[**Geometry without geometric algebra**](https://0fps.net/2018/11/04/geometry-without-geometric-algebra/)

Posted on [November 4, 2018](https://0fps.net/2018/11/04/geometry-without-geometric-algebra/)by [mikolalysenko](https://0fps.net/author/mikolalysenko/)

When I was younger I invested a lot of time into studying [geometric algebra](https://en.wikipedia.org/wiki/Geometric_algebra).  Geometric algebra is a system where you can add, subtract and multiply oriented linear subspaces like lines and hyperplanes (cf. [Grassmanian](https://en.wikipedia.org/wiki/Grassmannian)). These things are pretty important if you’re doing geometry, so it’s worth it to learn many ways to work with them. Geometric algebra emphasizes [exterior products](https://en.wikipedia.org/wiki/Exterior_algebra) as a way to parameterize these primitives (cf. [Plücker coordinates](https://en.wikipedia.org/wiki/Pl%C3%BCcker_embedding)).  Proponents claim that it’s simpler and more efficient than using “linear algebra”, but is this really the case?

In this blog post I want to dig into the much maligned linear algebra approach to geometry.  Specifically we’ll see:

1. Two ways to parameterize a linear subspace using matrices
2. How to transform subspaces
3. How to compute the intersection and span of any two subspaces

Encoding flats

A [subspace](https://en.wikipedia.org/wiki/Linear_subspace) of a vector space is a subset of vectors which are closed under scalar addition and multiplication.  Geometrically they are points, lines and planes (which pass through the origin unless we use [homogeneous coordinates](https://en.wikipedia.org/wiki/Homogeneous_coordinates)).  We can represent a k-dimensional subspace of an n-dimensional in two ways:

1. As the [span](https://en.wikipedia.org/wiki/Linear_span) of a collection of k vectors.
2. As the solution to a set of n – k [linear equations](https://en.wikipedia.org/wiki/Linear_equation).

These can be written using matrices:

* In the first case, we can interpret the span of k vectors as the [image](https://en.wikipedia.org/wiki/Image_(mathematics)) of an n-by-k matrix M : R^k \to R^n
* In the second, the solution of a set of n-k linear equations is another way of saying the [kernel](https://en.wikipedia.org/wiki/Kernel_(linear_algebra)) of an (n – k)-by-n matrix, A : R^n \to R^{n - k}.

These two forms are [dual](https://en.wikipedia.org/wiki/Dual_space) to one another in the sense that taking the matrix transpose of one representation gives a different subspace, which happens to be it’s [orthogonal complement](https://en.wikipedia.org/wiki/Orthogonal_complement).

Intersections and joins

The best parameterization depends on the application and the size of the flat under consideration.  Stuff that’s easy to do in one form may be harder in the other and vice-versa.  To get more specific, let’s consider the problem of intersecting and joining two subspaces.

If we have a pair of flats represented by systems of equations, then computing their intersection is trivial: just concatenate all the equations together.  Similarly we can compute the smallest enclosing subspace of a set of subspaces which are all given by spans: again just concatenate them.  And we can test if a subspace given by a span is contained in one given by equations by plugging in each of the basis vectors and checking that the result is contained in the kernel (ie maps to 0).

Linear transformations of flats

Depending on the form we pick flats transform differently.  If we want to apply a linear transformation T : R^n \to R^n to a flat, then we need to consider it’s encoding:

1. If the flat is given by the image of a map, M(V), then we can just multiply by T
2. And if a flat is a system of equations, ie A^{-1}(0), then we need to multiply by the inverse transpose of T.

The well known rule that normal vectors must transform by inverse transposes is a special case of the above.

Conversion

Finally we can convert between these two forms, but it takes a bit of work.  For example, finding the line determined by the intersection of two planes through the origin in 3D is equivalent to solving a 2×2 linear system.  In the general case one can use [Gaussian elimination](https://en.wikipedia.org/wiki/Kernel_(linear_algebra)#Computation_by_Gaussian_elimination).

Is this less intuitive?

I don’t really know.  At this point I’m too far gone to learn something else, but it’s much easier for me to keep these two ideas in my head and just grind through some the same basic matrix algorithm over and over than to work with all the specialized geometric algebra terms.  Converting things into exterior forms and plucker coordinates always seems to slow me down with extra details (is this a vee product, inner product, circle, etc.), but maybe it works for some people.