A summary of Projective Geometric Algebra

Patrolin

October 3, 2022

Abstract

Information on Geometric Algebra is scattered and scarce. This is an explanation of how to calculate PGA, but not necessarily why it works - for that you can try sudgylacmoe on youtube and Eric Lengyel's cheatsheet for point-based PGA (though he goes mad with power). We would also point you to bivector.net, but their dual calculations are whack, the ganja.js ones are presumably correct, since their demos seem to work.

I. Geometric numbers

$$1 + x^2 = 0$$
$$x = ?$$

x is not a real number; but if it's not real, why should the other numbers be real?

$$(e_1)^2 + (e_2)^2 = (e_0)^2$$
$$(e_1)^2 = 1;\ (e_2)^2 = -1;\ (e_0)^2 = 0$$

In fact, we can define as many of these as we want with a geometric algebra $\mathbb{G}_{a,b,c}$ having a numbers that square to 1, b that square to -1 and c that square to 0 (degenerate basis vectors):

$(a + be_1) \in \mathbb{G}_{1,0,0}$ `// hyperbolic numbers`
$(a + be_2) \in \mathbb{G}_{0,1,0}$ `// complex numbers`
$(a + be_0) \in \mathbb{G}_{0,0,1}$ `// dual numbers`

We can multiply these numbers together using the geometric product:

$$(e_i)^2 = \{1, -1, 0\}$$
$$e_i e_j = -e_j e_i$$

The result is another number in the same algebra $\mathbb{G}_{a,b,c}$

This product is neither commutative nor anticommutative, but it is distributive and associative:

$$AB \neq BA$$
$$AB \neq -BA$$
$$A(B+C) = AB + AC$$
$$(AB)C = A(BC)$$
$$aB = Ba;\ a \in \mathbb{R}$$

Thus the product of two complex numbers:

$$\begin{aligned}
&(A_1 + A_2 e_2)(B_1 + B_2 e_2) \\
&= A_1B_1 + A_1B_2e_2 + A_2e_2B_1 + A_2e_2B_2e_2 \\
&= A_1B_1 + A_1B_2e_2 + A_2B_1e_2 + A_2B_2 \\
&= (A_1B_1 + A_2B_2) + (A_1B_2 + A_2B_1)e_2
\end{aligned}$$

II. Rotations

A multivector with n basis vectors consists of 2^n blades:

```
- scalar = 0-vector = 1
- vector = 1-vector
- bivector = 2-vector
- trivector = 3-vector
- ...
- pseudovector = (n-1)-vector
- pseudoscalar = n-vector = 𝟙
```

Where a k-vector has $\binom{n}{k}$ blades, for example:

$$\begin{aligned}A = A_1 & \\ &+ A_2 e_0 + A_3 e_1 + A_4 e_2 \\ &+ A_5 e_{01} + A_6 e_{02} + A_7 e_{12} \\ &+ A_8 e_{012}\end{aligned}$$

We can abbreviate blades like $e_1 e_2$ as e_{12}.

We can use the exponential function to find a rotation e^A:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

$$\begin{aligned}e^{ae_2} &= 1 + ae_2 - \frac{a^2}{2} - \frac{a^3}{3} e_2 + ... \\ &= (1 - \frac{a^2}{2} + ...) + (a - \frac{a^3}{3} + ...) e_2 \\ &= cos(a) + sin(a) e_2\end{aligned}$$

Similarly you can find

$$e^{e_i} = \begin{cases} cosh(a) + sinh(a) e_i & ((e_i)^2 = 1) \\ cos(a) + sin(a) e_i & ((e_i)^2 = -1) \\ 1 + e_i & ((e_i)^2 = 0) \end{cases}$$

This gives us hyperbolic rotations, rotations and translations (rotations through infinity) respectively.

For the i-th blade X_i in a multivector:

$$e^A = \prod_i e^{A_i X_i}$$

III. Unary operators

We can define some operations, like reversing the order of basis vectors in the blade, that amount to flipping some signs:

$$\begin{aligned}\tilde{X}_i &= (-1)^{\lfloor k/2 \rfloor} X_i \texttt{ // reverse} \\ X_i^{\dagger} &= (-1)^{\lfloor k \rfloor} X_i \texttt{ // involute} \\ \bar{X}_i &= (-1)^{\lfloor k + k/2 \rfloor} X_i \texttt{ // conjugate}\end{aligned}$$

$$X_i \in \text{k-vector}$$

$$f(A) = \sum_i f(X_i)$$

Poincaré duality states that maps between k-vectors and (n-k)-vectors exist.

$$\begin{aligned}X_i\, dual(X_i) &= \pm \mathbb{1} \\ dual(X_i) &= \pm X_{2^n - i + 1} \\ dual(A) &= \sum_i dual(X_i)\end{aligned}$$

For example:

$$\begin{aligned}\underline{X_i} X_i &= \mathbb{1} \texttt{ // left complement} \\ X_i \overline{X_i} &= \mathbb{1} \texttt{ // right complement} \\ X_i X_i^{\star} &= sign(X_i^{ND} \widetilde{X_i^{ND}}) \mathbb{1} \texttt{ // hodge dual}\end{aligned}$$

Where X_i^{ND} is X_i without degenerate basis vectors, e.g.

$$X_i = e_{012};\ X_i^{ND} = e_{12}$$

For $\mathbb{G}_{a,b,0}$: $X_i^{\star} = X_i \tilde{\mathbb{1}}$

For $\mathbb{G}_{a,0,c}$: $X_i^{\star} = \overline{X_i}$

And applying a dual twice changes the signs, so we also want their inverses:

$$\begin{aligned}(X_i^{\star})^{\star^{-1}} &= X_i \\ \underline{\overline{X_i}} &= X_i\end{aligned}$$

In 2D and 3D PGA, we can simplify implementation by swapping two basis vectors in some blades such that $\overline{X_i}$ does not flip signs, e.g.

$$\begin{aligned}A &= A_1 + A_2 e_0 + A_3 e_1 + A_4 e_2 \\ &\quad + A_5 e_{01} + A_6 e_{20} + A_7 e_{12} + A_8 e_{012} \\ \underline{A} = \overline{A} &= A_8 + A_7 e_0 + A_6 e_1 + A_5 e_2 \\ &\quad + A_4 e_{01} + A_3 e_{20} + A_2 e_{12} + A_1 e_{012}\end{aligned}$$

IV. Shapes and sizes

Let $\mathbb{G}_{d,0,1}$ be a d-dimensional PGA:

$$(e_0)^2 = 0;\ (e_i)^2 = 1$$

Now we have a choice to make:

1. `Point-based PGA`
   - `- vectors are points`
   - `- (n-1)-vectors are hyperplanes`
2. `Plane-based PGA`
   - `- vectors are hyperplanes`

```
- (n-1)-vectors are points
```

Both of these are equally valid and many operations make use of computing in the dual algebra via $\underline{\overline{A} \text{ op } \overline{B}}$.

$$point = e_0 + xe_1 + ye_2 + ... + we_d$$
$$hyperplane = \overline{e_0 + xe_1 + ye_2 + ... + we_d}$$

$$\boxed{\begin{aligned} hyperplane &= e_0 + xe_1 + ye_2 + ... + we_d \\ point &= \overline{e_0 + xe_1 + ye_2 + ... + we_d} \end{aligned}}$$

We will denote Plane-based operations inside boxes whenever they differ.

Let $\langle A \rangle_k$ be the grade selection operator:

$$\langle X_i \rangle_k = \begin{cases} X_i & (X_i \in \text{k-vector}) \\ 0 & (X_i \notin \text{k-vector}) \end{cases}$$
$$\langle A \rangle_k = \sum_i \langle X_i \rangle_k$$

Then we can define the wedge $\wedge$ and anti-wedge $\vee$ products:

$$A \wedge B = \sum_{j,k} \left\langle \langle A \rangle_j \langle B \rangle_k \right\rangle_{j+k} = \underline{\overline{A} \vee \overline{B}}$$
$$A \vee B = \sum_{j,k} \left\langle \langle A \rangle_j \langle B \rangle_k \right\rangle_{n-((n-j)+(n-k))}$$
$$= \underline{\overline{A} \wedge \overline{B}}$$

These operators retain distributivity and associativity.

We can then join points into lines and lines into planes:

$$line = point_1 \, {}_{join} \, point_2$$
$$plane = point_1 \, {}_{join} \, point_2 \, {}_{join} \, point_3$$
$$A \, {}_{join} \, B = A \wedge B$$

$$\boxed{A \, {}_{join} \, B = A \vee B}$$

In fact this works with any two geometric objects, operators that don't have this property aren't really worth your time.

And we can meet two objects:

$$point = line_1 \, {}_{meet} \, line_2$$
$$point = plane \, {}_{meet} \, line$$
$$line = plane_1 \, {}_{meet} \, plane_2$$
$$A \, {}_{meet} \, B = A \vee B$$

$$\boxed{A \, {}_{meet} \, B = A \wedge B}$$

If you meet two parallel lines, you get a point at infinity = an infinite point:

$$line_1 \, {}_{meet} \, line_1 = xe_1 + ye_2 + ... + we_d$$

$$\boxed{line_1 \, {}_{meet} \, line_1 = \overline{xe_1 + ye_2 + ... + we_d}}$$

If you flip the direction of one of the lines, you get an infinite point in the other direction

All objects in PGA have an direction A^D and position A^P:

$$A = A^D + A^P$$
$$A^D = \sum_i A_i X_i \quad (e_0 \in X_i)$$
$$A^P = \sum_i A_i X_i \quad (e_0 \notin X_i)$$

$$\boxed{\begin{aligned} A^D &= \sum_i A_i X_i \quad (e_0 \notin X_i) \\ A^P &= \sum_i A_i X_i \quad (e_0 \in X_i) \end{aligned}}$$

In 2D and 3D:

$$point = e_0 + A^P$$
$$line = A^D + A^P$$

$$\boxed{point = \overline{e_0} + A^P}$$

In 3D:

$$plane = A^D + A_{15} e_{123}$$
$$= normal + distance$$

$$\boxed{plane = A^D + A_2 e_0}$$

We can take norms to measure the direction or the position:

$$\|A\|_D = \sqrt{\sum_i \left(A_i^D\right)^2}$$
$$\|A\|_P = \sqrt{\sum_i \left(A_i^P\right)^2}$$
$$\|A\| = \sqrt{\sum_i \left| \left(A_i X_i\right)^2 \right|} = \|A\|_P$$

$$\boxed{\|A\| = \|A\|_D}$$

$$A^{-1} = \tilde{A}\frac{1}{\|A\|^2}$$

A is finite if $\|A\|_D \neq 0$, therefore infinite objects have no direction.

We can use this to calculate lengths/areas/volumes/...:

$$length(edge\_loop) = \sum_i \|line_i\|_P$$
$$area(edge\_loop) = \frac{1}{2}\sum_i \|line_i\|_D$$
$$line_i = p_i \;{}_{join}\; p_{i+1}$$
$$area(triangle\_mesh) = \frac{1}{2}\sum_i \|plane_i\|_P$$
$$volume(triangle\_mesh) = \frac{1}{3}\sum_i \|plane_i\|_D$$
$$plane_i = p_i \;{}_{join}\; p_{i+1} \;{}_{join}\; p_{i+2}$$

i. Rasterization

We could just take the x, y components of a d-dimensional point and get an orthographic projection.

But we could also just make a line from the camera (at the origin) to a point and then intersect it with a plane:

$$p_{perspective} = (p_{camera} \;{}_{join}\; p_{vertex}) \;{}_{meet}\; (xy\_plane)$$
$$xy\_plane = p_1 \;{}_{join}\; p_2 \;{}_{join}\; p_3$$
$$depth(p_{vertex}) = \|p_{vertex}\|_P$$

TODO: project(A, B): $P = (A \cdot B^{-1})B$? TODO: project(point_at_origin, B) TODO: project(plane_at_infinity, B)

ii. Raytracing

TODO: $line_1 \wedge line_2$
$= signed\_distance(line_1, line_2)\left\|line_1^D \wedge line_2^D\right\|$
TODO: correlate of signed distance between lines: $line_1 \vee line_2$ -> ray-triangle intersection -> raytraced game
TODO: distance of line to A -> raytraced graphing calculator

TODO: ?

$$distance(A, B) = \begin{cases} \underline{A \wedge \overline{B}} & (type(A) = type(B)) \\ distance(A, project(A, B)) & (type(A) \neq type(B)) \end{cases}$$

TODO: cos(A, B) = A dot A / (A.norm() B.norm()); sin(A, B) = ?

iii. Raymarching

TODO: signed distance of point to A -> raymarched graphing calculator

iv.

V. Motors

Taking $e^{bivector}$ gives us a motor (motion operator), we can apply motors via $MA\tilde{M}$:

$$motor = e^{bivector}$$
$$rotor = e^{bivector^D}; rotor \quad \in motor$$
$$translator = e^{bivector^P}; translator \quad \in motor$$

In 3D, rotors are quaternions.

Where bivector blades are rotations. For example $\frac{\theta}{2}e_{12}$ is an xy rotation by θ degrees and $\frac{d}{2}e_{01}$ is a translation by d:

$$e^{\frac{\theta}{2}e_{12}} = cos\,\frac{\theta}{2} + \left(sin\,\frac{\theta}{2}\right)e_{12}$$
$$e^{\frac{d}{2}e_{01}} = 1 + \frac{d}{2}e_{01}$$

We can interpolate motors with nlerp or slerp, where BA^{-1} is a transformation that brings A to B:

$$nlerp(t, A, B) = \frac{lerp(t, A, B)}{\|lerp(t, A, B)\|}$$
$$lerp(t, A, B) = (1-t)A + tB$$
$$slerp(t, A, B) = (BA^{-1})^t A$$

nlerp does not allow for unnormalized motors, but we will be normalizing them in the physics simulation anyways.

TODO: line forces per https://enki.ws/ganja.js/examples/coffee and https://bivector.net/PGADYN.html