Mixed Units in Geometric and Wedge Swedge Products

Kurt Nalty

July 24, 2015

Abstract

Geometric algebra defines composite structures of scalars, vectors, bivectors and higher order terms. When two multivectors are multiplied, if the internal units are different, such as meters for the vectors, meters squared for the bivectors, and and meters cubed for the trivector, then the resulting ultivector product has mixed units in some of the product terms. (Example, the scalar portion now additionally has square meters, m^4 and m^6 terms.)

By contrast, for standard dimensionals, such as meters for vector, square meters for areas, and cubic meters for trivectors, the wedge product maintains the same spatial units as the two incoming factors.

Executive summary:

- Multivector product mixes units. Plan to use dimensionless quantities, or uniform dimensioned quantities when using the general multivector product.
- Wedge product preserves spatial dimensionality in the product.

3D Euclidean Geometric Product

The 3D Euclidean geometric algebra has one scalar, three orthogonal vector basii, three orthogonal bivector basii, and one trivector. Ordinarily multivectors are treated as having homogenous dimensional units, ideally being pure numbers in a fashion seen in unit vectors. In the multiplication process, all terms multiply, and the question arises as to the nature of this product when units differ between the vectors, bivectors and trivector.

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

```
G3E3 operator*(const G3E3 &a, const G3E3 &b) {
    G3E3 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;
}
```

To see the effect of mixed units, I used the ginac symbolic package to form products of two multivectors which explicitly carried a symbolic dimension L. The vectors elements were linear in L. The bivectors quadratic in L, and the trivector cubic in L.

```
    r = G3E3(a_q,L*a_x,L*a_y,L*a_z,L*L*a_xy,L*L*a_zx,L*L*a_yz,L*L*L*a_xyz);
    s = G3E3(b_q,L*b_x,L*b_y,L*b_z,L*L*b_xy,L*L*b_zx,L*L*b_yz,L*L*L*b_xyz);

    u = r*s;
    u.q = expand(u.q);
    u.x = expand(u.x);
    u.y = expand(u.y);
    u.z = expand(u.z);
    u.yz = expand(u.yz);
    u.zx = expand(u.zx);
    u.xy = expand(u.xy);
    u.xyz = expand(u.xyz);
```

```
    cout << "u = " << r <<" * " << s << " \n";
    cout << "Multivector\n";
    cout << "u.q   = " << u.q   << " \n";
    cout << "u.x   = " << u.x   << " \n";
    cout << "u.y   = " << u.y   << " \n";
    cout << "u.z   = " << u.z   << " \n";
    cout << "u.xy  = " << u.xy  << " \n";
    cout << "u.zx  = " << u.zx  << " \n";
    cout << "u.yz  = " << u.yz  << " \n";
    cout << "u.xyz = " << u.xyz << " \n\n";
```

The resulting product has significant unit mixing in most components.

```
u.q   =  + a.q*b.q + L^2( a.y*b.y + a.z*b.z + a.x*b.x)
         - L^4( a.yz*b.yz + a.zx*b.zx + a.xy*b.xy) - L^6(a.xyz*b.xyz)

u.x   = + L(a.x*b.q + a.q*b.x)
        + L^3((a.z*b.zx - a.zx*b.z) + (a.xy*b.y - a.y*b.xy))
        - L^5(a.yz*b.xyz + a.xyz*b.yz)

u.y   = + L(a.y*b.q + a.q*b.y)
        + L^3((a.x*b.xy - a.z*b.yz) + (a.yz*b.z - a.xy*b.x))
        - L^5(a.zx*b.xyz + a.xyz*b.zx)

u.z   = + L(a.z*b.q + a.q*b.z)
        + L^3((a.y*b.yz - a.x*b.zx) + (a.zx*b.x - a.yz*b.y))
        - L^5(a.xy*b.xyz + a.xyz*b.xy)

u.xy  = + L^2(a.xy*b.q + a.q*b.xy + a.x*b.y - a.y*b.x)
        + L^4(a.z*b.xyz + a.xyz*b.z + a.zx*b.yz - a.yz*b.zx)

u.zx  = + L^2(a.zx*b.q + a.q*b.zx + a.z*b.x - a.x*b.z)
        + L^4(a.y*b.xyz + a.xyz*b.y + a.yz*b.xy - a.xy*b.yz)

u.yz  = + L^2(a.yz*b.q + a.q*b.yz + a.y*b.z - a.z*b.y)
        + L^4(a.x*b.xyz + a.xyz*b.x + a.xy*b.zx - a.zx*b.xy)

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)
```

My conclusion, and take-away from this is that general multivector multiplication does not form a group under multiplication, except when the units

are homogenous across all terms, or when the product is explicity idempotent, negative potent, or nilpotent.

This does not mean that multivector multiplication is invalid. We can definitely form products, and add/subtract these products, but we need to pay attention to our unit power structure.

Wedge Product

By contrast to the general multivector product, the wedge product is nicely behaved, conserving the power structure of the factors in the product. The wedge product is the antisymmetric product for the basis elements, with the requirement that a squared basis is zero. Geometrically, the wedge product extends the two term to a higher order geometric entity. Line times line provides area, line times area provides volume, and so forth. When we restrict our space, say to three dimensions, the product of more than three terms will necessarily have a redundancy leading to a zero term. As we look at the expression for the general multivector product above, the terms including L^4, L^5 and L^6 are guaranteed not found in a wedge product.

Our wedge product (includes commutating scalar terms) is defined on the component level as

```
G3E3 Wedge(const G3E3 &a, const G3E3 &b) // wedge product
{
    G3E3 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;
}
```

We form the same input dimensional unit tracing terms L, but now use the wedge product.

```
r = G3E3(a_q,L*a_x,L*a_y,L*a_z,L*L*a_xy,L*L*a_zx,L*L*a_yz,L*L*L*a_xyz);
s = G3E3(b_q,L*b_x,L*b_y,L*b_z,L*L*b_xy,L*L*b_zx,L*L*b_yz,L*L*L*b_xyz);

u = Wedge(r,s);
u.q = expand(u.q);
u.x = expand(u.x);
u.y = expand(u.y);
u.z = expand(u.z);
u.yz = expand(u.yz);
u.zx = expand(u.zx);
u.xy = expand(u.xy);
u.xyz = expand(u.xyz);

cout << "u = " << r <<" * " << s << " \n";
cout << "Wedge\n";
cout << "u.q   = " << u.q   << " \n";
cout << "u.x   = " << u.x   << " \n";
cout << "u.y   = " << u.y   << " \n";
cout << "u.z   = " << u.z   << " \n";
cout << "u.xy  = " << u.xy  << " \n";
cout << "u.zx  = " << u.zx  << " \n";
cout << "u.yz  = " << u.yz  << " \n";
cout << "u.xyz = " << u.xyz << " \n\n";
```

The resulting product preserves the dimensional grading of the input.

```
Wedge
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((b.xy*a.q + a.xy*b.q) + (a.x*b.y - a.y*b.x))
u.zx  = L^2((a.q*b.zx + a.zx*b.q) + (a.z*b.x - a.x*b.z))
u.yz  = L^2((a.q*b.yz + a.yz*b.q) + (a.y*b.z - a.z*b.y))
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)
```

The wedge product preserves the dimensional units of the factors.

Source Code

Source code is at `http://www.kurtnalty.com/Mixed_Units.ginac.cp`