DRASIL GEOMETRIC ALGEBRA EXTENSION

Christopher Schankula

CAS-741

April 1st, 2025

Recap

Drasil is a platform to capture knowledge and create artifacts from this knowledge.

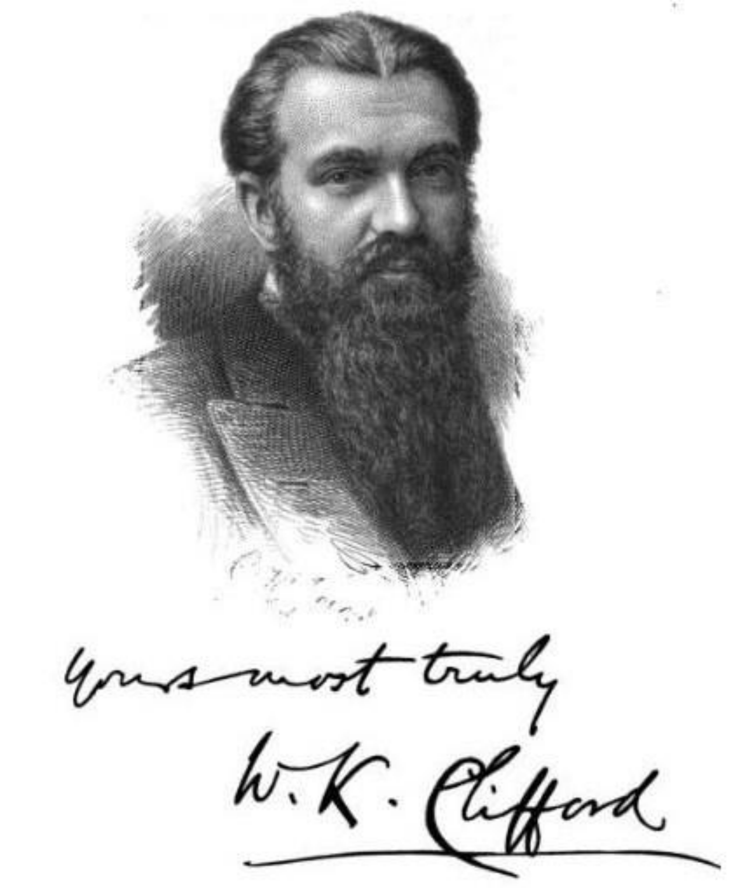
Project goal: Add better support for vectors and matrices to Drasil

Including: checking validity of operations (e.g. sizes of vectors when adding, sizes of matrices when multiplying)

Original idea: use tensors to represent vectors and matrices

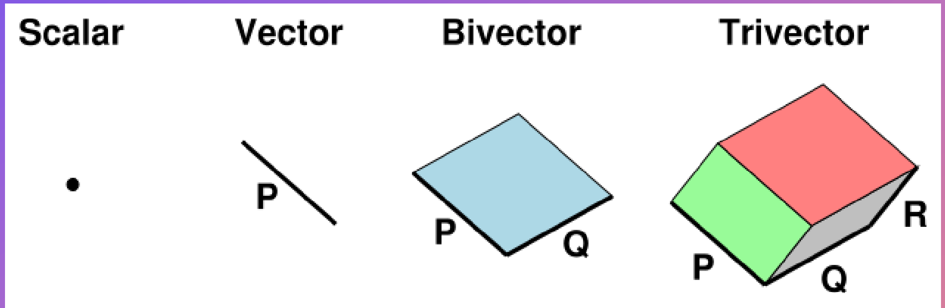
Switched to geometric algebra as we learned more about it

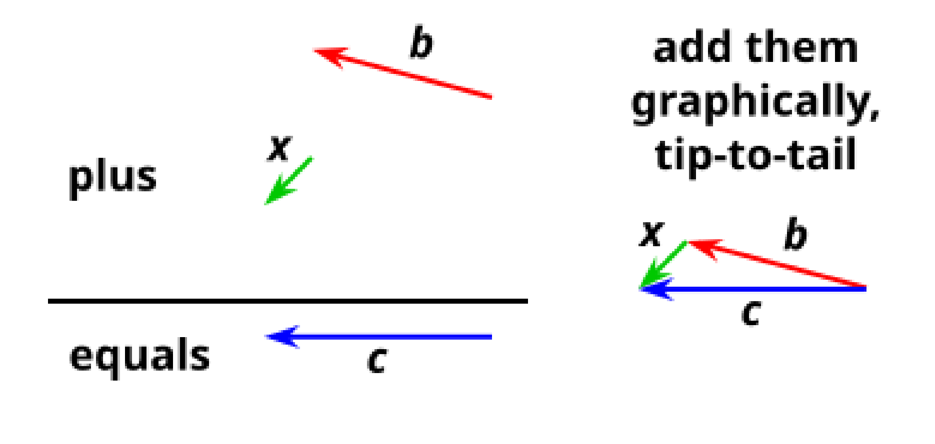
RECAP: GEOMETRIC ALGEBRA

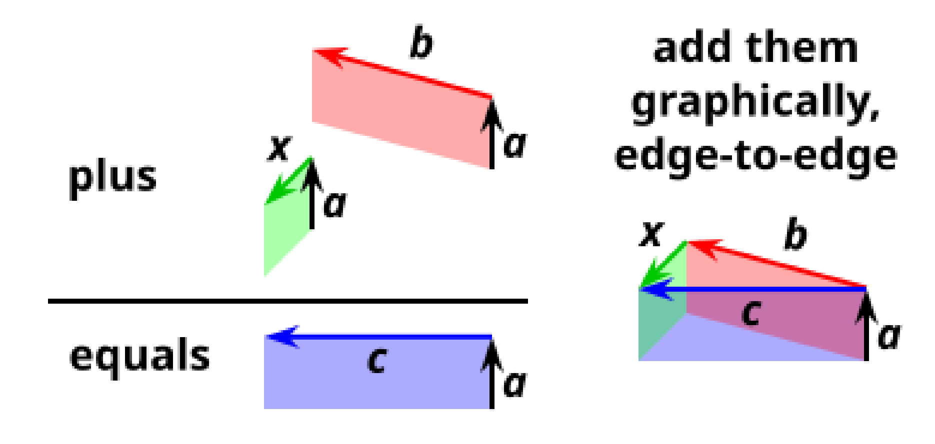


[2]

William Kingdon Clifford (1845-1879)







Right-side graphics from [1]

Geometric Algebra Cont'd

object	**visualized as**	**geometric extent**	**grade**
scalar	point	no geometric extent	0
vector	line segment	extent in 1 direction	1
bivector	patch of surface	extent in 2 directions	2
trivector	piece of space	extent in 3 directions	3

etc.

[1]

Why Geometric Algebra?

Real numbers embedded: grade 0 clifs are just real numbers (not super interesting by itself)

Vectors embedded: grade 1 clifs are just vectors of real numbers (same)

Complex numbers can be represented as a subalgebra using scalars and bivectors in two dimensions.

→ More interesting! Allow us to describe 2D rotations.

Quaternions as well: subalgebra containing scalars and bivectors in three dimensions.

→ Even more interesting! Allows describing 3D rotations.

Operations do not care about their basis

[1]

“Help stamp out cross products!”

- Geometric algebra defines the wedge product, a generalization of cross product.
- Any calculation involving cross products can be generalized / simplified using a wedge product.
- Example: Maxwell’s equations

$$
\begin{aligned}
\nabla \cdot \boldsymbol{E} &= \frac{1}{c\,\epsilon_0}\, c\varrho \\
\nabla \times c\boldsymbol{B} - \frac{\partial}{\partial ct}\boldsymbol{E} &= \frac{1}{c\,\epsilon_0}\, \boldsymbol{j} \\
\nabla \cdot c\boldsymbol{B} &= 0 \\
\nabla \times \boldsymbol{E} + \frac{\partial}{\partial ct} c\boldsymbol{B} &= 0
\end{aligned}
$$

$$\nabla\, F = \frac{1}{c\ \epsilon_0} J$$

[1]

Representing Clifs Component-Wise

- When we want to actually compute using clifs, we need to pick a basis.
- Let's call the basis vectors $\gamma_0, \gamma_1, \gamma_{2,}$etc.
- A convenient basis is an *orthonormal basis* of *spacelike* basis vectors
  - Orthonormal: $\gamma_i \cdot \gamma_j = 0 \text{ for } i \neq j$
  - Orthonormal also means that $|\gamma_i| = 1$ (unit vectors)
  - Spacelike: $\gamma_i \cdot \gamma_i > 0$ $(\gamma_i \cdot \gamma_i = 1$ due to orthonormality)

[1]

Representing Clifs Cont'd

- Example: In a 3D Clifford space, we have the following components:
  - 1 scalar component
  - 3 vector components: γ_0, γ_1, and γ_2.
  - 3 bivector components: $\gamma_0\gamma_{1,}$ $\gamma_1\gamma_{2,}$ and $\gamma_0\gamma_2$
  - 1 trivector component: $\gamma_0\gamma_1\gamma_2$
- Pop quiz! How many components in total, as function of a general dimension d?
  - Answer: 2^d components!

[1]

Geometric Product

- The geometric product is written AB, simply juxtaposing the multiplicands.
- Geometric product is:
  - Associative: $A(BC) = A(BC) = ABC$
  - And distributes over addition: A(B+C) = AB+AC
- Special cases:
  - Scalar times scalar is a scalar
  - Scalar times clif is a clif of the same grade, and this is commutative:
    $$sC = Cs$$
    - Note: geometric product is not commutative in general

[1]

Computing the Geometric Product

- To compute a geometric product, first pick a basis, project into that basis, then multiply out term-by-term and simplify
- Simplification rules:
  - $\gamma_i \gamma_j$ = - $\gamma_j \gamma_i$ for $i \neq j$
  - $\gamma_i \gamma_i = 1$ (spacelike)

[1]

Computing the Geometric Product

- Example: AB where $A = \gamma_1 + 2\gamma_1\gamma_2$ and $B = 3\gamma_3 + 5\gamma_1\gamma_2\gamma_3$
- Step 1: (Pairwise mult.)

	$\gamma 1$	$2\gamma 1\gamma 2$
$3\gamma 3$	$3\gamma 3\gamma 1$	$6\gamma 3\gamma 1\gamma 2$
$5\gamma 1\gamma 2\gamma 3$	$5\gamma 1\gamma 2\gamma 3\gamma 1$	$10\gamma 1\gamma 2\gamma 3\gamma 1\gamma 2$

- Step 2: (Rearrange)

	$\gamma 1$	$2\gamma 1\gamma 2$
$3\gamma 3$	$-3\gamma 1\gamma 3$	$6\gamma 1\gamma 2\gamma 3$
$5\gamma 1\gamma 2\gamma 3$	$5\gamma 1\gamma 1\gamma 2\gamma 3$	$-10\gamma 1\gamma 1\gamma 2\gamma 2\gamma 3$

- Step 3: (Simplify)

	$\gamma 1$	$2\gamma 1\gamma 2$
$3\gamma 3$	$-3\gamma 1\gamma 3$	$6\gamma 1\gamma 2\gamma 3$
$5\gamma 1\gamma 2\gamma 3$	$5\gamma 2\gamma 3$	$-10\gamma 3$

[1]

Computing the Geometric Product

	γ_1	$2\gamma_1\gamma_2$
$3\gamma_3$	$-3\gamma_1\gamma_3$	$6\gamma_1\gamma_2\gamma_3$
$5\gamma_1\gamma_2\gamma_3$	$5\gamma_2\gamma_3$	$-10\gamma_3$

[1]

- Therefore, $(\gamma_1 + 2\gamma_1\gamma_2)(3\gamma_3 + 5\gamma_1\gamma_2\gamma_3) = -3\gamma_1\gamma_3 + 5\gamma_2\gamma_3 + 5\gamma_2\gamma_3 + -10\gamma_3$

Representing Clifs Cont'd

- Pop quiz 2: What is the "shape" of these 2^d components?

$1s$

$1s \quad 1v$

$1s \quad 2v \quad 1b$

$1s \quad 3v \quad 3b \quad 1t$

$1s \quad 4v \quad 6b \quad 4t \quad 1q$

d =	Tot.
0	1
1	2
2	4
3	8
4	16

- Anyone recognize this?

How to represent dimensions

- From my original SRS:
  4. The system shall allow the specification of vector and matrix operations with fixed or variable sizes at specification time.
- To do so, created this data type:

```
data Dimension where
  -- | Fixed dimension
  Fixed :: Natural -> Dimension
  -- | Variable dimension
  VDim  :: String -> Dimension
```

- Allows fixed dimensions of any non-negative size, or named dimensions such as “n”

Representing Clifs in Haskell (Old)

- First iteration:
  ```
  Clif :: Natural -> S.Dimension -> Expr -> Expr
  ```
  - Had no representation for the space/components
  - Good for computing with basis-free, but could not be applied to a particular basis (the single Expr argument makes no sense)
- Next try:
  ```
  Clif :: S.Dimension -> Maybe [Expr] -> Expr
  ```
  - Removed the grade, represents a clif of any grade in the given dimension
  - Allows basis-free or particular-basis computations
  - Problem?

Problem?

- This gets big pretty fast!
- For the implementation, taking advantage of *sparsity* is very important.
  - Imagine adding two 250D vectors, we only need the 250 vector components, not all 2^{250^*} components!
- In fact, many operations, even wedge and geometric products on general clifs, do not use all or even most or many. There is a lot of sparsity!

$*2^{250}$ = $1.8092513943 \times 10^{75}$

Solution

- Lindsay et al [3].'s solution: use dictionaries!
- Instead of a list of all components, use a dictionary indexed by the component and containing the value, symbol or expression in that component.
- Sort components first by grade, then lexicographically, and index them using a binary number:

Component	**1**	γ_0	γ_1	γ_2	$\gamma_0\gamma_1$	$\gamma_0\gamma_2$	$\gamma_1\gamma_2$	$\gamma_0\gamma_1\gamma_2$
Key	000	001	010	100	011	101	110	111

Solution

- In Haskell code:

```
Clif :: S.Dimension -> BasisBlades Expr -> Expr

type BasisBlades e =
  Map BasisKey e -- Map is a dict in Haskell

data BasisKey =
    Y BasisKey
  | N BasisKey
  | E
```

- Example: $\gamma_1\gamma_2$ is represented as `Y (Y (N E))`

What about other bases?

- As mentioned, all the basis vectors so far have been *spacelike,* meaning a vector γ_i such that $\gamma_i \cdot \gamma_i > 0$.
- There are two other kinds:
  - Timelike: a vector γ_i such that $\gamma_i \cdot \gamma_i < 0$.
    - Useful for solving problems in general relativity! [1]
  - Null, or "lightlike": a vector γ_i such that $\gamma_i \cdot \gamma_i = 0$
    - Also useful in special relativity [4]
- Future work includes allowing you to change the types of basis vectors to create a more generalized Clifford space

Conclusion

- Have a representation of Clifford algebra in Drasil!
- Currently supports spacelike vectors of any dimension
- Must take advantage of sparsity to make the generalization worth it
- Still no idea how to represent matrices!
- Documentation (LaTeX) generation of vectors works

Next Steps

- Short-term: Getcode generation working, generated code should take advantage of sparsity too
- Short/medium-term: Create an program to compute the trajectory of a projectile in 2D, and use these new components (representing vectors as clifs, using clif addition, etc) to generate documentation and code
- Medium-term: Implement more operations on dimensions (e.g. concatenating vector of size m and n should yield a vector of size m+n, currently not possible)
- Medium-term: support timelike and lightlike basis vectors
- Long-term: create examples with advanced scientific computations (e.g. gyroscopic procession, Maxwell's equations, special relativity)

References

[1] J. S. Denker, "Clifford Algebra: A Visual Introduction," AV8N, [Online]. Available: https://www.av8n.com/physics/clifford-intro.htm. . [Accessed: Mar. 2025].

[2] Wikimedia Foundation, "William Kingdon Clifford," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/William_Kingdon_Clifford. . [Accessed: 01-Apr-2025].

[3] J. M. Lindsay and A. Soreni-Harari, "Beyond the Limits of Propulsion: Exploring Interstellar Spaceflight," arXiv, 2503.10451, Mar. 2025. [Online]. Available: https://arxiv.org/abs/2503.10451.

[4] E. W. Weisstein, "Lightlike," MathWorld, Wolfram Research, Inc., [Online]. Available: https://mathworld.wolfram.com/Lightlike.html. . [Accessed: Mar. 2025].

THANK YOU! QUESTIONS?

PR: HTTPS://GITHUB.COM/JACQUESCARETTE/DRASIL/PULL/4009
DOCUMENTATION: HTTPS://GITHUB.COM/CSCHANK/GEOMETRICCODEGEN