A MODERN TOOL FOR TEACHING PROJECTIVE GEOMETRIC ALGEBRA

Zachary Leger[a] and **Stephen Mann**[b]

Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
[a]zcjleger@uwaterloo.ca
[b]smann@uwaterloo.ca

Outline

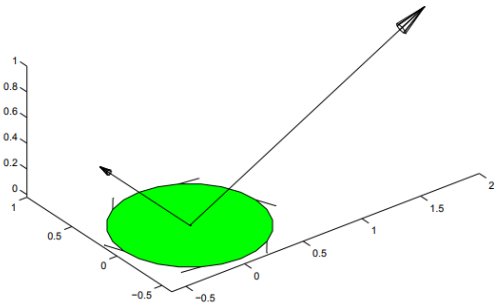
In this extended abstract, we detail the creation of PGABLE, a MATLAB package to aid in the teaching of Projective Geometric Algebra (PGA). We begin by discussing the precursor to PGABLE created in 1999, GABLE, and motivating the creation of PGABLE. Next, we discuss the various internal aspects of GABLE that needed to be modified to create PGABLE. We then describe the PGABLE tutorial we created to teach PGA.

Introduction

GABLE was a prototype MATLAB package supporting a few geometric algebras written in 1999. The main algebra used in GABLE was $\mathbb{R}^{3,0,0}$, although some support was available for the homogeneous model $\mathbb{R}^{2,1,0}$. In addition to being a prototype geometric algebra package, GABLE was designed to support a tutorial introduction to geometric algebra [1]. However, being a prototype, little work has been done on GABLE since 1999.

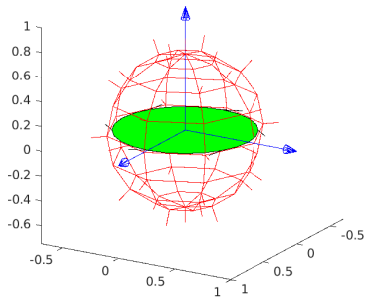
GABLE used MATLAB's command line interface to give users a rich mathematical setting in which to explore geometric algebra. In addition to writing equations, GABLE used MATLAB graphics to draw various objects, including vectors, bivectors, and trivectors (Figure 1).

In recent years, Projective Geometric Algebra (PGA) has become a popular model in the geometric algebra community to represent 3D rigid body motions [2]. As PGA is a 4D geometric algebra (the space $\mathbb{R}^{3,0,1}$), GABLE cannot support it. The goal of this work is to update and extend GABLE into PGABLE, a modern MATLAB package able to support PGA. In addition, we have created a tutorial alongside this software package to introduce users to PGA.



```
draw(2*e1+e3); draw(e2); draw(e1^e2)
```

(a)



(b)

Figure 1: GABLE. (a) Example text input and graphics output; (b) Drawing of vectors, bivectors, and trivectors.

GABLE Implementation Overview

In GABLE, a multivector A is stored via storing the 8×1 column matrix of the following equation:

$$A = [1,\ e_1,\ e_2,\ e_3,\ e_1 \wedge e_2,\ e_2 \wedge e_3,\ e_3 \wedge e_1,\ e_1 \wedge e_2 \wedge e_3] \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_{12} \\ A_{23} \\ A_{31} \\ A_{123} \end{bmatrix}$$

and we denote by $A.m = [A_0, A_1, A_2, A_3, A_{12}, A_{23}, A_{31}, A_{123}]^T$. For example, the basis vectors e_1, e_2, e_3 were implemented as functions in MATLAB, with each function returning the corresponding 8×1 matrix for that basis vector.

The geometric product $A\,B$ of two multivectors A and B is computed by converting A into an 8×8 matrix $[A]$ that is the linear function computing "the geometric product $A*$", and computing $[A]B.m$. The other products are implemented in a similar manner. MATLAB objects were used to overload the arithmetic operators `*` (geometric product), `^`(outer product), `+` (addition). The inner product was implemented as a MATLAB function `inner()`.

Elements of the algebra are drawn by explicitly calling a `draw()` routine. Lines on bivectors and on trivectors are used to indicate the orientation (sign) of the object. GABLE provided a variety of routines for rotating objects, although with more modern versions of MATLAB, rotation and scaling of objects in the graphics window can be done using the mouse.

Extending GABLE to PGABLE

Many of the design decisions used in GABLE were kept in PGABLE: multivectors are represented with column matrices; products of two multivectors are computed by expanding the column of one element into a square matrix and multiplying the column matrix of the other element by this square matrix; and the graphical representation of basic elements (vectors, bivectors, and trivectors) remains the same. MATLAB has evolved since 1999, and a variety of changes were made to update GABLE to current MATLAB, mostly dealing with how objects are represented in MATLAB. In particular, classes, which were implicitly represented in 1999, are now explicitly represented in the MATLAB language, and static methods were introduced.

Dimensionality Increase

With the addition of e_0 in PGA, elements in PGA must be represented by an array of 16 doubles. We chose to use the basis

$$1,\ e_0,\ e_1,\ e_2,\ e_3,\ e_0 \wedge e_1,\ e_0 \wedge e_2,\ e_0 \wedge e_3,\ e_1 \wedge e_2,\ e_1 \wedge e_3,\ e_2 \wedge e_3,$$
$$e_0 \wedge e_1 \wedge e_2,\ \ e_0 \wedge e_1 \wedge e_3,\ \ e_0 \wedge e_2 \wedge e_3,\ \ e_1 \wedge e_2 \wedge e_3,\ \ e_0 \wedge e_1 \wedge e_2 \wedge e_3$$

With this change in data structure, the pre-existing operations such as the geometric product, outer product, inner product, inverse and reverse needed to be rewritten. However, the underlying mathematics remained the same. It is simply the case that some 8×8 matrices were replaced by mathematically analogous 16×16 matrices.

Rendering Redesign

Traditional visual representations of geometric algebras draws scalars as points, vectors as lines, bivectors as planes, trivectors as volumes, etc., all centered at the origin. PGA is a model of geometric algebra which can represent points, lines, planes and volumes offset from the origin. This means the rendering abilities of GABLE had to be extended to draw elements that are offset from the origin.

Extending the rendering abilities alone did not suffice to accurately represent PGA. PGA is fundamentally different than most other geometric algebra models (such as the homogeneous model or the conformal model) in that vectors represent planes, not lines or points. Additionally, bivectors represent lines and trivectors represent points in PGA. This means that some elements possess a fundamental ambiguity as to how to represent them. Consider the element e_1. In the traditional view, this is represented by a unit arrow pointing along the positive direction on the e_1 axis. However, in the PGA view, this element represents the (oriented) plane whose *normal* is e_1. For this reason, we let PGABLE possess two modes to users: GA mode and PGA mode. Depending on the mode the user is currently in, the respective representation is drawn to the user when a draw call is made. When in GA mode, attempting to visualize a PGA element is not permitted.

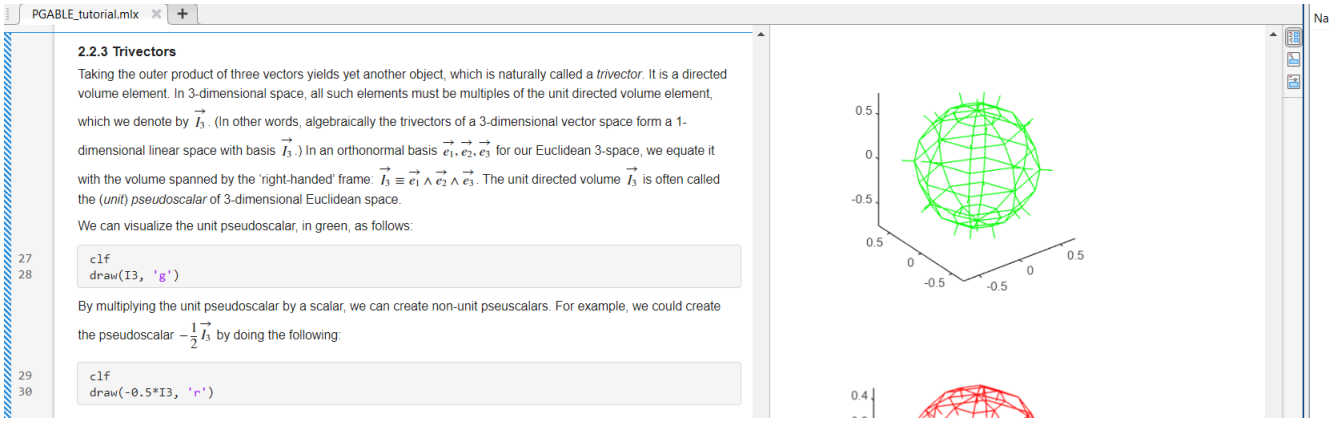


Figure 2: Live Script example.

Additional Operations

To show the full capabilities of PGA, we needed to add new computational abilities to GABLE. First, since PGA does not possess the traditional geometric algebra dual (since its pseudoscalar is not invertible), users typically want other types of dualization. We provided the Hodge dual [2] and the Poincaré Dual Map [5].

As the `join` operation is computed differently in PGA than in the standard geometric algebra, we provide a new operation `PGAjoin` which computes

$$\star^{-1}(\star A \wedge \star B),$$

which is the PGA join operation described in [2]. Since the PGA `meet` operation is simply the outer product, for the sake of consistency, we have provided a `PGAmeet` operation that performs a call to the outer product function.

Since PGABLE contains a GA and PGA mode, we needed a way to convert between PGA elements and GA elements for each mode. By default, elements of GA are interpreted as their direct PGA equivalent in PGA mode (for example, the element e_1 in GA is simply interpreted as e_1 in PGA). This can be accomplished by passing the GA element through the `PGA` constructor. However, users may want to have their GA elements converted into PGA elements through a geometric interpretation (for example, the arrow representing element e_1 in GA would be $e_2 \wedge e_3$ in PGA). Thus, we have provided users with the function `geoPGA` to cast GA elements into PGA using a geometric interpretation.

There are two norms in PGA, the Euclidean norm $||\cdot||$ and the vanishing norm $||\cdot||_\infty$. The `norm` function of PGA computes the Euclidean norm. For the vanishing norm, we provide the function `vnorm`, which is computed by the formula

$$||A||_\infty = ||\star A||$$

as described in [2].

Tutorial

The original GABLE tutorial [1] was a PDF document that introduced users to geometric algebra using the GABLE package. For PGABLE, we have provided a similar document. Notably, however, we have replaced the section detailing the homogeneous model with a section detailing PGA. Additionally, we have provided series of MATLAB Live Scripts so that users can follow the tutorial directly in MATLAB, running the example code snippets in real time; see Figure 2.

References

[1] Leo Dorst, Stephen Mann, and Tim Bouma. GABLE: A Matlab tutorial for geometric algebra. Available at `https://cs.uwaterloo.ca/~smann/GABLE/`, 1999.

[2] A Guided Tour to the Plane-Based Geometric Algebra PGA. Leo Dorst and Steven De Keninck `https://bivector.net/PGA4CS.html`

[3] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science*. Morgan-Kaufmann, 2007.

[4] Stephen Mann, Leo Dorst, and Tim Bouma. The Making of GABLE, a Geometric Algebra Learning Environment in Matlab. In *Advances in Geometric Algebra with Applications in Science and Engineering*. Editors: E. Bayro-Corrochano and G. Sobczyk. Birkhäuser, 2001.

[5] Jeremy Ong.
`https://www.jeremyong.com/klein/geometry-potpourri/#the-poincare-dual-map`