Simple 4D Rigid Body Dynamics

DIEGO LOPEZ, McGill University

Code: https://github.com/Symmetries/4d-rigid-body-dynamics/
Demo: https://diegolopez.me/4d-rigid-body-dynamics/

Additional Key Words and Phrases: geometric algebra, 2-vectors, multivectors, rotors

ACM Reference Format:
Diego Lopez. 2022. Simple 4D Rigid Body Dynamics. 1, 1 (May 2022), 2 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

In order to generalize the equations of motion to higher dimensions, one must deal with the fact that we can no longer use the cross product and must rely on other mathematical constructs to encode quantities such as orientation, angular velocity, angular momentum, torque, angular inertia. One way to do this, as presented by Bosch [2020], is to use objects from geometric algebra called multivectors, more specifically 2-vectors and rotors. We begin by giving some of the background theory. A more in depth discussion of this topic can be found in depth in Macdonald [2010].

To start, we present 2-vectors. Geometrically, bivectors can be thought like "oriented surface elements", much like standard vectors can be thought of as "oriented line elements". Thus, given a vector space V, a 2-vector can be formed by taking for vectors u, v and taking the exterior product $\wedge$, forming the 2-vector $u \wedge v$. Some notable features of this product are the following properties:

- $\lambda(u \wedge v) = (\lambda u) \wedge v = u \wedge (\lambda v)$,
- $(u + v) \wedge w = u \wedge w + v \wedge w$
- $u \wedge (v + w) = u \wedge v + u \wedge w$
- $u \wedge v = -v \wedge u$

in other words, it is linear in both arguments and *anti-symmetric*. We note that this implies that $u \wedge u = 0$. The fact that this product is anti-symmetric is key: this is was allows us to encode rotations in arbitrary dimensions, encoding the fact that rotating in the "opposite direction" coincides exactly with the notion of 2-vector negation.

To give a concrete example, bivectors formed from vectors in $\mathbb{R}^2$ form a one dimensional vectors space with basis vector $e_0 \wedge e_1$. Another example is that in $\mathbb{R}^3$, 2-vectors form a 3 dimensional vector space with basis

$$\{e_0 \wedge e_1, e_0 \wedge e_2, e_1 \wedge e_2\}.$$

In general, 2-vectors in $\mathbb{R}^n$ form an $\binom{n}{2}$ dimensional vector space.

This process can be generalized to obtain general k-vectors, using the fact that

$$u \wedge (v \wedge w) = (u \wedge v) \wedge w,$$

i.e. the exterior product is *associative*.

Now, we are ready to present multivectors. Similar to complex numbers or quaternions, multivectors are linear combinations of scalars, vectors, 2-vectors, etc. For instance, the following is a multivector in $\mathbb{R}^2$:

$$2 + e_0 + 3e_1 + 5e_1 \wedge e_2.$$

The operations of addition and scalar multiplication are done in the obvious way. We now introduce the geometric product, which operates on multivectors, and this is the key to the equations of motion in higher dimensions. Unfortunately, there is no simple way to describe the geometric product with an equation, but the following properties completely characterize it: let V be equipped with a bilinear map $g : V \times V \to K$, where K is the underlying field. For our purposes, we will be taking $K = \mathbb{R}$, g to be the dot product and $V = \mathbb{R}^4$. The *geometric product* is a map from multivectors to multivectors satisfying:

- $vv = v^2 = g(v, v)$
- $1a = a1 = a$
- $a(bc) = (ab)c$
- $a(b + c) = ab + bc$
- $(a + b)c = ac + bc$

where v is a vector, 1 is the multiplicative identity in K and a, b, c are multivectors. From now we will denote $g(u, v)$ by $u \cdot v$. It is useful to also discuss how the geometric product works on vectors:

$$uv = u \cdot v + u \wedge v.$$

Thus, in general, the geometric product of two vectors results in a multivector: a sum of a scalar and a 2-vector. We note that for any two distinct standard basis vectors in $\mathbb{R}^n$, we have

$$e_i e_j = e_i \cdot e_j + e_i \wedge e_j = e_i \wedge e_j.$$

Thus, from now on we write $e_i e_j$ instead of $e_i \wedge e_j$, but only for distinct standard basis vectors. Indeed, we have $e_i e_i = 1$ if the standard basis vectors are not distinct. Using these basic rules, the product of any multivectors where the underlying vector space is $K = \mathbb{R}^4$ can be computed. It merely suffices to write each k-vector as a linear combination of basis k-vectors and using the properties of the geometric map to eventually find the product. The general formula for the geometric product of two multivectors is too complicated, however, and for our purposes we only need to consider products of specific kinds of multivectors (rotors and 2-vectors).

That said, we now introduce rotors. To that end, we must introduce the operation of 'reversion' of a multivector. To reverse a multivector, we simply express the multivector as a linear combination of products of basis vectors, and we reverse this product. For example, the reverse of

$$2 + e_0 + 3e_1 + 5e_1 e_2 - e_0 e_1 e_2$$

is

$$2 + e_0 + 3e_1 + 5e_2 e_1 - e_2 e_1 e_0.$$

However, it's often useful to keep all the basis vectors in lexicographic ordering, in which case we make sure to use the properties of the geometric product (e.g. $e_i e_j = -e_j e_i$) to reorder the basis

Author's address: Diego Lopez, McGill University, diego.lopez2@mail.mcgill.ca.

2022. XXXX-XXXX/2022/5-ART $15.00
https://doi.org/0000001.0000001

vectors. One of many equivalent definitions of a rotor is that of a multivector R which can be written as a product of an even number of vectors (not necessarily basis vectors) and where $R\tilde{R} = 1$, where $\tilde{R}$ denotes the reverse of R. In $\mathbb{R}^4$, if we expand a rotor and write it in terms of linear combinations of products of basis vectors, it turns out we can always write a rotor as a sum of a scalar, four 2-vector products of basis vectors and one 4-vector product of basis vectors. The stricking usefulness of rotors is that they encode orientation. Indeed, to rotate a vector v by a rotor R, we simply compute the following quantity

$$RvR.$$

However, in $\mathbb{R}^4$, expanding the expression results in 256 terms (a lot of which cancel out). The issue is that finding the exact expression is too long and error-prone to do by hand, and so special care needs to be taken in order to implement this rotation action by rotors on vectors.

2 RELATED WORK

We discuss the work in Bosch [2020]. In it, the equations of motion are written using the language of geometric algebra:

$$x_t = v \qquad R_t = -\frac{1}{2}\omega R$$
$$v_t = F/m \quad L_t(\omega) = \tau$$
$$L_t(\omega) = R\mathcal{I}(\tilde{R}\omega_t R)\tilde{R} - \omega \times R\mathcal{I}(\tilde{R}\omega R)\tilde{R} = \tau,$$

where a subcript with t denotes the time derivative, x and v are vectors (position and velocity), R is a rotor (encoding orientation), ω and τ are 2-vectors (angular velocity and torque) and $\mathcal{I}$ denotes the inertia (represented here as a linear map from 2-vectors to 2-vectors) of the body with respect to the body's frame of reference and the commutator product $\times$ is given by

$$A \times B = \frac{1}{2}(AB - BA)$$

To simplify the problem, we chose to focus on cubes. In this case, the linear mapping $\mathcal{I}$ is simply a scaling map. Thus, the product

$$\omega \times R\mathcal{I}(\tilde{R}\omega R)\tilde{R}$$

vanishes and our equations of motion becomes:

$$L_t(\omega) = R\mathcal{I}(\tilde{R}\omega_t R)\tilde{R} = \lambda\omega_t = \tau$$

for some scalar λ which depends on the angular inertia of the cube.

3 METHODS

The main challenge is to implement 2-vectors, rotors, as well as the rotation action

$$Rv\tilde{R}$$

as well as the product between a 2-vector and a rotor:

$$\omega R$$

which results in another 2-vector. In order to avoid the possibility of making a mistake while deriving these equation by hand, as well as avoid the computationally intensive option of computing the products from the basic properties every time, we opted to write a program which generated the source code for these two products. This program simply expanded each value and added like terms, and outputted a TypeScript source code snippet. Once all the operations are implemented, the equations of motion were solved using a symplectic Euler integrator.

In order for the user to interact and visualize the simulation, we chose to project the tesseract into the xy plane. We also let the user select 6 values to encode a torque 2-vector which is then applied on the cube.

4 CONCLUSIONS

The conclusions for this project are three-fold. A project which utilizes many different kinds of objects benefits from being implemented in a programming language which is statically typed. However, this project was still able to run on the browser, since we chose to implement it in the TypeScript programming language: a superset of JavaScript with optional type annotations which is eventually transpiled into JavaScript. Making sure that types match and operations meant for rotors or 2-vectors are not being performed on the wrong type is very important, and using TypeScript was very useful in avoiding these errors at "transpile time".

Geometric algebra is a useful construct when one wishes to generalize the equations of motion to higher dimensions. However, these constructs also work in three dimensions, and there is some merit to using this instead of alternatives like quaternions and cross products: every construct has a tangible, intuitive geometric meaning. It is much easier to grasp the geometry of rotations by rotors in three dimensions than it is to understand rotations with quaternions. However, this comes at the cost of poor library support and the fact that new mathematical constructs need to be introduced and learned.

Lastly, the project itself can be expanded in many ways:

- rotor rotation works only if the rotor is normalized, so over time the tesseract appears as if it changes slighlty in size, so there should be a way to normalize rotors over time;
- the commutator product $\times$ as well as the action of rotors on 2-vectors need to be implemented so that we can simulate gyroscopic precession in four dimensions;
- the visualization method can be improved, perhaps giving the option to view with perspective projection or view a slide of the 4D simulation instead of the full projection;
- more hyperprisms together with collision detection and resolution could be added.

REFERENCES

M. T. Bosch. 2020. N-Dimensional Rigid Body Dynamics. *ACM Trans. Graph.* 39, 4, Article 55 (July 2020), 6 pages. https://doi.org/10.1145/3386569.3392483

A. Macdonald. 2010. *Linear and Geometric Algebra.* Alan Macdonald. https://books.google.ca/books?id=oxhJYgEACAAJ