N-Dimensional Rigid Body Dynamics

MARC TEN BOSCH, mtb design works, Inc., USA

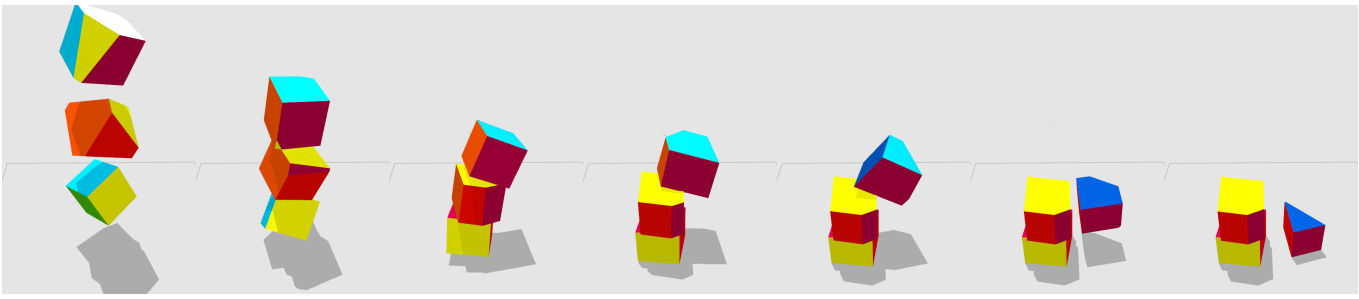


Fig. 1. A stack of three 4D hypercubes

I present a formulation for Rigid Body Dynamics that is independent of the dimension of the space. I describe the state and equations of motion of rigid bodies using geometric algebra. Using collision detection algorithms extended to nD I resolve collisions and contact between bodies. My implementation is 4D, but the techniques described here apply to any number of dimensions. I display these four-dimensional rigid bodies by taking a three-dimensional slice through them. I allow the user to manipulate these bodies in real-time.

CCS Concepts: • **Computing methodologies → Physical simulation**; **Collision detection**; *Virtual reality*.

Additional Key Words and Phrases: Rigid Bodies, N-Dimensional, Physics, Geometric Algebra, Fourth Dimension

ACM Reference Format:
Marc ten Bosch. 2020. N-Dimensional Rigid Body Dynamics. *ACM Trans. Graph.* 39, 4, Article 1 (July 2020), 6 pages. https://doi.org/10.1145/3386569.3392483

1 INTRODUCTION

Our experience of physical space is three dimensional. Consequently, physically-based simulations (physics engines) have so far been focused on and restricted to the two and three-dimensional cases. However using the appropriate formulation of the required equations it is possible to extend them to higher dimensions. Geometric algebra provides a simple dimension-independent formulation. This allows real time manipulation of n-dimensional shapes that collide with each other as if they were real objects, which makes them much less abstract, in stark contrast with most people's experience of them. While attention has been given to understanding and visualizing high-dimensional spaces and other abstract mathematical concepts, it has most often remained limited to visualizing these concepts without any physicality or object-to-object relationships.

Contributions. The contributions of this paper include:

(1) extending the geometric algebra-based formulation of classical 3D rigid body dynamics to nD. By representing geometric algebra operators as matrices, one can in particular construct, diagonalize and transform the inertia tensor for arbitrary nD simplicial meshes for any n, in a simple way. This enables formulating the Euler equation in nD, which allows e.g. studying the case of the 4D Euler equation under torque-free conditions.
(2) computing collision and contact processing in nD, including static and kinetic friction. I give an nD formulation for the Minkowski difference and separating axis theorem collision detection methods based on geometric algebra.
(3) a method of interacting with 4D objects that is akin to our 3D experience of reality.

2 RELATED WORK

Interactive Simulation of 3D Rigid Body Dynamics is a broad field. Bender et al. [2014] provide a survey.

[Cameron 1990] has formulated the 3D continuous collision detection problem as a discrete 4D collision problem by considering the extrusion of each object over time.

Visualizing 4D objects is an interesting and challenging problem of its own, with a long history [Abbott 1884; Banchoff 1990; Chu et al. 2009; Hilbert and Cohn-Vossen 1952]. Many ways of manipulating 4D shapes have been proposed [Yan et al. 2012; Zhang and Hanson 2006].

Geometric algebra operates on a space of elements called multivectors, of which vectors are a subspace. While vectors can be thought of as oriented line-segments, other elements can represent oriented areas (bivectors), volumes (trivectors), and so on. Geometric algebra also defines operations that correspond to translations and rotations of these elements. The book by Macdonald [2011]

Author's address: Marc ten Bosch, mtb design works, Inc., San Francisco, CA, USA, marc@marctenbosch.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0730-0301/2020/7-ART1 $15.00
https://doi.org/10.1145/3386569.3392483

and the course by Gunn and De Keninck [2019] provide introductions. Dorst et al. [2009] provide an introduction to implementing geometric algebra in programs. Doran and Lasenby [2003] apply geometric algebra to the case of 3D rigid bodies.

Cayley [1846] first proposed generalizing the Euler equation to nD. The problem has been studied analytically primarily using matrix analysis, which becomes complex quite quickly (e.g. [Sinclair and Hurtado 2005]).

I chose to use geometric algebra due to its simplicity, generality, and coordinate-free nature, which allows the equations of motions to remain the same as in the 3D case when generalized to nD.

Projective and conformal geometric algebras allow to combine the translational and rotational components into a single equation in a similar way to how homogenous coordinates allow matrices to represent both these components as single transformations. Gunn [2011] uses projective geometric algebra to formulate 2D and 3D rigid body motion in a metric-neutral way that also applies to non-euclidean spaces. In the interest of simplicity I chose not to use this formulation.

3 BACKGROUND

This section briefly reviews geometric algebra and its application to rigid body dynamics in 3D as described by Doran and Lasenby [2003]. However these equations remain the same in nD.

Using geometric algebra one can write equations for rigid body evolution in $n > 1$ dimensions as:

$$x_t = v \quad R_t = -\tfrac{1}{2}\omega R$$
$$v_t = F/m \qquad L_t = \tau$$

where x and R are the position (a vector) and orientation (a rotor), v is the velocity (a vector) and ω is the angular velocity (a bivector), F is the net force (a vector), m is the mass (a scalar), τ is the net torque (a bivector), and L is the angular momentum (a bivector). The t-subscript denotes the time-derivative. The product used for the angular velocity bivector and the orientation rotor is the geometric product, described later.

Angular velocity is represented as a bivector. A bivector is formed from the exterior product of two vectors:

$$B = a \wedge b$$

and has $k = \binom{n}{2}$ coordinates, one for each pair of different orthogonal basis vectors. These components can be stored in memory the same way as for a vector, in a continuous array of k numbers.

If a particle has momentum p and position vector x from some origin, the angular momentum of the particle about the origin is defined as the bivector

$$L = x \wedge p.$$

The usual representation involves a cross product, which is only defined in 3D. In three dimensions, vectors are "dual" to bivectors: roughly speaking, in 3D the space orthogonal to a line (vector) is a plane (bivector). This means that in 3D one can use vectors and bivectors somewhat interchangeably. This has led to the wide use of representing angular velocity as vectors. These "axial" vectors transform differently than regular vectors. Bivectors are a more natural representation because they are straightforwardly transformed to different coordinate systems using geometric algebra, and make the equations work in any number of dimensions.

N-dimensional rotors provide a representation of rotations that is a replacement for quaternions (3D) and complex numbers (2D) that works in any number of dimensions. The rotor that rotates in the plane defined by the vectors a and b by twice the angle between these two vectors is defined by the geometric product of a and b:

$$R = ab = a \cdot b + a \wedge b.$$

It has a scalar term and a bivector term, similar to the real and imaginary terms of complex numbers and quaternions. This kind of sum is called a multivector. The geometric product can be extended to multivectors and hence to rotors themselves. Just like for quaternions, the product of two rotors results in a new rotor that encodes applying the two rotations one after the other. To rotate a vector using a rotor, the following formula is used:

$$x' = Rx\tilde{R}$$

where $\tilde{R}$ denotes the reverse of R, which is similar to the conjugate for quaternions and complex numbers. Products of rotors produce multivectors that are the sum of m-vectors where m is even and less than or equal to n (this forms an algebra called the even sub-algebra). In 3D a general rotor still has only a scalar part and a bivector part (which has 3 components), i.e. the product of two rotors still results in a rotor that can be represented by a single plane of rotation. But in 4D an object can rotate around two independent planes of rotation simultaneously: a general rotor has a scalar part, a bivector part (which has 6 components), and a 4-vector part (which has only one component). I hence store a four-dimensional rotor in memory as a continuous array of 8 numbers.

The instantaneous velocity of a point r on the body, in the reference-frame of the body, is: $v + r \cdot \omega$. The dot product used ($\cdot$) is a generalization of the dot product to multivectors, also sometimes called the left contraction ($\rfloor$).

Angular momentum is related to angular velocity by a linear mapping

$$L(\omega) = \int_V r \wedge (r \cdot \omega)\, \mathrm{d}V$$

taking bivectors to bivectors. Doran and Lasenby [2003] take this integral over the three-dimensional volume of the object.

If a body is rotated by a rotor R its angular momentum can be expressed in terms of a time-independent linear mapping $\mathcal{I}$ computed once in the initial local frame of the body:

$$L(\omega) = R\mathcal{I}(\omega)\tilde{R}. \tag{1}$$

In 3D vectors are dual to bivectors, so the mapping $\mathcal{I}$ from bivectors to bivectors can be turned into a mapping from vectors to vectors by taking the dual from vectors to bivectors, applying the mapping, then taking the dual again to get back to vectors. This mapping can then be represented by a matrix: the inertia tensor.

Differentiating Equation (1) with respect to time, keeping in mind that it is only the rotors that are time dependent gives the familiar form of Euler's equation:

$$L_t(\omega) = \mathcal{I}(\omega_t) - \omega \times \mathcal{I}(\omega) = \tau \tag{2}$$

where $\times$ is not the regular cross product but the commutator product $A \times B = \frac{1}{2}(AB - BA)$, used here for two bivectors A and B. The

quantities in this equation are expressed in the reference-frame of the body (which is rotating with the body).

4 nDIMENSIONAL GENERALIZATION

Representing the inertia tensor as a mapping from vectors to vectors only works in 3D so I do not take the dual.

To represent this bivector mapping, I first define a $k \times n$ matrix $[r]_\star$ such that

$$r \wedge a = [r]_\star a \tag{3}$$

for two vectors r and a. It is a generalization of the cross product matrix $[r]_\times$ such that $r \times a = [r]_\times a$, but I do not take the dual: the range of this transformation is a bivector seen as a vector of dimension k. Also note that

$$r \cdot \omega = [r]_\star^T \omega$$

where, again, the domain of this matrix transformation is a bivector seen as a vector of dimension k.

Using these definitions I define a $k \times k$ matrix I that generalizes the inertia tensor:

$$\begin{aligned} \mathcal{I}(\omega) &= \int_V [r]_\star [r]_\star^T \, \mathrm{d}V \omega \\ &= \int_V \Delta I \, \mathrm{d}V \omega \\ &= I\omega. \end{aligned}$$

The integral is also taken over the n-dimensional volume instead of the three-dimensional volume as would normally be done.

I give $[r]_\star$ and ΔI in 2, 3, and 4 dimensions in appendix A. Note that they are given with respect to lexicographically-ordered bivector basis elements (ex: e_{xy},e_{xz},e_{yz}), but this is an arbitrary choice.

The inertia tensor's range and domain are bivectors and transform as such. Therefore I define a $k \times k$ matrix $[R]_2$ such that for a bivector B seen as a vector, $[R]_2 B = RB\tilde{R}$. Then:

$$I' = [R]_2 I [R]_2^T \tag{4}$$

or, in terms of the mapping itself: $R\mathcal{I}(\tilde{R}\omega R)\tilde{R} = I'\omega$. This lets me express equation (2) in a global inertial reference-frame:

$$L_t(\omega) = I'\omega_t - \omega \times I'\omega = \tau. \tag{5}$$

The formula for the time-derivative of a rotor ($R_t = -\frac{1}{2}\omega R$) is similar to and hence can be substituted for the formula for quaternions that would be used in 3D systems. My implementation uses a (symplectic Euler) time-stepping scheme based on the one by Guendelman et al. [2003] but other time-stepping schemes could be used. I integrate the gyroscopic term of eq. (5) separately using an implicit Euler method [Catto 2015].

As the angular velocity is integrated, small errors can accumulate in the rotor R representing the orientation of the body. In three dimensions, when a quaternion is used instead of a rotor, the effect is corrected by normalizing the quaternion after each time-step. However this is insufficient in 4D and higher: while simple rotors that are formed from the geometric product of two unit vectors do lie on a $(k+1)$-dimensional sphere within $\mathbb{R}^n$, this is not the case for rotors which correspond to double rotations (or more). To correct for small errors I factorize the rotor with respect to the geometric product to get a set of vectors whose product give the original rotor using the algorithm by Perwass [2009]. The algorithm returns normalized vectors, guarantying that the rotor re-formed from their product will represent a proper rotation. The gyroscopic term in particular often generates double rotations which cannot be corrected simply by normalizing.

5 INERTIA TENSOR OF ARBITRARY nD SIMPLICIAL MESH

A nD simplicial mesh is built from multiple $(n-1)$-simplices, ex: a 3D simplicial mesh is built from multiple triangles, a 4D simplicial mesh is built from multiple tetrahedra, and so on.

Due to the linearity of the inertia tensor, to compute the inertia tensor of an arbitrary n-mesh in any dimension it suffices to sum the inertia tensors of n-simplices built from each $(n-1)$-simplex and the origin of the reference frame.

It is possible to compute the inertia tensor of an arbitrary n-simplex directly from its vertices by evaluating the products of inertia integrals:

$$P_{jk} = \int_V jk\rho \, \mathrm{d}V$$

where j and k are coordinates axes. These integrals can be simplified using Gauss' theorem [Dobrovolskis 1996]. However, generalizing the method of Blow and Binstock [2004] proves to be simpler. They first compute the covariance matrix of the body, then transform the covariance matrix into the inertia tensor. Given a matrix M that transforms the vertices of a canonical simplex into those of an arbitrary simplex, the covariance matrix C' of the arbitrary simplex is related to that of the canonical simplex C such that $C' = det(M)MCM^T$. The covariance matrix of a canonical n-simplex has two terms of interest C_{xx} and C_{xy}. They are the integral of x^2 and xy over the hypervolume of the n-simplex aligned with the coordinate axes.

$$\begin{aligned} C_{xx} &= \frac{2}{(n+2)!}, \\ C_{xy} &= \frac{1}{(n+2)!}. \end{aligned}$$

To transform the covariance matrix into the inertia tensor in nD, I take inspiration from Trenkler [2001] and expand $[r]_\star$ in the coordinate basis:

$$[r]_\star = \sum_{i=0}^{n-1} r_i [e_i]_\star.$$

The inertia tensor can then be derived from the covariance matrix C:

$$\begin{aligned} [r]_\star [r]_\star^T &= (\sum_{i=0}^{n-1} r_i [e_i]_\star)(\sum_{j=0}^{n-1} r_j [e_j]_\star^T) \\ &= \sum_{i,j}^{n-1} r_i r_j [e_i]_\star [e_j]_\star^T \\ \int_V [r]_\star [r]_\star^T \, \mathrm{d}V &= \sum_{i,j}^{n-1} C_{ij} [e_i]_\star [e_j]_\star^T. \end{aligned}$$

It is also needed to compute the mass of the n-simplex, which can be computed from its density and its volume $\frac{1}{n!}det(M)$.

It is usual and useful to chose a body's local coordinate-frame such that its inertia tensor I is diagonal. A traditional matrix diagonalization algorithm would produce a rotation matrix, but this $(k \times k)$ matrix would apply only to bivectors (like in eq. (4)) instead of vectors, and so would not be useful as is. However, note that if the $n \times n$ covariance matrix is diagonal, then the inertia tensor will be as well. Therefore before computing the inertia tensor I first rotate the body using the rotation matrix that diagonalizes the covariance matrix.

6 COLLISION RESOLUTION

Using the matrix from (3) I generalize the method of collisions, including static and kinetic friction, of Guendelman et al. [2003], but the scheme is general enough to be applied to other methods. I also use their Contact Graph and Shock Propagation schemes.

After a collision I apply equal and opposite impulses j to each body to obtain $v_0 = v \pm j/m$ and $\omega_0 = \omega \pm I^{-1}(r \wedge j) = \omega \pm I^{-1}[r]_\star j$ where r points from their respective centers of mass to the collision location. The new velocities at the point of collision will be $u_0 = u \pm Kj$ where $K = \delta/m + [r]_\star^T I^{-1}[r]_\star$ with δ the $n \times n$ identity matrix.

Using this formulation static and kinetic friction can be expressed the same way as in three dimensions.

7 COLLISION DETECTION

7.1 Hyperspheres with Polytope or Cylinder

To determine if a $(n-1)$-sphere is colliding with an n-polytope or n-cylinder I generalize the sphere/convex-object collision detection algorithm which is based on the Minkowski Difference (for example: [Ericson 2004]). It finds the closest point on the surface of the object to the center of the hypersphere and checks that it is less than one hypersphere radius away. Note that the closest point may be on a $(n-1)$-cell, or on the cell's boundary, which it shares with one or more cells — the dimension of the boundary depends on the number of cells that share it (for example in 4D two 3D cells share a face, three cells share an edge, and so on). The projection operator $a_{\|} = (a \cdot B)/B$ of a vector a onto a subspace represented by the n-vector B is used to find the closest point.

7.2 Two arbitrary convex polytopes

For two arbitrary convex polytopes I generalize the separating axis theorem collision detection method [Gottschalk 1996] to nD. A n-polytope a contains a certain number of m-cells, where $m < n$. Each m-cell i is spanned by m vectors. The exterior product of these m vectors is the m-vector $V_a^m(i)$. Given two n-polytopes a and b, for each sets of cells such that $m_a + m_b = n - 1$, form all the exterior products:

$$V_a^{m_a}(i) \wedge V_b^{m_b}(j) \quad \forall i, j.$$

The potential separating axes are the dual vectors to the resulting $(n-1)$-vectors. Note that this contains the case of the empty exterior product where $m_a = n-1$ and $m_b = 0$. These are simply the normals to the $(n-1)$-cells (the faces in 3D) of each polytope.

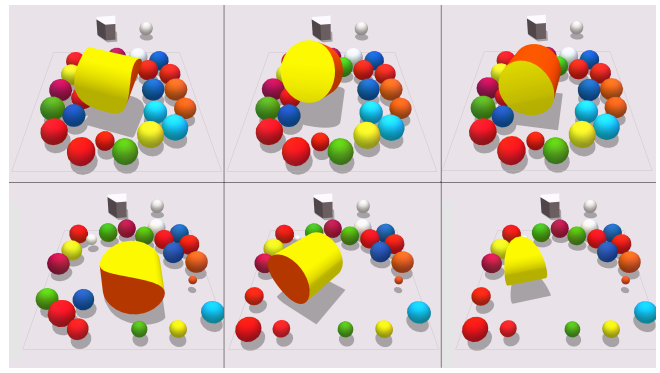


Fig. 2. A user rolls a duocylinder on top of a layer of hyperspheres. Some of them appear to be floating because they are offset in the fourth dimension from our view.

In 3D this formulation is equivalent to taking the cross product (i.e. the dual of the exterior product of two 3D vectors) of the edge direction vectors as potential separating axes. This corresponds to edge/edge collisions. In 4D one instead needs to take the dual of the exterior product of all the edge direction vectors of polytope a with the face bivectors of polytope b and inversely. This corresponds to the case of 1D edge / 2D face collisions. The 1D edge / 1D edge case is subsumed by this case, similarly to how the vertex case is subsumed by the line case in 3D [Ericson 2004].

I implement the specific case of Oriented Boxes by extending the method of Eberly [2002] to nD. All scalar triple products are replaced by exterior products. A 4D hypercube has 8 cells, 24 faces, 32 edges and 16 vertices, but because it is symmetric it has only 4 unique vectors (normals or edge directions) and one can only form 6 unique bivectors. Therefore one needs to check 4×2 vectors coming from cell normals and $4 \times 6 \times 2 = 48$ vectors coming from exterior products.

8 INTERACTIVE SIMULATION AND DISPLAY

I display a three-dimensional slice of the interactive four-dimensional objects. These 4D objects have a three-dimensional surface, represented by a 4D mesh made of simplices (in this case tetrahedra). At any moment only the intersection between a single 3D plane and the 4D objects is visible. This is done by slicing the 3D surface of each object to get a 2D surface to display, in a manner similar to the method by Chu et al. [2009], but taking only a single slice. Slicing each tetrahedron gives a 2D polygon (either a triangle or a quad), and all these polygons together make up the 2D surface to display.

The user can interact with the objects using either a mouse, touch screen or VR controller. They can tap, drag and rotate objects only within the confines of their current 3D slice, but obviously the 4D objects are not confined to that space, and so might disappear out of view. In that case, the user can drag a slider to move the 3D slice along the fourth axis to find them again. I decided on this display and interaction method because it lets users play with the 4D objects in the usual 3D physical environment they are familiar with, while still allowing them to find the objects in a very simple way if they disappear.

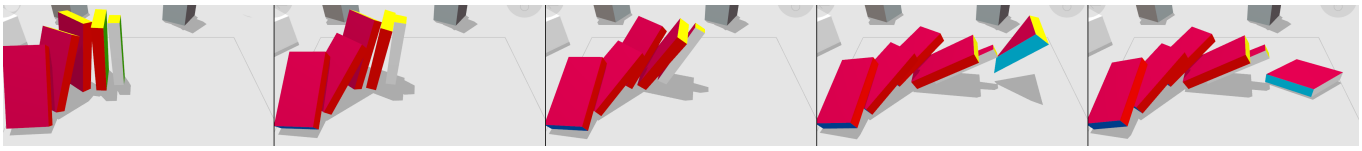


Fig. 3. A users knocks down a stack of hypercubic dominoes. The last dominoes are initially out of view, offset in the fourth dimension.

For the purpose of this simulation, gravity trivially generalizes to nD: it points along the normal vector to the ground, towards the ground. Air friction also has the same form as in the 3D case.

On touchscreen devices such as phones an accelerometer is usually available and is used to control the direction of gravity. Initially the direction of gravity is restricted to the visible 3D slice, but a button can be pressed to let the gravity vector rotate to face slightly along the fourth dimension.

Some VR controllers support haptic feedback, and when a 4D object held by the user hits another 4D object the program slightly vibrates the controller holding the object.

Any position that the user can see corresponds to a position in 4D, and so the simulation can apply forces or impulses at these positions to move the objects to follow the motion of the user. To rotate an object it is only needed to find the 2D plane of rotation in order to form a bivector to apply as torque. In the most general case of rotating one 4D frame to another, I first find a rotor that performs this transformation, then extract its bivector part. To find this rotor I apply successive rotors to transform corresponding pairs of coordinate axes into each other.

Examples of 4D situations and user interactions can be seen in the accompanying video and in figures 1, 2, and 3. Commercial software is also available [ten Bosch 2017].

9 4D DZHANIBEKOV EFFECT

In this section I briefly look at the case of the 4D Euler equation (5) in torque-free conditions. In 3D a consequence of this equation is the Dzhanibekov effect, first described by Poinsot [1851]. Given a body with three unique moments of inertia, each moment of inertia corresponds to a specific plane of rotation. Rotations in the planes with lowest and highest moments of inertia are stable to small perturbations in angular momentum, while for rotations in the intermediate plane a small perturbation gets amplified to a larger, periodic rotation.

In 4D the ω_{ij} component of the equation is:

$$I_{ij}\dot{\omega}_{ij} = (I_{jk} - I_{ik})\omega_{ik}\omega_{jk} + (I_{jm} - I_{im})\omega_{im}\omega_{jm}$$

where k and m are the other two indices besides i and j. Note the presence of a sum in this equation — unlike in 3D the conditions for 4D rotational stability around each plane cannot be definitely stated solely based on the body having distinct principal moments of inertia.

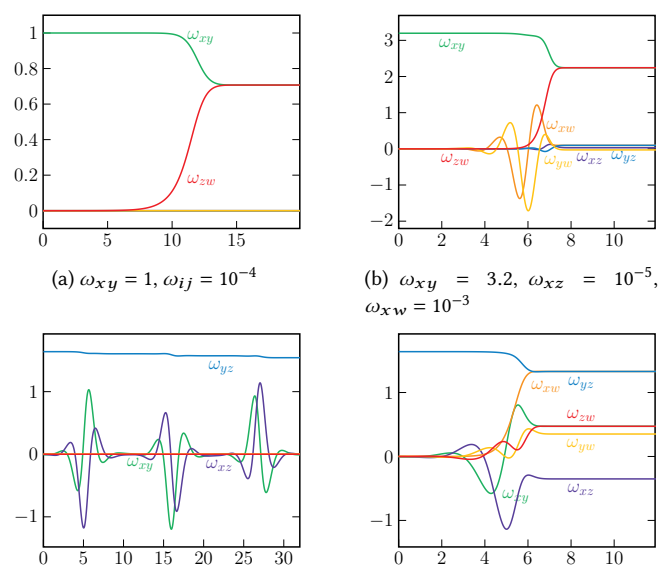
The equation for plane (ij) depends on all the other planes of rotation (ik,jk,im,jm) except for the plane perpendicular to it (km). This is expected, as 4D objects can rotate in two perpendicular planes independently.

The equations reduce to the 3D case if the initial angular velocities lie within a 3D subspace (i.e. if $\omega_{ic} = \omega_{ci} = 0$ for any specific c): figure 5c shows the usual unstable rotation when rotating around an intermediate plane. The small decrease in angular momentum over long time scales is due to the implicit integration method.

Double rotations appear to be stable states: figure 5a shows how a rotation around the (xy) plane and small perturbations around the other planes introduces the dual rotation (zw). The moments of inertia seem to have little effect on the behavior, a similar graph is generated for cuboids of size $(\frac{1}{2}, 1, \frac{3}{2}, 2)$, $(\frac{1}{2}, \frac{1}{2}, 3, \frac{1}{2})$ and others. Even though rotation around the dual plane is independent (perturbing only the (zw) plane does not affect the (xy) plane), the equations are coupled through the other planes, which is what creates the effect.

Figure 5b shows the effect removing perturbations from the initial state of figure 5a. Additional oscillations are created but the system still ends up in a stable state.

Figure 5d and 4 show the effect of adding an additional perturbation to the initial state of figure 5c. The effect is similar to the previous case.



(a) $\omega_{xy} = 1$, $\omega_{ij} = 10^{-4}$

(b) $\omega_{xy} = 3.2$, $\omega_{xz} = 10^{-5}$, $\omega_{xw} = 10^{-3}$

(c) $\omega_{xy} = -0.008$, $\omega_{yz} = 1.63$

(d) $\omega_{xy} = -0.008$, $\omega_{yz} = 1.63$, $\omega_{yw} = 0.0018$

Fig. 5. Angular velocity over time (in seconds) for a 4D cuboid of size $(1.1, 0.3, 2.5, 0.7)$ in various initial conditions.

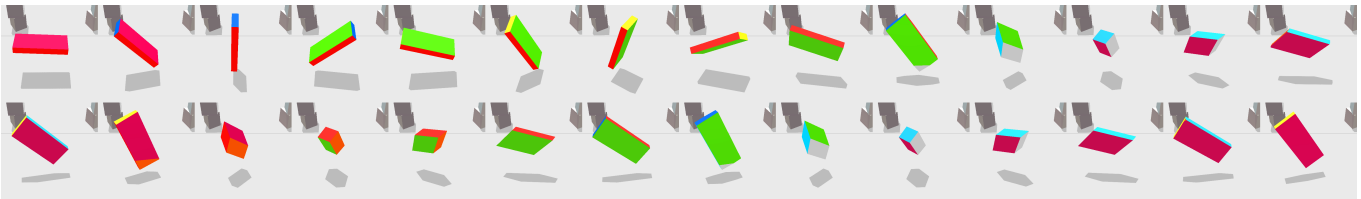


Fig. 4. A hypercuboid rotating under gyroscopic forces.

10 FUTURE WORK

The formulation of rigid body dynamics could be extended to non-euclidean spaces, such as hyperbolic space. Other methods for collision detection, soft bodies, as well as other forces and constraints could be generalized to higher dimensions. Concerning the interaction method, it would be interesting to allow users to manipulate 4D objects outside of their 3D visible slice.

A BIVECTOR OPERATOR AND INERTIA TENSOR MATRICES

2D case:

$$[r]_\star = \begin{pmatrix} -y & x \end{pmatrix}$$

$$\Delta I = \begin{pmatrix} x^2 + y^2 \end{pmatrix}$$

3D case:

$$[r]_\star = \begin{pmatrix} -y & x & 0 \\ -z & 0 & x \\ 0 & -z & y \end{pmatrix}$$

$$\Delta I = \begin{pmatrix} x^2 + y^2 & yz & -xz \\ yz & x^2 + z^2 & xy \\ -xz & xy & y^2 + z^2 \end{pmatrix}$$

4D case:

$$[r]_\star = \begin{pmatrix} -y & x & 0 & 0 \\ -z & 0 & x & 0 \\ -w & 0 & 0 & x \\ 0 & -z & y & 0 \\ 0 & -w & 0 & y \\ 0 & 0 & -w & z \end{pmatrix}$$

$$\Delta I = \begin{pmatrix} x^2 + y^2 & yz & yw & -xz & -xw & 0 \\ yz & x^2 + z^2 & zw & xy & 0 & -xw \\ yw & zw & w^2 + x^2 & 0 & xy & xz \\ -xz & xy & 0 & y^2 + z^2 & zw & -yw \\ -xw & 0 & xy & zw & w^2 + y^2 & yz \\ 0 & -wx & xz & -yw & yz & w^2 + z^2 \end{pmatrix}$$

REFERENCES

E.A. Abbott. 1884. *Flatland: A Romance of Many Dimensions.* Seeley & Co.

T. Banchoff. 1990. *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions.* Scientific American Library. https://books.google.com/books?id=8n55QgAACAAJ

Jan Bender, Kenny Erleben, and Jeff Trinkle. 2014. Interactive simulation of rigid body dynamics in computer graphics. *Computer Graphics Forum (Print)* 33, 1 (2014), 246–270. https://doi.org/10.1111/cgf.12272

Jonathan Blow and Atman J Binstock. 2004. How to find the inertia tensor (or other mass properties) of a 3D solid body represented by a triangle mesh. (2004). http://number-none.com/blow/inertia/

Stephen Cameron. 1990. Collision detection by four-dimensional intersection testing. *Robotics and Automation, IEEE Transactions on* 6 (07 1990), 291 – 302. https://doi.org/10.1109/70.56661

Erin Catto. 2015. Physics for Game Programmers : Numerical Methods. (2015). https://www.gdcvault.com/play/1022197/Physics-for-Game-Programmers-Numerical Presentation at Game Developers Conference 2015.

Arthur Cayley. 1846. Sur quelques propriétés des déterminants gauches. *Journal für die reine und angewandte Mathematik* 1846, 32 (1846), 119–123.

Alan Chu, Chi-Wing Fu, Andrew J. Hanson, and Pheng-Ann Heng. 2009. GL4D: A GPU-based Architecture for Interactive 4D Visualization. *IEEE Transactions on Visualization and Computer Graphics (IEEE Visualization 2009)* 15, 6 (Nov.-Dec. 2009), 1587–1594.

Anthony R Dobrovolskis. 1996. Inertia of any polyhedron. *Icarus* 124, 2 (1996), 698–704.

Chris Doran and Anthony Lasenby. 2003. *Geometric algebra for physicists.* Cambridge University Press.

L. Dorst, D. Fontijne, and S. Mann. 2009. *Geometric Algebra for Computer Science: An Object-oriented Approach to Geometry.* Elsevier. https://books.google.com/books?id=yaEqlAEACAAJ

David Eberly. 2002. Dynamic collision detection using oriented bounding boxes. *Geometric Tools, Inc* (2002).

Christer Ericson. 2004. *Real-Time Collision Detection.* CRC Press, Inc., USA.

Stefan Gottschalk. 1996. Separating axis theorem, Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill. (1996).

Eran Guendelman, Robert Bridson, and Ronald Fedkiw. 2003. Nonconvex rigid bodies with stacking. In *ACM SIGGRAPH 2003 Papers* (San Diego, California) *(SIGGRAPH '03).* ACM, ACM, New York, NY, USA, 871–878. https://doi.org/10.1145/1201775.882358

Charles Gunn. 2011. *Geometry, Kinematics, and Rigid Body Mechanics in Cayley-Klein Geometries.* Doctoral Thesis. Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Berlin. https://doi.org/10.14279/depositonce-3058

Charles G. Gunn and Steven De Keninck. 2019. Geometric Algebra and Computer Graphics. In *ACM SIGGRAPH 2019 Courses* (Los Angeles, California) *(SIGGRAPH '19).* Association for Computing Machinery, New York, NY, USA, Article 12, 140 pages. https://doi.org/10.1145/3305366.3328099

D. Hilbert and S. Cohn-Vossen. 1952. *Anschauliche Geometrie.* Chelsea Publishing Company. https://books.google.com/books?id=d6sBd9h1HbMC

A. Macdonald. 2011. *Linear and Geometric Algebra.* Alan Macdonald. https://books.google.com/books?id=oxhJYgEACAAJ

Christian Perwass. 2009. *Geometric Algebra with Applications in Engineering* (1st ed.). Springer Publishing Company, Incorporated.

Louis Poinsot. 1851. *Théorie nouvelle de la rotation des corps.* Bachelier.

Andrew J Sinclair and John E Hurtado. 2005. Cayley kinematics and the Cayley form of dynamic equations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 461, 2055 (2005), 761–781.

Marc ten Bosch. 2017. 4D Toys. https://4dtoys.com

Götz Trenkler. 2001. The vector cross product from an algebraic point of view. *Discuss. Math. - General Algebra and Applications* 21, 1 (2001), 67–82.

Xiaoqi Yan, Chi-Wing Fu, and Andrew J. Hanson. 2012. Multitouching the Fourth Dimension. *Computer* 45, 9 (2012), 80–88. https://doi.org/10.1109/MC.2012.77

Hui Zhang and Andrew J. Hanson. 2006. *Physically Interacting with Four Dimensions.* Springer Berlin Heidelberg, Berlin, Heidelberg, 232–242. https://doi.org/10.1007/11919476_24