Euclidean Geometry and Geometric Algebra

1 Introduction

Geometric algebra is a very natural language for describing concepts in Euclidean geometry, but one aspect that can confuse people at first is that there is no single unique way to work. Different problems can require different algebras for their optimal solution and new uses for geometric algebras are being discovered constantly. In a recent SIGGRAPH presentation [1] Charles Gunn and Steven De Keninck described a framework for Euclidean geometry based on planes as geometric primitives, which they call PGA. This article describes how PGA relates to other geometric algebras and explains why it may be the ultimate algebra for graphics.

Most introductions to geometric algebra (GA) start with a picture of vectors as rays out from a common origin [2]. These have an inner product, the traditional scalar product, and an outer product that encodes the plane swept out by the two vectors. These are united in a geometric product in the famous relationship

$$ab = a\cdot b + a\wedge b. \tag{1}$$

The geometric product is the simplest product we can construct that is invertible. Given ab and a, with $a^2 \neq 0$ we can recover b simply by multiplying the right-hand side by $a^{-1} = a/a^2$. Building out the algebra of 2D vectors leads us to 'invent' complex numbers, and in 3D to the discovery of the quaternion algebra. Indeed, the quaternions should be understood via their embedding in the geometric algebra of space, where they represent planes and generate rotations about the origin.

The fact that quaternions are the optimal algebra for working with rotations is well known to graphics and games programmers, and most game engines have an optimised quaternion library buried somewhere in their code (typically in the physics engine). But there are at least two further structures that are often employed in graphics code that do not sit obviously in this framework: projective geometry and dual quaternions.

2 Projective Geometry

Projective geometry lies at the heart of the graphics pipeline and solves a fundamental problem. If we fix an origin and a frame, and compute the coordinates of a vector x in this frame we find three numbers (x_1, x_2, x_3). We can encode rotations around the origin as 3×3 matrices acting on these coordinates, but translations

Chris Doran, June 2020.

cannot be encoded as a linear transformation. We want to combine rotations and translations to get from object space to world space to camera space and would like to do this in a simple, unified way. The trick is to introduce a fourth coordinate and work with 4-dimensional vectors with coordinates $(x_1, x_2, x_3, 1)$. In GA we write this as working with the vector X:

$$X = x + e_4 \tag{2}$$

where $x = x_1e_1 + x_2e_2 + x_3e_3$ is the usual Euclidean 3-vector. Projective geometry linearises translations and from this starting point a very rich algebra can be developed. Though most of this is irrelevant in graphics, which only really exploits the unification of rotations and translations. The line L between two points is described by their outer product

$$L = X_1 \wedge X_2 \tag{3}$$

and these inhabit a 6-dimensional space. Similarly, the outer product of three points defines a plane P

$$P = X_1 \wedge X_2 \wedge X_3. \tag{4}$$

The somewhat mysterious concept of projective duality is encoded simply as multiplication by the pseudoscalar $I_4 = e_1e_2e_3e_4$, which interchanges outer and inner products. As a consequence, the line L resulting from the intersection of two planes is found by

$$L = I(P_1 \times P_2) \tag{5}$$

where $A \times B$ denotes the antisymmetric product of two multivectors.

The projective representation is homogeneous, so X and λX represent the same point, for any non-zero scalar λ, and intersection algorithms are robust against special cases. For example, two parallel planes meet in a 'line at infinity', which has a finite representation. Many important geometric theorems can be proved efficiently in this setup, and duality often gives you two proofs for the price of one. Most of the main results were worked out by Hestenes and Ziegler [3], which built on a rich literature on exterior theory and duality going back to the 19th century.

But one aspect of this work remained disappointing: projective transformations exist as operations outside the algebra. One of the key unifying principles of 3D GA is that vectors and the operations on them are all part of the same algebra. That is lost in the projective setup.The reason for this is that projective transformations do not preserve angles, so do not naturally sit inside GA. A further consequence of this is that there is very little use for the geometric product, except at intermediate steps in derivations. All results usually involve exterior products or duality.

3 Dual Quaternions

There is a way to perform Euclidean transformations using GA elements that was introduced by Clifford himself, and that is via the dual quaternions. These are not as widely used in the graphics community as quaternions, but they are used for skinning operations in animation, and are sometimes employed in rigid body physics solvers. A dual quaternion is made up from a pair of quaternions:

$$D = q_1 + \epsilon q_2 \tag{6}$$

where ϵ is an algebraic entity invented solely to have the property that

$$\epsilon^2 = 0. \tag{7}$$

We then find that normalised dual quaternions, $DD^\dagger = 1$, form a group that includes rotations and translations. These act on elements of the form

$$Y = 1 + \epsilon(y_1 i + y_2 j + y_3 k) \tag{8}$$

though the operation is not simply $DYD^\dagger$ as a further sign flip is required on the ϵ term. None of this is very obvious or looks very natural, and dual quaternions have tended to be a niche topic.

4 Conformal GA

In 2000 a new use for geometric algebra started to evolve based on conformal geometry. This is now known as conformal geometric algebra (CGA). The building blocks of CGA had also been around since the 19th century, but it wasn't until the start of the 21st century that all of the elements were pulled together. The starting point for CGA is to represent the Euclidean point x as the vector

$$X = x + m - \tfrac{1}{2}x^2 n \tag{9}$$

where m and n are a pair of new vectors, orthogonal to the e_i, that satisfy:

$$m^2 = n^2 = 0, \quad m \cdot n = 1. \tag{10}$$

Vectors with a zero norm are called 'null' vectors. They may be unfamiliar, but are a standard feature of special relativity. These definitions ensure that

$$X^2 = 0. \tag{11}$$

So points are also represented as null vectors in a space 2 dimensions higher. The key concept behind the conformal model is that the inner product of two points is related to the distance between them. Consider

$$X \cdot Y = -\tfrac{1}{2}(x^2 + y^2 - 2x \cdot y) = -\tfrac{1}{2}(x - y)^2. \tag{12}$$

This explains why points have to be null vectors. The distance of a point from itself must be zero.

The representation is also homogeneous, so X and λX represent the same point. Our final expression for distance is therefore

$$d(x, y)^2 = \frac{-2X \cdot Y}{(X \cdot n)(Y \cdot n)}. \tag{13}$$

Euclidean transformations must leave distances unchanged, so in CGA they must preserve the inner product. Transformations that achieve this can always be built from elements in the algebra. The transformation must also leave n invariant, which means that they are built from even elements that commute with n. With a bit of work one finds that elements satisfying this latter requirement have the form

$$M = q_1 + e_1 e_2 e_3 n q_2 \tag{14}$$

where q_1 and q_2 are quaternions in the 3D Euclidean algebra. We have rediscovered precisely the dual quaternions, and the mysterious ϵ turns out to be a null 4-vector $I_3 n$ where

$$I_3 = e_1 e_2 e_3. \tag{15}$$

Normalised elements, $MM^\dagger = 1$ are called rotors (sometimes 'motors') and these act on points of the form X to perform rotations and translations.

This is another grand unification. In the same way that quaternions are best understood in 3D GA, dual quaternions arise naturally in the 5D CGA. The CGA has many other wonderful features. Points, lines, planes, circles and spheres all have simple expressions, as do intersection algorithms, and both the objects and the operations on them are contained in the same algebra.

CGA clearly has much to offer graphics, but there is a problem. The underlying representation is built on a 5D vector space, which is quite verbose and for many cases not as efficient as simple projective geometry. For example, the line-plane intersection returns a point pair, with one of the points at infinity. This is technically correct, but an unwanted complication in practice. Returning to the representation of X (eq. 9) we see that the origin is represented by m, and in many applications we can drop the $x^2 n$ term. But the origin m is not the vector n that appears in the dual quaternions that drive transformations. It looks very much like you need both m and n, hence the full 5D algebra, to combine the advantages of projective geometry and dual quaternions.

Chris Doran, June 2020.

5 PGA

But it turns out there is a way to work entirely in a 4D space, with points lines and planes and the generators of rotations and translations all in the one algebra and retaining the benefits of a homogeneous representation. The idea originated in the robotics community [4] and it is this idea, PGA, that Gunn and De Keninck descibed at SIGGRAPH.

We can motivate the algebra by going back to the CGA form of dual quaternions and noting that they are constructed entirely in an algebra generated by $\{e_1, e_2, e_3, n\}$. So we ask the obvious question: what objects can we build in CGA that contain only these 4 generators? First look at vectors. We suspect that we really want to kill off the x^2 term to get back to familiar projective geometry, so we form

$$X \wedge n = x \wedge n + m \wedge n. \tag{16}$$

This still contains a factor of m. However, if we dualise in 5D space with

$$I_5 = I_3 m \wedge n, \tag{17}$$

we form

$$I_5(X \wedge n) = -I_3 x n + I_3. \tag{18}$$

This is great! We now have an object constructed entirely from our basis elements $\{e_1, e_2, e_3, n\}$. Euclidean rotors will commute with both I_5 and n so this object will transform correctly as a point under Euclidean transformations. In components

$$I_5(X \wedge n) = n(x_1 e_3 e_2 + x_2 e_3 e_1 + x_3 e_2 e_1) + e_1 e_2 e_3 \tag{19}$$

which is precisely the representation used by PGA. We can also see a close parallel between this and the representation of points in dual quaternions. Of course, this object is a trivector, which is an unusual thing to use to represent a point, but let's see where this thinking leads us.

Next consider a line between x and y. In CGA we form

$$X \wedge Y \wedge n = x \wedge y \wedge n + (x - y) m \wedge n \tag{20}$$

and now we know what to do. We dualise this in CGA to form

$$I_5(X \wedge Y \wedge n) = -(I_3 x \wedge y) n + I_3(x - y) \tag{21}$$

and again we are back in the algebra generated by $\{e_1, e_2, e_3, n\}$. So now lines are represented as bivectors and a pattern is starting to emerge. Finally we look at the plane formed by three points

$$X \wedge Y \wedge Z \wedge n = x \wedge y \wedge z \wedge n + (x \wedge y + y \wedge z + z \wedge x) \wedge m \wedge n \tag{22}$$

and dualising this we get a vector of the form

$$I_5(X \wedge Y \wedge Z \wedge n) = \lambda n + d \tag{23}$$

where d is a Euclidean vector normal to the plane. We have planes as grade 1 objects, so everything is 'dual' to the usual representation. But we have an algebra with all the properties we want for graphics. Points, lines and planes are all represented, and rotations and translations on these objects are performed by even elements in the same algebra. The representation is homogeneous and immediately consistent with the projective representation in graphics. Even better, given a line L as a bivector, rotations about that line are formed simply be exponentiating it! (See the SIGGRAPH course notes for more details [1].)

There are many useful identities that fall out from this algebra, but a key point is that it successfully captures concepts like orthogonality, which are harder to capture in standard projective geometry. To see this, consider the product of two normalised planes:

$$p_1 p_2 = p_1 \cdot p_2 + p_1 \wedge p_2 = \cos\theta + L \sin\theta \tag{24}$$

where θ is the dihedral angle between the planes, and L is the line of intersection, which generates rotations. Both parts of the product are geometrically relevant. We can also form decompositions of the form

$$X = Xpp = (X \cdot p)p + (X \wedge p)p \tag{25}$$

where X is a point and p is a plane. This decomposes X into the point $(X \cdot p)p$, which is the perpendicular projection of X onto the plane p, and the term $(X \wedge p)p$ which is along the direction perpendicular to the plane. This is a natural decomposition and again shows how all aspects of the geometric product make sense.

So we have an algebra that does everything we want, and all we had to do was make the '4th' vector in projective geometry a null vector. Why is this not more widely known? Part of the answer is that the idea that points should be represented as trivectors seems unnatural at first. There is also a price you pay for having a null basis element, which is that the pseudoscalar now squares to zero. You lose the natural concept of duality as multiplication by I. PGA does have a simple notion of duality, which is required for the join operation, though it is not performed by an element in the algebra. But this is a small price to pay, particularly as at the data level the duality operation is a trivial re-labeling. There is an important point here. This duality operation does not commute with transformations. If you take a point, dualise it to a plane, transform the plane, and dualise back you do not get the same result as transforming the point. For practical applications we have to fix the representation with planes as grade 1, lines grade 2 and points grade 3. Again,

this is not a problem in practice, and the fact that there is only one way to work is helpful as it uniquely fixes your data types.

This leads on to the key point that PGA is very hardware friendly. In applications you always work with elements that are purely even or purely odd. These are 8 dimensional objects, so the absolute worst case multiplication operation takes $8 \times 8 = 64$ operations. The same as 4×4 matrix multiplication. But in practice a number of these products are zero so you usually end up doing better. Also, everything fits neatly into 4 slots, which is good for both GPU and SIMD implementations. The underlying products can be blocked up in 4s, and most boil down to quaternion multiplication. If you already have an optimised quaternion library, then the products are extremely fast. Finally, the entire algebra sits within the CGA, so it is easy to jump up to the full CGA should we need to take advantage of some of the operations there. This could be handy when using constructs like bounding sphere hierarchies.

After much searching, it does finally look like we have found the optimal algebra for 3D geometry computation.

References

[1] Charles Gunn and Steven De Keninck, *Geometric Algebra for Computer Graphics*. SIGGRAPH 2019. https://bivector.net/doc.html.

[2] Chris Doran and Anthony Lasenby, *Geometric Algebra for Physicists*. Cambridge University Press (2003).

[3] D. Hestenes and R. Ziegler, *Projective Geometry with Clifford Algebra*. Acta. Appli. Math. 23(25), 1991.

[4] Jon Selig, *Clifford algebra of points, lines and planes*. Robotica 18(545), 2000.