to a reversal of the order of multiplication of the basis vectors. Note that Dorst uses a tilde overbar in his work.

```
GA3DE Rev(GA3DE w) // Reverse
{
    GA3DE v;
    v.q =   w.q;
```

```
    v.x =   w.x;
    v.y =   w.y;
    v.z =   w.z;
    v.xy =  -w.xy;
    v.zx =  -w.zx;
    v.yz =  -w.yz;
    v.xyz = -w.xyz;
    return v;
}
```

We notice that the complement is the negative of the reverse of the dual, $\bar{G} = -(G * I)^\dagger$. Some algebraic identities:

$$\begin{aligned} (M^\dagger)^\dagger &= M \\ (AB)^\dagger &= B^\dagger A^\dagger \end{aligned}$$

The product of a multivector with its reverse has a scalar component similar to the norm of a vector, and a vector portion, usually ignored.

```
r = (a.q, a.x, a.y, a.z, a.xy, a.zx, a.yz, a.xyz)
r*Rev(r)
u.q   =  + a.q^2 + a.x^2 + a.y^2 + a.z^2 + a.xy^2 + a.zx^2 + a.yz^2 + a.xyz^2
u.x   =  + 2*a.q*a.x + 2*a.y*a.xy - 2*a.z*a.zx + 2*a.yz*a.xyz
u.y   =  + 2*a.q*a.y + 2*a.z*a.yz - 2*a.x*a.xy + 2*a.zx*a.xyz
u.z   =  + 2*a.q*a.z + 2*a.x*a.zx - 2*a.y*a.yz + 2*a.xyz*a.xy
u.xy  = 0
u.zx  = 0
u.yz  = 0
u.xyz = 0
```

Parity

The parity operator changes the sign of a multivector component by inverting each basis term.

```
GA3DE Par(GA3DE w) // Reverse
{
    GA3DE v;
    v.q =   w.q;
    v.x =   -w.x;
    v.y =   -w.y;
```

```
    v.z =   -w.z;
    v.xy =  w.xy;
    v.zx =  w.zx;
    v.yz =  w.yz;
    v.xyz = -w.xyz;
    return v;
}
```

Calling the parity operator twice recovers the original multivector. Dorst uses the circumflex (hat) either as an overbar or as an exponent to indicate parity reversal. Some algebraic identities:

$$\begin{aligned} (\widehat{M})^{\wedge} &= M \\ (A \wedge B)^{\wedge} &= \widehat{A} \wedge \widehat{B} \end{aligned}$$

A multivector times its parity reverse image has a zero trivector component, and has a superficial resemblance to a projection.

```
r = (a.q,  a.x,  a.y,  a.z, a.xy, a.zx, a.yz,  a.xyz)
s = (a.q, -a.x, -a.y, -a.z, a.xy, a.zx, a.yz, -a.xyz)
r*Par(r)
u.q   =  + a.q^2 - a.x^2 - a.y^2 - a.z^2 - a.xy^2 - a.zx^2 - a.yz^2 + a.xyz^2
u.x   =  + 2*a.z*a.zx - 2*a.y*a.xy
u.y   =  + 2*a.x*a.xy - 2*a.z*a.yz
u.z   =  + 2*a.y*a.yz - 2*a.x*a.zx
u.xy  =  + 2*a.q*a.xy - 2*a.z*a.xyz
u.zx  =  + 2*a.q*a.zx - 2*a.y*a.xyz
u.yz  =  + 2*a.q*a.yz - 2*a.x*a.xyz
u.xyz = 0
```

Conjugate

The Clifford conjugate in Dorst, is $\hat{M}^{\dagger}$.

```
GA3DE ClifConj(GA3DE w) // Clifford Conjugate
{
    GA3DE v; // parity first, dagger second
    v.q =   w.q;
    v.x =   -w.x;
    v.y =   -w.y;
    v.z =   -w.z;
    v.xy =  -w.xy;
```

```
    v.zx =  -w.zx;
    v.yz =  -w.yz;
    v.xyz = w.xyz;
    return v;
}
```

Double conjugation recovers the original multivector. A multivector times the Clifford conjugate results in a complex number for the norm, treating the multivector component as the complex component.

```
r = (a.q,  a.x,  a.y,  a.z,  a.xy,  a.zx,  a.yz, a.xyz)
s = (a.q, -a.x, -a.y, -a.z, -a.xy, -a.zx, -a.yz, a.xyz)
r*ClifConj(r) (L)
u.q   =  + a.q^2 - a.x^2 - a.y^2 - a.z^2 + a.xy^2 + a.zx^2 + a.yz^2 - a.xyz^2
u.x   = 0
u.y   = 0
u.z   = 0
u.xy  = 0
u.zx  = 0
u.yz  = 0
u.xyz =  + 2*a.q*a.xyz - 2*a.x*a.yz - 2*a.y*a.zx - 2*a.z*a.xy
```

Regressive Product

Per John Browne, the Grassman regressive product is

$$A \vee B = \overline{\overline{A} \wedge \overline{B}}$$

```
GA3DE RegressPerBrowne(const GA3DE &a, const GA3DE &b) // conj( conj(a) ^ conj(b)  )
{
    GA3DE c;

    c.q = (a.q*b.xyz + a.x*b.yz + a.y*b.zx + a.z*b.xy
           + a.xy*b.z + a.zx*b.y + a.yz*b.x + a.xyz*b.q);
    c.x = (a.x*b.xyz - a.xy*b.zx + a.zx*b.xy + a.xyz*b.x);
    c.y = (a.y*b.xyz - a.yz*b.xy + a.xy*b.yz + a.xyz*b.y);
    c.z = (a.z*b.xyz - a.zx*b.yz + a.yz*b.zx + a.xyz*b.z);
    c.yz = (a.yz*b.xyz + a.xyz*b.yz);
    c.zx = (a.zx*b.xyz + a.xyz*b.zx);
    c.xy = (a.xy*b.xyz + a.xyz*b.xy);
    c.xyz = (a.xyz*b.xyz);
    return c;

}
```

By contrast, per Hestenes in [8], equation 14, the Grassman regressive product is

$$(A \vee B)^{\check{}} \;=\; \tilde{A} \wedge \tilde{B}$$

Specializing to three dimensional Euclidean space,

$$\begin{aligned} \tilde{A} &= AI^{\dagger} = -AI \\ (A \vee B)^{\check{}} &= \tilde{A} \wedge \tilde{B} \\ ((A \vee B)^{\check{}})^{\check{}} &= (\tilde{A} \wedge \tilde{B})^{\check{}} \\ -(A \vee B) &= (\tilde{A} \wedge \tilde{B})^{\check{}} \\ A \vee B &= -(\tilde{A} \wedge \tilde{B})^{\check{}} \end{aligned}$$

In code, $A \vee B$ = -dual((dual(A) $\wedge$ dual(B)))

```
GA3DE RegressPerHestenes(const GA3DE &a, const GA3DE &b)
// -dual( dual(a) ^ dual(b)  )
{
    GA3DE c;

    c.q = (a.q*b.xyz + a.x*b.yz + a.y*b.zx + a.z*b.xy
           + a.xy*b.z + a.zx*b.y + a.yz*b.x + a.xyz*b.q);
    c.x = (a.x*b.xyz - a.zx*b.xy + a.xy*b.zx + a.xyz*b.x);
    c.y = (a.y*b.xyz - a.xy*b.yz + a.yz*b.xy + a.xyz*b.y);
    c.z = (a.z*b.xyz - a.yz*b.zx + a.zx*b.yz + a.xyz*b.z);
    c.yz = (a.yz*b.xyz + a.xyz*b.yz);
    c.zx = (a.zx*b.xyz + a.xyz*b.zx);
    c.xy = (a.xy*b.xyz + a.xyz*b.xy);
    c.xyz = (a.xyz*b.xyz);
    return c;

}
```

Comparing these two forms, we see sign differences in the bivector products in the vector terms. The Hestenes formula is equal the Browne formula with reversed order of inputs. RegressPerHestenes(A, B) = RegressPerBrowne(B, A).

Units and the Regressive Product

With the wedge product, we saw that the wedge product preserved the units of measure when the scalar was without units, the vectors in linear measure, the bivectors in areal measure, and the trivector in volume measure. With the regressive product, we see similar scaling, but with an overall factor of L^3 applied.

```
r = (a.q, L*a.x, L*a.y, L*a.z, L^2*a.xy, L^2*a.zx, L^2*a.yz, L^3*a.xyz)
s = (b.q, L*b.x, L*b.y, L*b.z, L^2*b.xy, L^2*b.zx, L^2*b.yz, L^3*b.xyz)
Regressive Product with Units (L)
u.q   = L^3*( + a.q*b.xyz + a.x*b.yz + a.y*b.zx + a.z*b.xy
              + a.xy*b.z + a.zx*b.y + a.yz*b.x + a.xyz*b.q )
u.x   = L^4*( + a.x*b.xyz - a.xy*b.zx + a.zx*b.xy + a.xyz*b.x )
u.y   = L^4*( + a.y*b.xyz - a.yz*b.xy + a.xy*b.yz + a.xyz*b.y )
u.z   = L^4*( + b.xyz*a.z - a.zx*b.yz + a.yz*b.zx + a.xyz*b.z )
u.xy  = L^5*( + a.xy*b.xyz + a.xyz*b.xy )
u.zx  = L^5*( + a.zx*b.xyz + a.xyz*b.zx )
u.yz  = L^5*( + a.yz*b.xyz + a.xyz*b.yz )
u.xyz = L^6*( + a.xyz*b.xyz)
```

Dot Products and Contractions

In my opinion, dot product, left and right contractions are poorly defined, due to inclusion of scalars in the wedge product. In my current mindframe, I define dot, left contraction and right contraction using Bromborsky Sympy GA as a guide, but try to avoid these products. Instead, I try to use the symmetric product, or the difference of GA product minus wedge.

Dot

This is the dot product from Bromborsky. Notice no scalar terms participate in the actual product. I believe these are defective definitions, due to the lumping of scalar terms in the wedge product.

```
GA3DE dot(const GA3DE &a, const GA3DE &b)
// Based Upon Bromborsky (A|B)
{
GA3DE c;
```

```
c.q =  + a.x*b.x  + a.y*b.y  + a.z*b.z
       - a.xy*b.xy  - a.zx*b.zx  - a.yz*b.yz  - a.xyz*b.xyz ;
c.x =  - a.y*b.xy + a.z*b.zx + a.xy*b.y - a.zx*b.z   - a.yz*b.xyz - a.xyz*b.yz ;
c.y =  + a.x*b.xy - a.z*b.yz - a.xy*b.x - a.zx*b.xyz + a.yz*b.z   - a.xyz*b.zx ;
c.z =  - a.x*b.zx + a.y*b.yz - a.xy*b.xyz + a.zx*b.x   - a.yz*b.y   - a.xyz*b.xy ;
c.xy = + a.z*b.xyz + a.xyz*b.z  ;
c.zx = + a.y*b.xyz + a.xyz*b.y  ;
c.yz = + a.x*b.xyz + a.xyz*b.x ;
c.xyz = 0;

return c;
}
```

Left Contraction $(a \rfloor b)$

The left contractions is supposed to return "the part of b most unlike a" (Dorst, p591). The left contraction below is from Bromborsky. This passes the test of Dorst, p78, Eqn 3.20 $(A \wedge B) \rfloor C = A \rfloor (B \rfloor C)$.

```
GA3DE LeftContract(const GA3DE &a, const GA3DE &b)
// Based Upon Bromborsky (A<B)
{
    GA3DE c;

    c.q =  + a.q*b.q + a.x*b.x + a.y*b.y  + a.z*b.z
           - a.xy*b.xy - a.zx*b.zx - a.yz*b.yz - a.xyz*b.xyz ;
    c.x =  + a.q*b.x - a.y*b.xy + a.z*b.zx - a.yz*b.xyz ;
    c.y =  + a.q*b.y + a.x*b.xy - a.z*b.yz - a.zx*b.xyz ;
    c.z =  + a.q*b.z - a.x*b.zx + a.y*b.yz - a.xy*b.xyz ;
    c.xy =  + a.q*b.xy + a.z*b.xyz ;
    c.zx =  + a.q*b.zx + a.y*b.xyz ;
    c.yz =  + a.q*b.yz + a.x*b.xyz ;
    c.xyz = + a.q*b.xyz ;

    return c;
}
```

Right Contraction $(a \lfloor b)$

```
GA3DE RightContract(const GA3DE &a, const GA3DE &b)
// Based Upon Bromborsky (A>B)
{
    GA3DE c;
```

```
    c.q =   + a.q*b.q + a.x*b.x  + a.y*b.y + a.z*b.z
            - a.xy*b.xy - a.zx*b.zx - a.yz*b.yz - a.xyz*b.xyz ;
    c.x =   + a.x*b.q + a.xy*b.y - a.zx*b.z - a.xyz*b.yz ;
    c.y =   + a.y*b.q - a.xy*b.x + a.yz*b.z  - a.xyz*b.zx ;
    c.z =   + a.z*b.q + a.zx*b.x  - a.yz*b.y  - a.xyz*b.xy ;
    c.xy =  + a.xy*b.q + a.xyz*b.z ;
    c.zx =  + a.zx*b.q + a.xyz*b.y ;
    c.yz =  + a.yz*b.q + a.xyz*b.x ;
    c.xyz = + a.xyz*b.q ;

    return c;
}
```

Meet

According to David Hestenes [8] page 5, Meet$(A, B) = A \vee B$. This appears to conflict with Dorst, although Dorst mentions the above interpretation (due to Rota) on page 136. I don't yet have this figured out.

Join

According to David Hestenes [8] page 5, Join$(A, B) = A \wedge B$. This appears to conflict with Dorst. I don't yet have this figured out.

References

[1] Alan Bromborsky, Geometric Algebra Module for Sympy online

[2] Alan Bromborsky, An Introduction to Geometric Algebra and Calculus online

[3] John Browne, *Grassman Algebra* Barnard Publishing, ISBN 978-1479197637

[4] Chris Doran and Anthony Lasenby, *Geometric Algebra for Physicists* Cambridge University Press, ISBN 978-0-521-71595-9

[5] Leo Dorst, Daniel Fontune and Stephen Mann, *Geometric Algebra for Computer Science* Morgan Kaufmann Publishers, ISBN 978-0-12-374942-0

[6] David Hestenes, *New Foundations for Classical Mechanics* Kluwer Academic Publishers, ISBN 0-7923-5514-8

[7] David Hestenes and Garret Sobczyk, *Clifford Algebra to Geometric Calculus* D. Reidal Publishing Company, ISBN 978-90-277-2581-5

[8] David Hestenes, *Universal Geometric Algebra*, Quarterly Journal of Pure and Applied Mathematics, Volume 62 (1988)

[9] Eric Langyel, Game Developers Conference 2012 Slideset
`http://www.terathon.com/lengyel/`

[10] Anthony Lasenby and Chris Doran, *Lectures and Handouts 1999*
`http://www.mrao.cam.ac.uk/clifford/ptIIIcourse/`