

# Intro to Geometric Algebra

Robin Kahlow

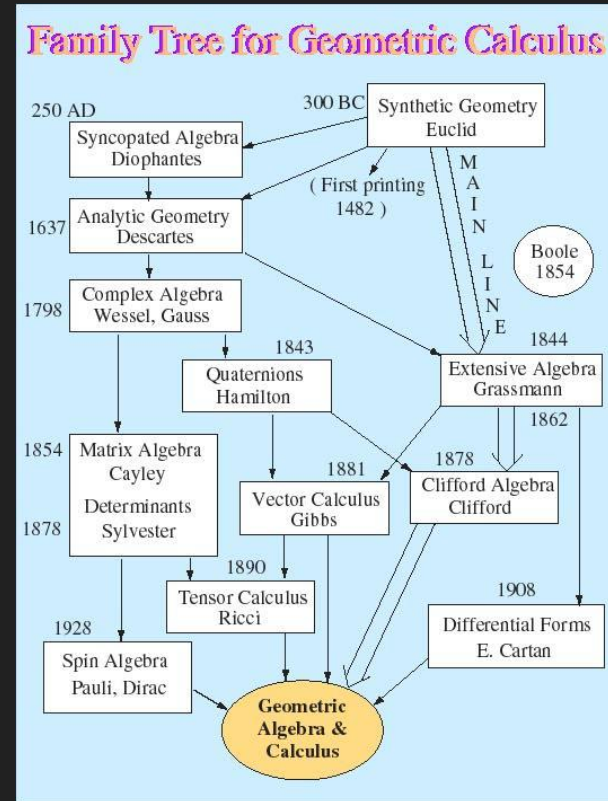
# Outline

1. What is Geometric Algebra?
2. Applications
3. Geometric Algebra Introduction
4. Examples
  - a. Rotors
  - b. Automatic Differentiation
5. Software implementation (TFGA presentation)

# 1. What is Geometric Algebra?

# 1. What is Geometric Algebra?

- Mathematical framework
  - ...**Unifying** traditionally distinct concepts
  - ...**Simplifying** difficult concepts
  - ...**Understanding** abstract concepts
  - ...**Generalizing** existing concepts
- Invented in 19th century by Grassmann, Clifford among others
- Somewhat forgotten in favor of Vector algebra until later (~1960s by David Hestenes)
- “Geometry without algebra is dumb! Algebra without geometry is blind!” - Hestenes



## 2. Applications

## 2. Applications - Computer Graphics



### Computer Geometry

#### VECTOR AND MATRIX ALGEBRA

point

direction

position

rotation

velocity

force

```
// construct lines
line_from_points (p1, p2)
line_from_points_and_dir (p, d)
line_from_plucker (a, b, c, d, e, f)

// construct planes
plane_from_points (p1, p2, p3)
plane_from_point_and_dirs (p, d1, d2)
plane_from_points_and_dir (p1, p2, d)
plane_from_point_and_line (p, l)
plane_from_equation (a, b, c, d)
```

```
// construct transformations
mtx_translate (x, y, z)
mtx_rotate_euler (h, p, b)
mtx_rotate_axis_angle (x, y, z, a)
mtx_look_at (from, to, pole)

// apply transformations
transform_point (M, p)
transform_direction (M, d)
transform_line (M, l)
transform_plane (M, P)
```

```
// LA LA Land
factor_QR (M)
factor_LDL (M)
factor_SVD (M)
factor_LU (M)

.. eigenvalues ..
.. gradient descent ..
.. LMA ..
.. back prop ..
.. AD ..
```

line

plane

matrix

quaternion

dual quaternion

```
// Intersections
intersect_line_plane (l, P)
intersect_plane_plane (P1, P2)
intersect_planes (P1, P2, P3)

// Projections
project_point_plane (p, P)
project_line_plane (l, P)
project_point_line (p, l)
```

```
// construct transformations
quat_from_euler (h, p, b)
quat_from_axis_angle (x, y, z, a)
quat_look_at (from, to, pole)
quat_from_matrix (M)
quat_to_matrix (Q)

// apply transformations
transform_point (Q, p)
transform_line (Q, l)
```

```
// and even more code..
dquat_to_matrix (DQ)
dquat_from_matrix (M)
dquat_from_direction (d)
dquat_from_euler (h, p, b)

.. meshes ..
.. computational geometry ..
.. keeps going ..
```

### Projective Geometric Algebra – recap.

Same as opening but now in PGA - implementation is shorter than function names !  
no coordinates - no chirality - no gimbal lock - no conversions

point

direction

position

rotation

```
// construct lines
line_from_points (p1, p2) = p1&p2
line_from_points_and_dir (p, d) = p&(p+d)

// construct planes
plane_from_points (p1,p2,p3) = p1&p2&p3
plane_from_point_and_dirs (p,d1,d2) = p&(p+d1)&(p+d2)
plane_from_points_and_dir (p1,p2,d) = p1&p2&(p1+d)
plane_from_point_and_line (p,l) = p&l
plane_from_equation (a,b,c,d) = ae1+be2+ce3-de0
```

```
// construct transformations -> motors, not matrices !
translate (x,y,z) = E**(x*e01+y*e02+z*e03)
rotate_euler (h,p,b) = E**(b*e12)*E**(p*e23)*E**(h*e13)
rotate_axis_angle (x,y,z,a) = E**(a*x*e23+a*y*e13+a*z*e12)
look_at (from,to,pole) = (1 + (e1-from)*(to-from))

// apply transformations -> note : all the same..
transform_point (M, p) = M*p~M
transform_direction (M, d) = M*d~M
transform_line (M, l) = M*l~M
transform_plane (M, P) = M*P~M
```

line

plane

matrix

quaternion

```
// Intersections
intersect_line_plane (l, P) = l*P
intersect_plane_plane (P1, P2) = P1*P2
intersect_planes (P1, P2, P3) = P1*P2*P3

// Projections
project_point_plane (p, P) = P|p*P
project_line_plane (l, P) = P|l*P
project_point_line (p, l) = l|p*l
```

```
// construct transformations
quat_from_euler (h, p, b)
quat_from_axis_angle (x, y, z, a)
quat_look_at (from, to, pole)
quat_from_matrix (M)
quat_to_matrix (Q)

// apply transformations
transform_point (Q, p)
```



Geometric Algebra at SIGGRAPH 2019 - [https://www.youtube.com/watch?v=tX4H\\_ctgqYo](https://www.youtube.com/watch?v=tX4H_ctgqYo)

### Traditional approach

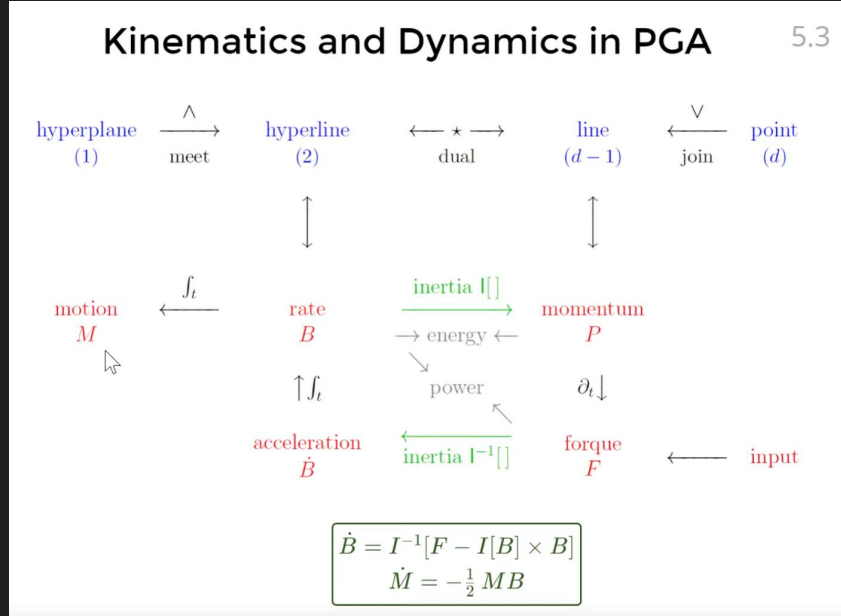
- Many types of objects: **Vector**, **Matrix**, **Quaternion**, **Dual-Quaternion**, ...
- Implementations require a lot of work

### (Projective) Geometric Algebra approach

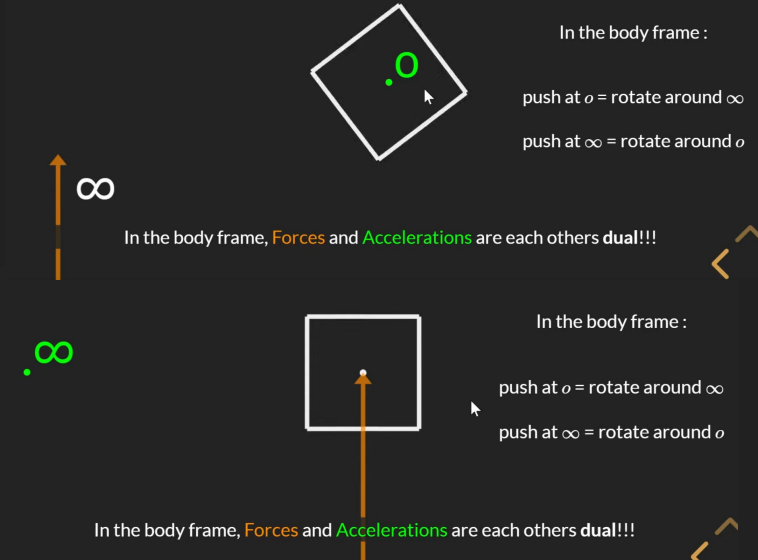
- Only one type of object: **Multivector**
- 1-line implementations with simple GA operations

## 2. Applications - Physics - Classical Mechanics

- Projective Geometric Algebra unifies rotation and translation
  - Force / Torque  $\rightarrow$  "Forque",
- Gives insight on origin of Quantum Spin
- Works in any dimension, equations stay the same (<https://youtu.be/5R2sv9GCwz0?t=674>)



Forques unify linear force and angular torque.



## 2. Applications - Physics - Electromagnetism

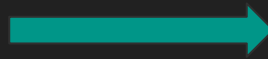
- Electromagnetism
  - Maxwell's 4 equations reduced to a single equation
  - Special Relativity calculations become much easier (eg. how a moving observer sees an E-field)

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon_0}$$

$$\vec{\nabla} \times \vec{B} = \mu_0 \left( \vec{J} + \epsilon_0 \frac{\partial \vec{E}}{\partial t} \right)$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$$

$$\vec{\nabla} \cdot \vec{B} = 0$$



$$\nabla F = \frac{J}{c\epsilon_0}$$

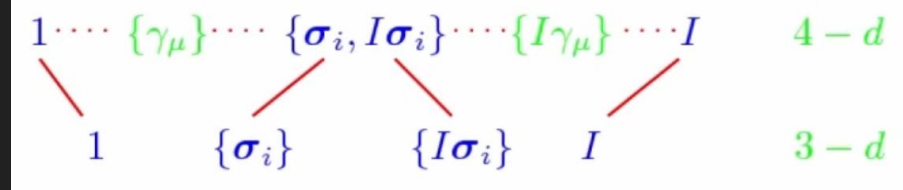
$$(F = E + |B|)$$



## 2. Applications - Physics - QM & GR

- Quantum Mechanics - becomes more interpretable
  - 4 Gamma matrices  $\rightarrow$  4 basis vectors of spacetime (not abstract!)
  - 3 Pauli matrices  $\rightarrow$  3 basis vectors of space (and can be made from the 4 spacetime vectors!)

$$\hat{\sigma}_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \hat{\sigma}_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \hat{\sigma}_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$



- General Relativity (Gravity), two approaches:
  - Usual curved-space approach but with GA simplifying calculations
  - Gauge-Theory Gravity (<https://arxiv.org/abs/gr-qc/0405033>)
    - Space is flat, introduce gauge fields instead
    - Predictions almost all identical as ordinary approach
    - Lead to theoretical discovery of gravitational wave memory-effect

# 3. Geometric Algebra Introduction

# You could have invented Geometric Algebra

- Multiply two vectors  $a = a_1e_1 + a_2e_2 + a_3e_3$   $b = b_1e_1 + b_2e_2 + b_3e_3$

# You could have invented Geometric Algebra

- Multiply two vectors  $a = a_1e_1 + a_2e_2 + a_3e_3$   $b = b_1e_1 + b_2e_2 + b_3e_3$

$$\underline{e_i e_i = 1} \quad \underline{e_i e_j = -e_j e_i}$$

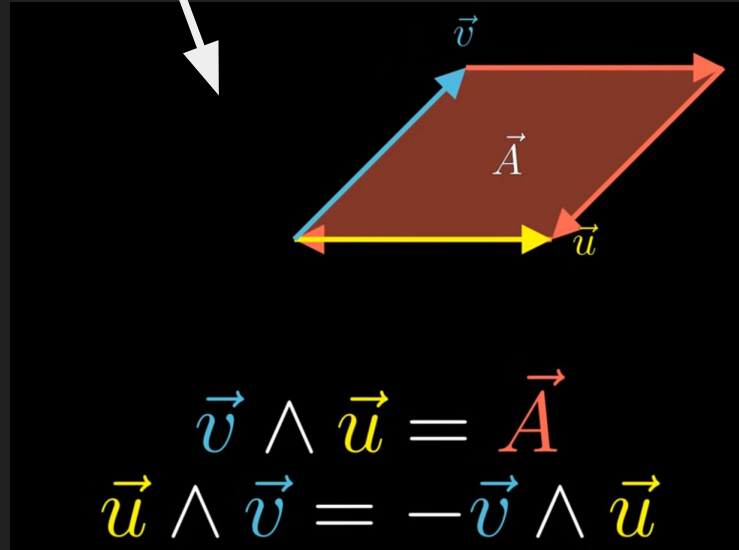
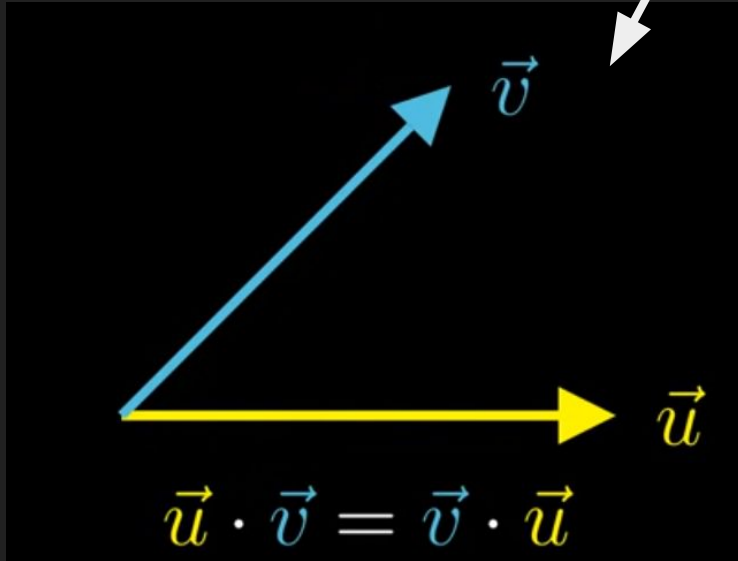
$$\begin{aligned} ab &= (a_1e_1 + a_2e_2 + a_3e_3)(b_1e_1 + b_2e_2 + b_3e_3) \\ &= \underline{a_1b_1e_1e_1} + \underline{a_1b_2e_1e_2} + \underline{a_1b_3e_1e_3} + \\ &\quad \underline{a_2b_1e_2e_1} + \underline{a_2b_2e_2e_2} + \underline{a_2b_3e_2e_3} + \\ &\quad \underline{a_3b_1e_3e_1} + \underline{a_3b_2e_3e_2} + \underline{a_3b_3e_3e_3} \\ &= \underline{a_1b_1 + a_2b_2 + a_3b_3} + \\ &\quad \underline{(a_1b_2 - a_2b_1)e_1e_2} + \\ &\quad \underline{(a_1b_3 - a_3b_1)e_1e_3} + \\ &\quad \underline{(a_2b_3 - a_3b_2)e_2e_3} \\ &= \underline{a \cdot b} + \underline{a \wedge b} \end{aligned}$$

- Result has two different parts:
  - **Scalar part from Dot product**
  - “Bivector” part from “Wedge product”
  - Wedge product in 3D looks like Cross product
  - But: it’s a bivector and not a vector
  - Only in 3D:
    - For every plane, there is a vector orthogonal to it

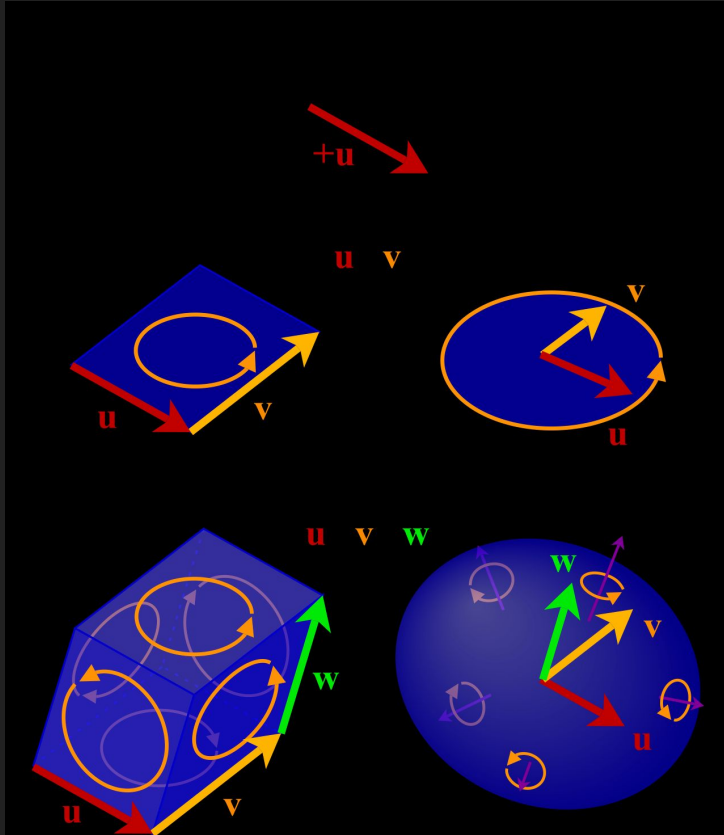
# What does this mean geometrically?

- Multiply two vectors
- Often called Geometric Product but really is just the ordinary product

$$ab = a \cdot b + a \wedge b$$



# Geometry directly representable



Vectors: Line elements

$$e_1, e_2, e_3$$

Bivectors: Plane / Area elements

$$e_1 e_2, e_2 e_3, e_1 e_3$$

Trivectors: Volume elements

$$e_1 e_2 e_3$$

# 4. Examples

## Rotors

## 4. Examples - Rotors

- Multiplying two vectors gave us a bivector part ( $e_1 e_2$ )
- Can be interpreted as plane area element
- Let's try to square a basis bivector

$$e_i e_j = -e_j e_i$$

$$e_i e_i = 1$$

$$(e_1 e_2)^2 = e_1 e_2 e_1 e_2 = -e_1 e_1 e_2 e_2 = -1$$

- The basis bivector is like the Imaginary Unit from Complex Numbers!



## 4. Examples - Rotors

- Traditionally

- A complex number is the sum of a real and imaginary part
- Vectors and complex numbers are seen as separate concepts
- Complex numbers can be used to represent and compose 2D rotation

$$e^{i\varphi} = \cos \varphi + i \sin \varphi$$


$$e^{i\varphi_1} e^{i\varphi_2} = e^{i(\varphi_1 + \varphi_2)}$$

## 4. Examples - Rotors

- In 3D, we have 3 basis bivectors  $\rightarrow$  3 Imaginary numbers  $\rightarrow$  Quaternions

$$e_1 e_2, e_2 e_3, e_1 e_3$$

- 3 planes of rotation
- Quaternions rotate in them just like Complex Numbers did in 2D in 1 plane of rotation
- Formula for applying rotor in 2D to a vector ( $R v$ ) was a special case, in general (any dim):

“Reverse” 

$$R v \widetilde{R}$$
$$e_1 e_2 = e_2 e_1 = -e_1 e_2$$

# 4. Examples

## Automatic Differentiation

# Example - Automatic Differentiation

- So far we had basis vectors squaring to +1
- Can also choose different number, eg. square to 0
- Can be used for forward-mode automatic differentiation:

$$e_1^2 = 0, f(x) = x^2$$

$$\begin{aligned} f(x + e_1) &= (x + e_1)^2 = x^2 + 2xe_1 + e_1^2 \\ &= x^2 + 2xe_1 \end{aligned}$$

- Can be extended to work for multiple inputs and higher order derivatives
- More details in my writeup: <https://discourse.bivector.net/t/automatic-differentiation/289>

# 5. Software Implementation (TFGA Presentation)

<https://tfgap.warlock.ai/#/6>

# More resources on Geometric Algebra

## - Online

- Sudgy Lacmoe's "A Swift Introduction to Geometric Algebra": [https://youtu.be/60z\\_hpEAtD8](https://youtu.be/60z_hpEAtD8)
- Videos on Youtube channel Bivector: <https://www.youtube.com/channel/UCZZ3MA6ChVTViQ8ORmFfCPA>
- Geometric Algebra Discord community: <https://discord.gg/vGY6pPk>
- Bivector website: <https://bivector.net/>
- My own GA tutorials: <https://geometricalgebratutorial.com/>
- Coffeeshop, lots of interactive GA examples: <https://enkimute.github.io/ganja.js/examples/coffeeshop.html>

## - Literature

- Geometric Algebra for Physicists: <https://www.cambridge.org/core/books/geometric-algebra-for-physicists/FB8D3ACB76AB3AB10BA7F27505925091>
- Geometric Algebra for Computer Science: <https://geometricalgebra.org/>
- Geometric Algebra by Eric Chisolm: <https://arxiv.org/abs/1205.5935>
- Linear and Geometric Algebra: <http://www.faculty.luther.edu/~macdonal/laga/index.html>