The AntiWedge or Regressive Product

Kurt Nalty

January 31, 2017

Abstract

Hermann Grassmann's wedge product has been widely adopted in physics and mathematics, but the related regressive product is rarely presented. Eric Lengyel, in *Foundations of Game Engine Development*, Volume 1, interprets the regressive product as a complement to the wedge product, and provides a geometrical view of the regressive product as a measure of intersection of the multiplied elements.

Two Different Complements Exist

While authors such as Browne, Doran, Dorst, Fountijne, Hestenes, Lasenby and Mann present the regressive product as part of the join and meet operations, their presentations vary and lack the simplicity of Eric Lengyel's presentation.

Eric's first important point is to introduce the right and left complement operations for multivector elements. Eric's notation uses a horizontal overbar for the right complement, and a horizontal underbar for the left complement. In odd numbered dimensions, such as three dimensional Euclidean space, the right and left complement coincide, while in even dimensional spaces, such as Minkowski spacetime, the two complements differ. The rule for the right complement is to choose the wedge factor, which postmultiplying the basis element, results in the pseudoscalar for the space at hand. Using three dimensional Euclidean space as an example, we have the following table for the right complement.

Element	Right Complement	Element $\wedge$ (Right Complement)
1	$\overline{1} = e_{xyz}$	$1 \wedge e_{xyz} = e_{xyz}$
e_x	$\overline{e_x} = e_{yz}$	$e_x \wedge e_{yz} = e_{xyz}$
e_y	$\overline{e_y} = e_{zx}$	$e_y \wedge e_{zx} = e_{xyz}$
e_z	$\overline{e_z} = e_{xy}$	$e_z \wedge e_{xy} = e_{xyz}$
e_{yz}	$\overline{e_{yz}} = e_x$	$e_{yz} \wedge e_x = e_{xyz}$
e_{zx}	$\overline{e_{zx}} = e_y$	$e_{zx} \wedge e_y = e_{xyz}$
e_{xy}	$\overline{e_{xy}} = e_z$	$e_{xy} \wedge e_z = e_{xyz}$
e_{xyz}	$\overline{e_{xyz}} = 1$	$e_{xyz} \wedge 1 = e_{xyz}$

Table 1: Right Hand Complement in 3D Euclidean Space

Element	Left Complement	(Left Complement) $\wedge$ Element
1	$\underline{1} = e_{xyz}$	$e_{xyz} \wedge 1 = e_{xyz}$
e_x	$\underline{e_x} = e_{yz}$	$e_{yz} \wedge e_x = e_{xyz}$
e_y	$\underline{e_y} = e_{zx}$	$e_{zx} \wedge e_y = e_{xyz}$
e_z	$\underline{e_z} = e_{xy}$	$e_{xy} \wedge e_z = e_{xyz}$
e_{yz}	$\underline{e_{yz}} = e_x$	$e_x \wedge e_{yz} = e_{xyz}$
e_{zx}	$\underline{e_{zx}} = e_y$	$e_y \wedge e_{zx} = e_{xyz}$
e_{xy}	$\underline{e_{xy}} = e_z$	$e_z \wedge e_{xy} = e_{xyz}$
e_{xyz}	$\underline{e_{xyz}} = 1$	$1 \wedge e_{xyz} = e_{xyz}$

Table 2: Left Hand Complement in 3D Euclidean Space

In a similar fashion, for the left complement, choose the factor which pre-multiplying via wedge product on the left, produces the pseudoscalar element. For three dimensions, we have the same terms as the right complement.

In Table 3, we present the left and right complements for four dimensional Euclidean space, as well as the double complements. (This is Table 4.4 of Eric's *Foundations of Game Engine Development* book.)

Element	Right Complement	Left Complement	Double Complement
1	$\overline{1} = e_{xyzt}$	$\underline{1} = e_{xyzt}$	1
e_x	$\overline{e_x} = e_{yzt}$	$\underline{e_x} = -e_{yzt}$	$-e_x$
e_y	$\overline{e_y} = e_{zxt}$	$\underline{e_y} = -e_{zxt}$	$-e_y$
e_z	$\overline{e_z} = e_{xyt}$	$\underline{e_z} = -e_{xyt}$	$-e_z$
e_t	$\overline{e_z} = e_{xzy}$	$\underline{e_z} = -e_{xzy}$	$-e_t$
e_{tx}	$\overline{e_{tx}} = -e_{yz}$	$\underline{e_{tx}} = -e_{yz}$	e_{tx}
e_{ty}	$\overline{e_{ty}} = -e_{zx}$	$\underline{e_{ty}} = -e_{zx}$	e_{ty}
e_{tz}	$\overline{e_{tz}} = -e_{xy}$	$\underline{e_{tz}} = -e_{xy}$	e_{tz}
e_{yz}	$\overline{e_{yz}} = -e_{tx}$	$\underline{e_{yz}} = -e_{tx}$	e_{yz}
e_{zx}	$\overline{e_{zx}} = -e_{ty}$	$\underline{e_{zx}} = -e_{ty}$	e_{zx}
e_{xy}	$\overline{e_{xy}} = -e_{tz}$	$\underline{e_{xy}} = -e_{tz}$	e_{xy}
e_{yzt}	$\overline{e_{yzt}} = -e_x$	$\underline{e_{yzt}} = e_x$	$-e_{yzt}$
e_{zxt}	$\overline{e_{zxt}} = -e_y$	$\underline{e_{zxt}} = e_y$	$-e_{zxt}$
e_{xyt}	$\overline{e_{xyt}} = -e_z$	$\underline{e_{xyt}} = e_z$	$-e_{xyt}$
e_{xzy}	$\overline{e_{xzy}} = -e_t$	$\underline{e_{xzy}} = e_t$	$-e_{xzy}$
e_{xyzt}	$\overline{e_{xyzt}} = 1$	$\underline{e_{xyzt}} = 1$	e_{xyzt}

Table 3: Complements in 4D Euclidean Space

In odd numbered dimensions, double application of the complement recovers the original multivector. In all dimensions, application of left complement followed by application of right complement, or vice versus, recovers the original multivector.

$$\overline{\underline{A}} = A$$

Having the complements at hand, we can define the antiwedge product via DeMorgan style duality relationships.

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

We can also define the wedge product in terms of the antiwedge product.

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

C++ Code Implementation

Using C++, easily downgraded to C, we can implement the antiwedge product. Our three dimensional Euclidean data strucure is

```
struct G3E3{
ex q,x,y,z,xy,zx,yz,xyz;
G3E3() {q = 0; x = 0; y = 0; z = 0; xy = 0; zx = 0; yz = 0; xyz = 0;}
};
```

Our wedge product is

```
G3E3 operator^(const G3E3 &a, const G3E3 &b) // scalar and 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;
}
```

Our right and left complements (which coincide in 3D) are

```
G3E3 RComp(const G3E3 &a) // Right Complement per Eric Lanyel
{
G3E3 b;

b.q = a.xyz;

b.x = a.yz;
b.y = a.zx;
b.z = a.xy;

b.xy = a.z;
b.zx = a.y;
b.yz = a.x;

b.xyz = a.q;

return b;
}
```

Our antiwedge product, via complements, is

```
G3E3 AntiWedgeViaComp(const G3E3 &a, const G3E3 &b)
{
G3E3 c;
c = RComp(Wedge(RComp(a),RComp(b)));
return c;
}
```

Our antiwedge product, checked against the duality relations above, are

```
G3E3 AntiWedge(const G3E3 &a, const G3E3 &b)
// Uses Langyels Complement to obtain this formula
{
G3E3 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.xy*b.yz - a.yz*b.xy + a.xyz*b.y ;
c.z   = + a.z*b.xyz - a.zx*b.yz + a.yz*b.zx + a.xyz*b.z ;
c.xy  = + a.xy*b.xyz + a.xyz*b.xy ;
c.zx  = + a.zx*b.xyz + a.xyz*b.zx ;
c.yz  = + a.yz*b.xyz + a.xyz*b.yz ;
c.xyz = + a.xyz*b.xyz;

return c;
}
```

In multiplication table format, the antiwedge product is

```
 Multiplication table format using AntiWedge, prefactor on left
                            PostFactor
    |  q      x      y      z      xy     zx     yz     xyz
-------------------------------------------------------------
  q |  0      0      0      0      0      0      0      q
  x |  0      0      0      0      0      0      q      x
  y |  0      0      0      0      0      q      0      y
  z |  0      0      0      0      q      0      0      z
 xy |  0      0      0      q      0     -x      y      xy
 zx |  0      0      q      0      x      0     -z      zx
 yz |  0      q      0      0     -y      z      0      yz
xyz |  q      x      y      z      xy     zx     yz     xyz
```

The Antiwedge as Intersection

From the multiplication table, we see that in three dimensions, the trivector is the identity element. This is consistent with the intersection of anything with all of space is that first item.

Systematically investigating our different grade products, we find the following.

Antiwedging with a pure scalar is usually zero, the exception being the trivector identity element.

```
AntiWedge(q,q) = (0, 0, 0, 0, 0, 0, 0, 0)
```

```
AntiWedge(q,V) = (0, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(q,BV) = (0, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(q,MV) = (a.q*b.xyz, 0, 0, 0, 0, 0, 0, 0)
```

```
AntiWedge(V,q) = (0, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(V,V) = (0, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(V,BV) = (a.x*b.yz+b.xy*a.z+b.zx*a.y, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(V,MV) = (a.x*b.yz+b.xy*a.z+b.zx*a.y, a.x*b.xyz, a.y*b.xyz,
                   a.z*b.xyz, 0, 0, 0, 0)
```

```
AntiWedge(BV,q) = (0, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(BV,V) = (a.xy*b.z+b.x*a.yz+b.y*a.zx, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(BV,BV) = (0, -b.zx*a.xy+b.xy*a.zx, -b.xy*a.yz+a.xy*b.yz,
                        -b.yz*a.zx+a.yz*b.zx, 0, 0, 0, 0)
AntiWedge(BV,MV) = (+a.xy*b.z+b.x*a.yz+b.y*a.zx, -b.zx*a.xy+b.xy*a.zx,
                     -b.xy*a.yz+a.xy*b.yz, -b.yz*a.zx+a.yz*b.zx,
                      a.xy*b.xyz, a.zx*b.xyz, a.yz*b.xyz, 0)
```

```
AntiWedge(TV,q) = (a.xyz*b.q, 0, 0, 0, 0, 0, 0, 0)
AntiWedge(TV,V) = (0, b.x*a.xyz, b.y*a.xyz, a.xyz*b.z, 0, 0, 0, 0)
AntiWedge(TV,BV) = (0, 0, 0, 0, b.xy*a.xyz, b.zx*a.xyz, a.xyz*b.yz, 0)
AntiWedge(TV,MV) = (a.xyz*b.q, b.x*a.xyz, b.y*a.xyz, a.xyz*b.z,
                    b.xy*a.xyz, b.zx*a.xyz, a.xyz*b.yz, a.xyz*b.xyz)
```

References

[1] John Browne, *Grassmann Algebra*, Barnard Publishing, 9-781479-197637

[2] Hermann Grassmann, *A New Branch of Mathematics*, Translated by Lloyd C. Kannenberg, Open Court Publishing, ISBN 0-8126-9276-4

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

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

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

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

[7] Eric Lengyel, *Foundations of Game Engine Development*, Terathon Software LLC, ISBN 9-780985-811747