Chapter 17

Visualization and Rigid Body Dynamics

9 February 2011

Adapted from *An Introduction to Computer Simulation Methods* by Harvey Gould, Jan Tobochnik, and Wolfgang Christian

We study affine transformations in order to visualize objects in three dimensions. We then solve Euler's equation of motion for rigid body dynamics using the quaternion representation of rotations.

17.1 Two-Dimensional Transformations

Physicists frequently use transformations to convert from one system of coordinates to another. A very common transformation is an *affine transformation*, which has the ability to rotate, scale, stretch, skew, and translate an object. Such a transformation maps straight lines to straight lines. They often are represented using matrices and are manipulated using the tools of linear algebra such as matrix multiplication and matrix inversion. Because linear algebra and affine transformations are used extensively in imaging and drawing APIs, we begin our study of two- and three-dimensional visualization techniques by studying the properties of transformations.

It is straightforward to rotate a point (x, y) about the origin by an angle θ or scale the distance from the origin by (s_x, s_y) using matrices:

$$\begin{bmatrix} x\prime \\ y\prime \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{17.1}$$

$$\begin{bmatrix} x\prime \\ y\prime \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{17.2}$$

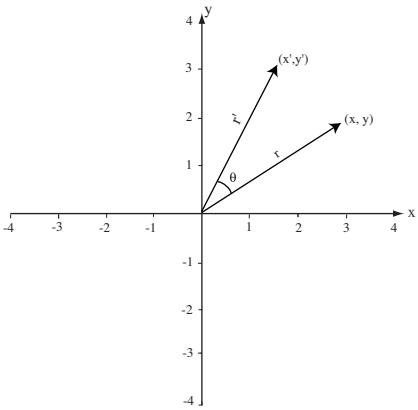

Figure 17.1: A two-dimensional rotation of a point (x, y) produces a point with new coordinates (x', y').

Performing several transformations corresponds to multiplying matrices. However, the translation of the point (x, y) by (d_x, d_y) is treated as an addition and not as a multiplication and must be written differently:

$$\begin{bmatrix} x\prime \\ y\prime \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}. \tag{17.3}$$

This inconsistency in the type of mathematical operation is easily overcome if points are expressed in terms of *homogeneous coordinates* by adding a third coordinate w. Homogeneous coordinates are used extensively in computer graphics to treat all transformations consistently. Instead of representing a point in two dimensions by a pair of numbers (x, y), each point is represented by a triple (x, y, w). Because two homogeneous coordinates represent the same point if one is a multiple of the other, we usually *homogenize* the point by dividing the x-y coordinates by w and write the coordinates in the form $(x, y, 1)$. (The w coordinate can be used to add perspective (see Foley et al.).) By using homogeneous coordinates, an arbitrary affine transformation can be written as:

$$\begin{bmatrix} x\prime \\ y\prime \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{17.4}$$

A translation, for example, can be expressed as:

$$\begin{bmatrix} x\prime \\ y\prime \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{17.5}$$

Exercise 17.1. Homogeneous coordinates

a. How are the rotation and scaling transformations expressed in matrix notation using homogeneous coordinates? Sketch the transformation matrices for a 30° clockwise rotation and for a scaling along the x-axis by a factor of two and then write the transformation matrices. Do these matrices commute?

b. Describe the effect of the affine transformation:

$$\begin{bmatrix} 1 & 0.2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{17.6}$$

■

Exercise 17.1 shows that a coordinate transformation can be broken into parts using a *block matrix* format.

$$\begin{bmatrix} \mathcal{A} & \mathbf{d}^T \\ \mathbf{0} & 1 \end{bmatrix}. \tag{17.7}$$

We will use boldface for row vectors such as $\mathbf{0}$ and $\mathbf{d}$ and calligraphic symbols to represent matrices. The translation vector $\mathbf{d}$ is transposed to convert it to a column vector. The upper left-hand submatrix $\mathcal{A}$ produces rotation and scaling while the vector $\mathbf{d} = [d_x, d_y]$ produces translation.

Homogeneous coordinates have another advantage because they can be used to distinguish between points and vectors. Unlike points, vectors should remain invariant under translation. To transform vectors using the same transformation matrices that we use to transform points, we set w to zero, thereby removing the effect of the last column. Note that the difference between two homogeneous points produces a w equal to zero. The elimination of the effect of translation makes sense because the difference between two points is a displacement vector, and vectors are defined in terms of their components, not their location.

The Affine Transformations in Two Dimensions model in this chapter's source directory shows how to define objects using homogeneous coordinates and how to manipulate them using affine transformations represented by matrices. The model transforms a polygon with n vertices and an arrow. Each polygon vertex transforms as a 3D point located at (x, y) and is stored using homogeneous 2D-coordinates as a `double[] {x,y,1}` array. The arrow components (v_x, v_y) are stored in homogeneous coordinates as a `double[] {vx, vy, 0}` array. Because the arrow represents a vector, it can change direction but remains invariant under translation.

Exercise 17.2. Two-dimensional affine transformations

a. Test the Affine Transformations in Two Dimensions model by applying a transformation to produce a 30° clockwise rotation. About what point does the polygon rotate?

b. The polygon can be rotated about a vertex by first translating the object before performing the rotation. Perform these operations. Although the same transformation is applied to both the polygon and the arrow, the arrow does not translate. Why?

c. Modify the model to apply a skew transformation when a button is pressed.

d. Affine transformations have the property that transformed parallel lines remain parallel. Demonstrate that this property is plausible by transforming simple polygons using arbitrary values for the transformation matrix. Are there restrictions on the transformation matrix?

■

The Affine Transformation in Two Dimensions model uses matrix multiplication to directly transform model data. A useful feature of *EJS* 2D elements is that the third array element is ignored when setting the element's 2D position property so homogeneous coordinate arrays can be used for computation.

Polygon and arrow elements can also be oriented using the Transform property field in their inspector panels. An integer value Transform property will rotate a 2D Element by the given number of degrees and a double value will rotate the element by the given number of radians. The position property fields translate a 2D Element. Behind the scene *EJS* converts these property values into affine transformations just as we have done in this section.

17.2 Three-Dimensional Transformations

There are several available APIs for three-dimensional visualizations using Java. Although Sun has developed the Java 3D package, this package is currently not included in the standard Java runtime distribution. The `gl4java` and `jogl` libraries are also popular because they are based on the Open GL language. Because add-on 3D graphics libraries are in a state of active development and are not always available and because we want our three-dimensional visualizations to work on all Java-enabled computers, *EJS* has a three-dimensional visualization framework that relies only on the standard Java API. *EJS* also supports a second visualization framework that uses the Java 3D package to improve the quality and performance of 3D models. This section describes the mathematics that forms the basis of these libraries.

The simplest rotation is a rotation about one of the coordinate axes with the center of rotation at the origin. This transformation can be written using a 3×3 matrix acting on the coordinates (x, y, z). For example, a rotation about the z axis can be written as:

$$\begin{bmatrix} x\prime \\ y\prime \\ z\prime \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathcal{R}_z \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{17.8}$$

The extension of homogeneous coordinates to three dimensions is straightforward. We add a w-coordinate to the spatial coordinates to create a homogenous point $(x, y, z, 1)$. This point is transformed as:

$$\begin{bmatrix} x\prime \\ y\prime \\ z\prime \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathcal{R}_z & \mathbf{d}^T \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{17.9}$$

The Affine Transformations in Three Dimensions model in this chapter's source directory shows how to define data using homogeneous coordinates and how to manipulate them using affine transformations represented by matrices. The model defines congruent polygons each having n vertices together with n line segments that connect the two polygons. Each polygon vertex transforms as a 3D point located at (x, y, z) and is stored using homogeneous 3D coordinates as a `double[] {x,y,z,1}` array. Data for each n-vertex polygon are stored in a `double[n][4]` array. Because a directed line segment behaves like a vector (v_x, v_y, v_z) and remains invariant under

translation, it is stored using homogeneous 3D coordinates as a `double[] {vx,vy,vz,0}` array. Data for the line segments are stored in a `double[n][4]` array.

A projection transforms an object in a coordinate system of dimension d into another object in a coordinate system less than d. The simplest projection is an *orthographic parallel projection*, which maps an object onto a plane perpendicular to a coordinate axis. For example, if we choose to project along the z-axis, the point (x, y, z) is mapped to the point (x, y) by dropping the third coordinate. A line is projected by projecting the end points and then connecting the projected values. A sphere with radius R is displayed by projecting the center and drawing a circle with the sphere's radius.

The Affine Transformations in Three Dimensions model shows an xy-projection of the 3D scene in a second window. This projection is easy to do because 2D drawing elements interpret the first two values in the position data vector as the (x, y) coordinates and ignore additional array values. Unfortunately, the drawing order in a 2D drawing panel is determined by an object's location in the *EJS* Tree of Elements, not by its spatial location in the 3D line of sight.

Exercise 17.3. Three-dimensional affine transformations

a. Run the Affine Transformation in Three Dimensions example and use the control buttons to apply various transformations to the data. If two rotations are applied sequentially, does the order of the transformations matter? What if two translations are applied sequentially? A translation and a rotation? The fill option allows you to distinguish between the upper and lower polygon and the connecting segments.

b. Use the mouse to rotate the 3D view so that the camera (eye) is located directly above the z-axis looking toward the origin. Are the 3D view and the 2D views similar? What is different? Apply various transformations and note that the 2D view often produces a misleading representation because it does not draw 2D polygons and line segments in the correct order.

■

Problem 17.1. Methane visualization

Develop a CH_4 methane 3D visualization using 2D elements and a 2D drawing panel. Place the carbon atom at the origin and place the hydrogen atoms at the corners of a tetrahedron with C at the center. Use an H-H separation of unity. What is the angle between any two hydrogen bonds? Hint: Place the first hydrogen atom on the z-axis at $(0, 0, 0.75h)$ where $h^2 = 1 - 4\cos^2(\pi/6)/9$.

■

Matrix multiplication and polygon rendering are at the heart of 3D visualization and computers now have specialized hardware to to perform these operations. For example, the processor chip on a high-end graphics cards has many hundreds of graphics processing units (GPUs) each of which can process tens of thousands of matrix multiplications per second. Although the standard Java distribution uses some of this graphics processing power for rendering 2D images, the standard distribution does not support a high-performance 3D library. *EJS* models can use this graphics processing power if the Java 3D library is installed and if the 3D Drawing Panel implementation property is set to J3D. Although *EJS* defaults to the Simple 3D implementation if the Java 3D library is not available, you are encouraged to install Java 3D from the java.net developer site for better performance and more attractive rendering.

17.3 Rotation Representations

Three-dimensional transformations can be described in many different ways including spoken language "Rotate the cube by thirty degrees about the y-axis" but physicists and mathematicians prefer mathematical descriptions such as the matrices described on Section 17.2. Other implementations, such as the Rodrigues angle-axis formula and quaternions, are available in *EJS* and in the *OSP* numerics package and are often used in rigid body problems.

17.3.1 Euler angles

EJS provides rotation Elements on the 3D tools and utilities palette that can be applied to 3D Elements and groups. The simplest rotation elements are x-, y-, and z-axis rotations that have angle and origin property fields. These axis rotations can be combined to produce the well-known Euler angle sequence as shown in the Euler Angle Demonstration model in this chapter's source code directory.

The Euler Angle Demonstration shows a block with coordinate axes labeled $\hat{1}$, $\hat{2}$, and $\hat{3}$ that are aligned with the block sides. This coordinate system is attached to the block and is known as the body frame. The model starts with the body frame aligned with the xyz-Cartesian space frame. The first Euler angle rotation is about the body frame's z-axis by an angle ϕ. This angle is called the angle of precession. The second rotation is about the new x-axis through an angle θ. This angle is called the angle of nutation. The final rotation is about the new z-axis through an angle ψ. This angle is called the spin angle. Multiplying the three rotation matrices gives the final transformation matrix in terms of Euler angles.

$$R_z(\psi)R_x(\theta)R_z(\phi) =$$

$$\begin{bmatrix} \cos\psi\cos\phi - \cos\theta\sin\phi\sin\psi & \cos\psi\sin\phi + \cos\theta\cos\phi\sin\psi & \sin\theta\sin\psi \\ -\sin\psi\cos\phi - \cos\theta\sin\phi\cos\psi & -\sin\psi\sin\phi + \cos\theta\cos\phi\cos\psi & \sin\theta\cos\psi \\ \sin\psi\sin\theta & -\cos\psi\sin\theta & \cos\theta \end{bmatrix} \tag{17.10}$$

Euler angle notation is not unique but our notation is used in many physics textbooks to express the approximate analytical solution to the rapidly spinning gyroscope model. (See Goldstein.) Unfortunately, the matrix in (17.10) is singular when $\sin\theta = 0$. Because we must invert the above matrix to solve problems in rigid body dynamics, differential equations based on the the Euler-angle representation become unstable whenever θ approaches 0 or π. A better approach for numerical computation is to abandon Euler angles and to use the *quaternion* representation described below.

Exercise 17.4. Euler angle rotation

Add a second 3D view to the The Euler Angle Demonstration to display the block with its body axes. Orient the block using a rotation matrix computed from Euler angles (17.10) and show that its orientation matches the view obtained using *EJS* rotation elements. Store the rotation matrix elements in a `double[9]` array and use the array's variable name as the Transform property in the element's inspector to perform the transformation.

■

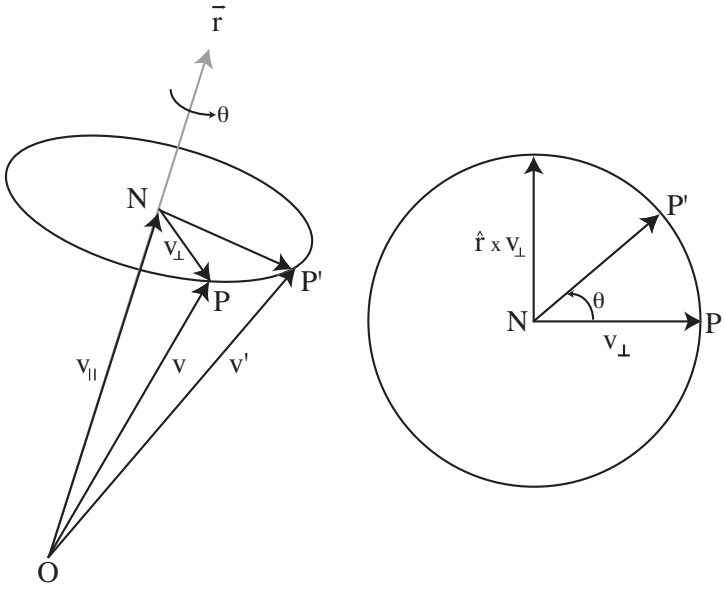

Figure 17.2: A rotation of the vector $\mathbf{v}$ about an axis $\hat{\mathbf{r}}$ to produce the vector $\mathbf{v}'$ can be decomposed into parallel $\mathbf{v}_\parallel$ and perpendicular $\mathbf{v}_\perp$ components.

Problem 17.2. Gyroscope analytic solution

Develop a spinning top (gyroscope) visualization using the Euler angle analytic solution found in mechanics textbooks. Input fields should allow the user to enter initial values for the spin and precession rates. ■

17.3.2 Rodrigues formula

Although rotations about one of the coordinate axes are easy to derive and can be combined using the rules of linear algebra to produce an arbitrary orientation, the general case of rotation about the origin by an angle θ around an arbitrary axis $\hat{\mathbf{r}}$ can be constructed directly. The strategy is to decompose the vector $\mathbf{v}$ into components that are parallel and perpendicular to the direction $\hat{\mathbf{r}}$ as shown in Figure 17.2. The parallel part $\mathbf{v}_\parallel$ does not change while the perpendicular part $\mathbf{v}_\perp$ is a two-dimensional rotation in a plane perpendicular to $\hat{\mathbf{r}}$. The parallel part is the projection of $\hat{\mathbf{r}}$ onto $\mathbf{v}$,

$$\mathbf{v}_\parallel = (\mathbf{v} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}}, \tag{17.11}$$

and the perpendicular part is what remains of $\hat{\mathbf{v}}$ after we subtract the parallel part:

$$\mathbf{v}_\perp = \mathbf{v} - (\mathbf{v} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}}. \tag{17.12}$$

To calculate the rotation of $\mathbf{v}_\perp$, we need two perpendicular basis vectors in the plane of rotation. If we use $\mathbf{v}_\perp$ as the first basis vector, then we can take the cross product with $\hat{\mathbf{r}}$ to produce a vector $\mathbf{w}$ that is guaranteed to be perpendicular to $\mathbf{v}_\perp$ and $\hat{\mathbf{r}}$:

$$\mathbf{w} = \hat{\mathbf{r}} \times \mathbf{v}_\perp = \hat{\mathbf{r}} \times \mathbf{v}. \tag{17.13}$$

The rotation of $\mathbf{v}_\perp$ is now calculated in terms of this new basis.

$$\mathbf{v}' = \mathcal{R}(\mathbf{v}_\perp) = \cos\theta\,\mathbf{v}_\perp + \sin\theta\,\mathbf{w}. \tag{17.14}$$

The final result is the sum of this rotated vector and the parallel part that does not change:

$$\begin{aligned}\mathcal{R}(\mathbf{v}) &= \mathcal{R}(\mathbf{v}_\perp) + \mathbf{v}_\parallel && (17.15a)\\ &= \cos\theta\,\mathbf{v}_\perp + \sin\theta\,\mathbf{w} + \mathbf{v}_\parallel && (17.15b)\\ &= \cos\theta\,[\mathbf{v} - (\mathbf{v}\cdot\hat{\mathbf{r}})\hat{\mathbf{r}}] + \sin\theta\,(\hat{\mathbf{r}}\times\mathbf{v}) + (\mathbf{v}\cdot\hat{\mathbf{r}})\hat{\mathbf{r}} && (17.15c)\end{aligned}$$

or

$$\mathcal{R}(\mathbf{v}) = [1-\cos\theta](\mathbf{v}\cdot\hat{\mathbf{r}})\hat{\mathbf{r}} + \sin\theta\,(\hat{\mathbf{r}}\times\mathbf{v}) + \cos\theta\,\mathbf{v}. \tag{17.16}$$

Equation (17.16) is known as the *Rodrigues formula* and provides a way of constructing rotation matrices in terms of the direction of the axis of rotation $\hat{\mathbf{r}} = (r_x, r_y, r_z)$, the cosine of the rotation angle $c = \cos\theta$, and the sine of the rotation angle $s = \sin\theta$. If we expand the vector products in (17.16), we obtain the matrix

$$\mathcal{R} = \begin{bmatrix} tr_xr_x + c & tr_xr_y - sr_z & tr_xr_z + sr_y \\ tr_xr_y + sr_z & tr_yr_y + c & tr_yr_z - sr_x \\ tr_xr_z - sr_y & tr_yr_z + sr_x & tr_zr_z + c \end{bmatrix}, \tag{17.17}$$

where $t = 1 - \cos\theta$.

Exercise 17.5. Rodrigues formula

Show that a rotation about the z-axis is consistent with (17.16) and (17.17). That is, define the direction of rotation to be $\hat{\mathbf{r}} = (0, 0, 1)$, and show that both formulas give the same result and that this result is consistent with a two-dimensional rotation in the x-y plane.

Develop and test a model that computes a 3×3 transformation matrix using input fields for the axis and rotation direction. Store the rotation matrix elements in a `double[9]` array and use the array's variable name as the Transform property in the element's inspector to perform the transformation. ■

Exercise 17.6. Inverse matrix

What is the inverse matrix of (17.17)? Hint: What happens physically if you change the sign of the rotation angle. Is the rotation matrix orthogonal? ■

17.3.3 Quaternions

The rotation of an arbitrary two-dimensional vector $\mathbf{A} = a_x\hat{x} + a_y\hat{y}$ by an angle θ can be reformulated using complex numbers as

$$a' = ae^{i\theta} = (a_x + ia_y)(\cos\theta + i\sin\theta), \tag{17.18}$$

where the vector $\mathbf{A}$ is expressed as a complex number $a = a_x + ia_y$ and the real and imaginary components correspond to the vector components. This idea can be extended to three dimensions

using quaternions. A quaternion can be represented in terms of real and *hypercomplex* numbers i, j, and k as

$$\hat{q} = q_0 + iq_1 + jq_2 + kq_3 = (q_0, q_1, q_2, q_3), \tag{17.19}$$

where the hypercomplex numbers obey Hamilton's rules

$$i^2 = j^2 = k^2 = ijk = -1. \tag{17.20}$$

Similar to imaginary numbers, the quaternion conjugate is defined as

$$\hat{q}^* = q_0 - iq_1 - jq_2 - kq_3 = (q_0, -q_1, -q_2, -q_3). \tag{17.21}$$

Unlike imaginary numbers, quaternion multiplication does not commute and obeys the rules:

$$ij = k, \quad jk = i, \quad ki = j, \quad ji = -k, \qquad kj = -i, \quad ik = -j. \tag{17.22}$$

Although it would be a mistake to identify hypercomplex numbers with unit vectors in three-dimensional space (just as it would be a mistake to identify the imaginary number i with the y direction in a two-dimensional space), it is convenient to think of a quaternion as the sum of a scalar q_0 and a vector $\mathbf{q}$

$$\hat{q} = q_0 + \mathbf{q} = (q_0, \mathbf{q}). \tag{17.23}$$

The quaternion is said to be *pure* if the scalar part is zero.

By using the above definitions, the product of two quaternions $\hat{p}$ and $\hat{q}$ can be shown to be

$$\hat{p}\hat{q} = (p_0, \mathbf{p})(q_0, \mathbf{q}) = p_0 q_0 - \mathbf{p} \cdot \mathbf{q} + q_0 \mathbf{p} + p_0 \mathbf{q} + \mathbf{p} \times \mathbf{q}. \tag{17.24}$$

Note that except for the additional cross product term, quaternion multiplication is similar to complex multiplication, $(a_0, a_1)(b_0, b_1) = (a_0 b_0 - a_1 b_1, a_1 b_0 + b_1 a_0)$. The norm squared (length squared) of a quaternion is defined to be $|\hat{q}\hat{q}^*| = q_0 q_0 + q_1 q_1 + q_2 q_2 + q_3 q_3$.

Exercise 17.7. Quaternion rules

Show that Hamilton's rules for hypercomplex numbers in (17.20) lead to (17.24). ■

Quaternions provide another way to implement rotations in *EJS*. It can be shown (see Shoemake) that a rotation through an angle θ about an axis with direction cosines (u_1, u_2, u_3) can be represented as a unit quaternion with components

$$\hat{q} = \cos\frac{\theta}{2} + (iu_1 + ju_2 + ku_3)\sin\frac{\theta}{2}. \tag{17.25}$$

Given a unit quaternion $\hat{q}$ that represents a rotation, how do we apply this rotation to an arbitrary vector $\mathbf{A}$? If we define a pure quaternion using the vector components as quaternion components $\hat{a} = (0, a_x, a_y, a_z)$, it can be shown that

$$\hat{a}' = \hat{q}\hat{a}\hat{q}^*, \tag{17.26}$$

where the resulting quaternion $\hat{a}' = (0, a'_x, a'_y, a'_z)$ contains the components of the rotated vector $\mathbf{A}'$ (see Rapaport).

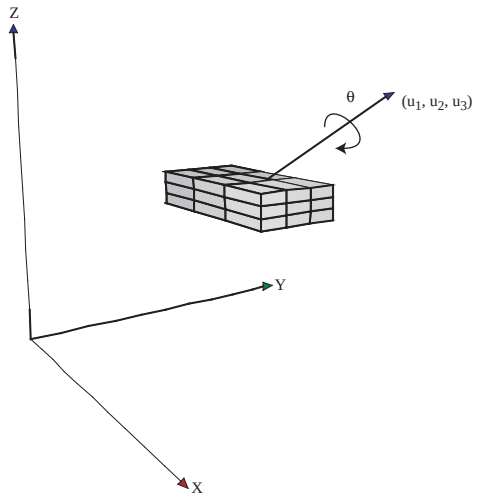

Figure 17.3: The most general change of a rigid body with one point fixed is a rotation about a fixed axis with direction cosines (u_1, u_2, u_3) by the angle θ.

Exercise 17.8. Quaternion properties

a. Use the properties of the direction cosines to show that (17.25) defines a quaternion of unit length.

b. Show that the length of the vector **A** does not change when using (17.26).

■

Because the quaternion representation of rotations does not involve trigonometric functions, it is very efficient for computing rigid body dynamics.

Exercise 17.9. Quaternion rotation

a. Run the Quaternion Rotation model in this chapter's source code directory and enter a quaternion that orients the long side of the block at 45° in the x-y plane. Repeat for the z-y plane.

b. Use a quaternion to orient the long axis of the box along an axis in the $(1, 2, 1)$ direction.

c. What happens in the simulation if the quaternion does not have unit norm? If the quaternion is zero?

■

Exercise 17.10. Quaternion to angle-axis

Given the quaternion $\hat{a} = (q_0, \mathbf{q})$, the rotation axis directions cosines $\mathbf{e}$ and rotation angle θ can be computed by

$$\theta = 2\cos^{-1} q_0 \tag{17.27a}$$
$$\mathbf{e} = \mathbf{q}/|\mathbf{q}| \tag{17.27b}$$

Add an arrow showing the rotation axis and add output fields showing the direction cosines and the rotation angle to the Quaternion Rotation model. ■

If we represent a rotation using a unit quaternion (q_0, q_1, q_2, q_3), carry out (17.26) using (17.24), and express the result in matrix form, the rotation matrix can be expressed using quaternion components as

$$\mathcal{R} = 2\begin{bmatrix} \frac{1}{2} - q_2^2 - q_3^2 & q_1q_2 + q_0q_3 & q_1q_3 - q_0q_2 \\ q_1q_2 - q_0q_3 & \frac{1}{2} - q_1^2 - q_3^2 & q_2q_3 + q_0q_1 \\ q_1q_3 + q_0q_2 & q_2q_3 - q_0q_1 & \frac{1}{2} - q_1^2 - q_2^2 \end{bmatrix}. \tag{17.28}$$

This matrix is equivalent to the rotation matrix derived from the Rodrigues formula (17.16) and can be used to transform vectors to and from the rotating object's body frame.

17.4 Dynamics of a Rigid Body

The dynamical behavior of a rigid body is determined by

$$\frac{d\mathbf{P}}{dt} = \mathbf{F} \tag{17.29a}$$
$$\frac{d\mathbf{L}}{dt} = \mathbf{N}, \tag{17.29b}$$

where the rate of change of the total linear momentum $\mathbf{P}$ and total angular momentum $\mathbf{L}$ about a point O is determined by the total force $\mathbf{F}$ on the body and the total torque $\mathbf{N}$ about O. These momenta are expressed in terms of the translational velocity $\mathbf{V}$ and rotational velocity $\boldsymbol{\omega}$ as

$$\mathbf{P} = M\mathbf{V} \tag{17.30a}$$
$$\mathbf{L} = \mathcal{I}\boldsymbol{\omega}, \tag{17.30b}$$

where M is the mass and $\mathcal{I}$ is the moment of inertia tensor. For an unconstrained body, the point O is usually taken to be the center of mass. For a constrained body, such as a spinning top, the point O is usually taken to be the point of support.

Although the translational and rotational equations of motion appear similar, the fact that the inertia tensor is not always constant with respect to axes fixed in space complicates the analysis. To use a constant inertia tensor, we must describe the motion using a non-inertial reference frame known as the *body frame* that is fixed in the body. Because we are free to orient the axes within the body, we choose axes for which the moment of inertia tensor is diagonal. These axes are referred to as the body's principal axes and are easy to determine for symmetrical objects. The diagonal

body	axis	Moment of Inertia
ellipsoid, axes $(2a, 2b, 2c)$	axes a	$I = m(b^2 + c^2)/5$
	axes b	$I = m(a^2 + c^2)/5$
	axes c	$I = m(a^2 + b^2)/5$
parallelopiped, sides (a, b, c)	perpendicular to (a, b)	$I = m(a^2 + b^2)/12$
	perpendicular to (b, c)	$I = m(b^2 + c^2)/12$
	perpendicular to (c, a)	$I = m(c^2 + a^2)/12$
sphere, radius r	any diameter	$I = m(2/5)r^2$
thin rod, length l	normal to length at center	$I = ml^2/12$
thin circular sheet, radius r	normal to sheet at center	$I = mr^2/2$
	any diameter	$I = mr^2/4$
thin rectangular sheet, sides (a, b)	normal to sheet at center	$I = m(a^2 + b^2)/2$
	parallel to a at center	$I = mb^2/12$
	parallel to b at center	$I = ma^2/12$

Table 17.1: The moment of inertia about the center of mass of various geometric shapes in the *EJS* 3D library. The mass m is assumed to be uniformly distributed.

elements of the moment of inertia tensor are calculated using the volume integrals

$$I_1 = \int_V \rho(y^2 + z^2)\, dV \tag{17.31a}$$

$$I_2 = \int_V \rho(z^2 + x^2)\, dV \tag{17.31b}$$

$$I_3 = \int_V \rho(x^2 + y^2)\, dV, \tag{17.31c}$$

where ρ is the mass density. The off-diagonal elements are zero in the principal axis coordinate system. The moments of inertia for various simple geometrical objects are shown in Table 17.1.

The Rotation About a Fixed Axle model in the source code directory uses (17.30) and (17.31) to show the physics of a block rotating on a fixed shaft with uniform angular velocity ω. The mass can be tilted on the shaft to produce an out-of-balance configuration that causes the system to shake unless a torque is applied to the shaft. The model displays the angular momentum vector and the torque vector using color-coded arrows.

The model's view contains a 3D group with a block, an angular momentum vector, and a z-axis rotation element that tilts the body through an angle α. This body group is within an axle group that rotates about the space frame z-axis. The evolution workpanel increments the axle rotation angle ϕ by $\Delta\phi$ and updates the view with this new value. This view update is very important because the default update sequence recomputes the view after fixed relations are evaluated. Our model invokes 3D element methods within the fixed relations workpanel and these methods depend on the block having the correct orientation.

Open the Rotation About a Fixed Axle model and examine the fixed relation code. The body frame code uses the block's `toBodyFrame` method to transform the space frame angular velocity into the body frame before computing the angular momentum vector using (17.30) *in the body*

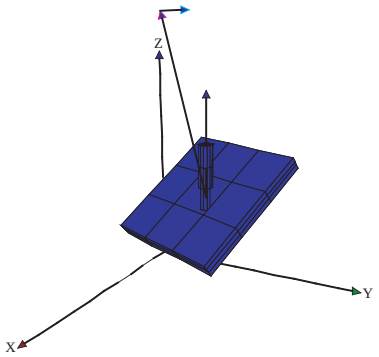

Figure 17.4: The angular momentum of a rotating block.

frame. The ability to transform coordinates between space and body frames makes it easy to solve rigid body dynamics problems but we need to insure that the space and body frame coordinate origins coincide when transforming vectors. Otherwise, we must subtract the coordinate origin translation.

The space frame code in the fixed relations workpanel explicitly shows the matrix algebra need to compute (17.30) in the space frame. The code concatenates (multiplies) two Matrix3DTransformation objects to create the body frame to space frame transformation M. (The Matrix3DTransformation class is defined in the *OSP* numerics library and this class must be imported into the model.) The matrix transformation is then used to transform the moment of inertia tensor from the body frame to the space frame by

$$I_{space} = M I_{body} M^{-1} \tag{17.32}$$

This time-dependent space frame moment of inertia matrix is shown in a dialog window.

Exercise 17.11. Body and space frame transformations

Vector algebra can be performed in any reference frame but the physics must remain the same. Modify the Rotation About a Fixed Axle model to do the following.

a. Show that the magnitude of the space frame and body frame angular momentum are the same.

b. Use the block's `toSpaceFrame` method to transform the body frame angular momentum vector into the space frame and show that the transformed vector components match the space frame vector components. In other words, compute $\mathbf{L} = I\mathbf{w}$ in the space frame.

■

Given that the general relation between the derivative of a vector $\mathbf{A}$ in the space frame to the derivative in the body frame is (see Goldstein)

$$\left(\frac{d\mathbf{A}}{dt}\right)_{\text{space}} = \left(\frac{d\mathbf{A}}{dt}\right)_{\text{body}} + \boldsymbol{\omega} \times \mathbf{A}, \tag{17.33}$$

it is easy to show that the rotational equation of motion in the body frame can be written as:

$$\frac{d\mathbf{L}}{dt} + \boldsymbol{\omega} \times (\mathcal{I}\boldsymbol{\omega}) = \mathbf{N}. \tag{17.34}$$

Equation (17.34) is Euler's equation for the motion of a rigid body. Because the moment of inertia tensor $\mathcal{I}$ is diagonal in the body frame,(17.34) may be written in component form as

$$I_1\dot{\omega}_1 + (I_3 - I_2)\omega_3\omega_2 = N_1 \tag{17.35a}$$
$$I_2\dot{\omega}_2 + (I_1 - I_3)\omega_1\omega_3 = N_2 \tag{17.35b}$$
$$I_3\dot{\omega}_3 + (I_2 - I_1)\omega_2\omega_1 = N_3. \tag{17.35c}$$

Solving Euler's equation of motion requires three orientation variables as well as their rates of change. The orientation often is given in terms of the Euler angles ψ, θ, and ϕ as described in Section 17.3. To use these angles as differential equation state variables in (17.35), we must be able to calculate their rate as a function of the body-frame angular velocity. The expression for the angular velocity in the body frame in terms of the Euler angles is (see Goldstein)

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} = \begin{bmatrix}\sin\theta\sin\psi & \cos\psi & 0\\ \sin\theta\cos\psi & -\sin\psi & 0\\ \cos\theta & 0 & 1\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}. \tag{17.36}$$

Unfortunately, the matrix in (17.36) is singular when $\sin\theta = 0$. Because we must invert this matrix to solve for the rate of the Euler angles, the Euler-angle rate equation becomes unstable when θ approaches 0 or π.

Problem 17.3. Body frame solution

Develop a model to solve Euler's equation of motion for a block with sides a, b, and c in the body frame. That is, solve the first order differential equations (17.35) for $\mathbf{w}(t)$ and display the body frame view of the block with angular velocity and angular momentum vectors. Trace the vector tips to show their trajectories in the body view.

a. Show that the motion is stable if the initial angular velocity is close to the principal axis with largest or smallest moment of inertia.

b. Show that the motion is unstable if the initial angular velocity is close to the 2 axis where $I_1 < I_2 < I_3$.

The fundamental difficulty in using the body frame solution of (17.35) is that the block appears stationary and that we do not know the location of the xyz-axis system.

■

17.5 Quaternion equations of motion

Rapaport shows that angular velocity in the body frame can be written as a quaternion rate $\dot{\hat{q}}(t) = \frac{1}{2}\hat{\omega}(t)\hat{q}(t)$, where $\hat{\omega}$ is a pure quaternion $(0, \omega_1, \omega_2, \omega_3)$. The time dependence of a vector $\mathbf{r}$

can be expressed as a transformation of its initial value $\mathbf{r}_0$ as

$$\hat{r}(t) = \hat{q}(t)\hat{r}_0\hat{q}^*(t). \tag{17.37}$$

If we differentiate $\hat{r}(t)$ with respect to time, we have

$$\dot{\hat{r}} = \dot{\hat{q}}\hat{r}_0\hat{q}^* + \hat{q}\hat{r}_0\dot{\hat{q}}^*, \tag{17.38}$$

where we have dropped the explicit time dependence of $\hat{q}$. We substitute $\hat{r}_0 = \hat{q}^*\hat{r}\hat{q}$, and obtain

$$\dot{\hat{r}} = \dot{\hat{q}}\hat{q}^*\hat{r} + \hat{r}\hat{q}\dot{\hat{q}}^* = \dot{\hat{q}}\hat{q}^*\hat{r} - \hat{r}\dot{\hat{q}}\hat{q}^*, \tag{17.39}$$

where we have used the fact $\hat{q}\hat{q}^* = 1$. The only part of the $\hat{r}$ and $\dot{\hat{q}}\hat{q}^*$ product that does not commute is the vector cross product. The scalar part commutes and is zero. If we denote the pure (vector) part of the quaternion $\dot{\hat{q}}\hat{q}^*$ by $\mathbf{u}$, we find

$$\dot{\mathbf{r}} = \mathbf{u} \times \mathbf{r} - \mathbf{r} \times \mathbf{u} = 2\mathbf{u} \times \mathbf{r}. \tag{17.40}$$

Because $\dot{\mathbf{r}} = \boldsymbol{\omega} \times \mathbf{r}$ for rotational motion, we obtain

$$\boldsymbol{\omega} = 2\mathbf{u}. \tag{17.41}$$

Equation (17.41) can be expressed using components by writing $\boldsymbol{\omega}$ as

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ 0 \end{bmatrix} = 2\mathcal{W} \begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}, \tag{17.42}$$

where

$$\mathcal{W} = \begin{bmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \\ q_0 & q_1 & q_2 & q_3 \end{bmatrix}. \tag{17.43}$$

Because $\mathcal{W}$ is orthogonal, the transpose of (17.43) is its inverse $\mathcal{W}^T\mathcal{W} = \mathbf{1}$, and

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2}\mathcal{W}^T \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ 0 \end{bmatrix}. \tag{17.44}$$

Because the quaternion derivatives depend on both $\hat{q}$ and $\boldsymbol{\omega}$ even in the simplest Euler-like ODE algorithm, solving (17.44) is a two-step process. A modified leap-frog algorithm that advances the angular momentum in the space frame at every time step,

$$\mathbf{L}(t + \Delta t/2) = \mathbf{L}(t - \Delta t/2) + \mathbf{N}(t)\Delta t, \tag{17.45}$$

and then transforms the angular momentum to the body frame to advance the quaternion,

$$\hat{q}(t + \Delta t) = \hat{q}(t) + \dot{\hat{q}}(t + \Delta t/2)\Delta t, \tag{17.46}$$

may be applied to simple systems (see Allen and Tildesley). We use this algorithm to study the dynamics of a spinning plate.

Richard Feynman (see references) noticed that the wobble and spin of a Cornell cafeteria plate as it was tossed into the air was in a 2:1 ratio. Although this result can be derived analytically,[1] we will simulate the dynamics using (17.46) to discover the simple dynamics and to test our simple quaternion-based numerical algorithm.

The Feynman Plate model in the source code directory shows a spinning plate as seen from the space frame. In order minimize numerical error, a small evolution step (17.46) is executed multiples times per display interval using a loop on the Evolution workpanel. Each evolution step executes the following code.

```
//use Element to convert angular momentum vector to body frame
double[] LBody= _view.bodyGroup.toBodyFrame(LSpace.clone());
omega[0]=LBody[0]/I1;
omega[1]=LBody[1]/I2;
omega[2]=LBody[2]/I3;
// compute quaternion rate of change
double q0dot = 0.5*(-q1*omega[0]-q2*omega[1]-q3*omega[2]); // dq0/dt
double q1dot = 0.5*( q0*omega[0]-q3*omega[1]+q2*omega[2]); // dq1/dt
double q2dot = 0.5*( q3*omega[0]+q0*omega[1]-q1*omega[2]); // dq2/dt
double q3dot = 0.5*(-q2*omega[0]+q1*omega[1]+q0*omega[2]); // dq3/dt
// use the quaternion rate to advance in time; delta is time step
q0 += q0dot*delta; q1 += q1dot*delta;
q2 += q2dot*delta; q3 += q3dot*delta;
// renormalize to eliminate drift
double norm = 1/Math.sqrt(q0*q0+q1*q1+q2*q2+q3*q3);
q0 *= norm; q1 *= norm;
q2 *= norm; q3 *= norm;
// update the Element transformation
_view.quaternion3D.setCoordinates(q0,q1,q2,q3);
```

The first statement clones the space frame angular momentum array and transforms its components into the spinning plate's body frame. The body frame angular momentum is then used with the diagonal moment of inertia tensor to compute the angular velocity **w**. Next, the quaternion rate (17.44) is evaluated and the rotation is advanced. Finally, the quaternion is renormalized to eliminate drift and the new values are passed to the rotation element so that the `toBodyFrame` method can again be used.

Problem 17.4. Simulation of a wobbling plate

a. Does the Feynman Plate model confirm the observation of the wobbling plate in the Cornell cafeteria? Are the discrepancies between Feynman's description and the simulation due to the fact that the wobble is not small or due to numerical inaccuracies? Justify your answer.

b. How well does the simple algorithm (17.46) conserve energy and angular momentum?

[1] See Tuleja et al. for a simple and elegant proof of the wobble to spin ratio using basic mechanics and symmetry arguments.

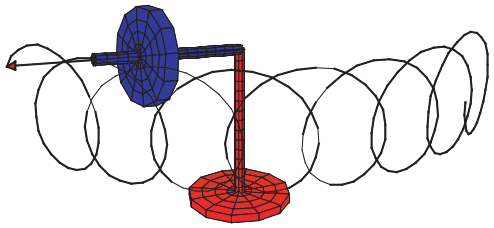

Figure 17.5: The dynamics of a spinning gyroscope.

c. Will a rectangular food tray wobble the same way as a plate?

d. Does the plate wobble the same way if it is spun about an axis close to a diameter rather than close to a line perpendicular to the flat side of the disk?

e. Add a second visualization to the the Feynman Plate model showing the motion as viewed in the non-inertial body frame.

■

Problem 17.5. Torque-free rotation of a symmetrical body

a. Add a plot showing the time dependence of the angular velocity components $\omega_1(t)$, $\omega_2(t)$, and $\omega_3(t)$ to the `FeynmanPlateApp` class.

b. Verify that $\omega_1(t)$ and $\omega_2(t)$ exhibit an out of phase sinusoidal dependence if $I_1 = I_2$.

c. If α denotes the angle between the body-frame $\hat{3}$-axis and the axis of rotation, verify that

$$\Omega = \left(\frac{I_3}{I_s} - 1\right)\omega \cos\alpha, \tag{17.47}$$

where Ω is the angular velocity of the $\boldsymbol{\omega}$ vector's precession about the body frame's $\hat{3}$-axis.

■

17.6 Rigid Body Models

As seen in Problem 17.4, the simple rigid body algorithm presented in Section 17.5 does a good job of conserving energy and angular momentum, but should be improved if our goal is to compute accurate trajectories over long times. To use a higher order differential equation solver, we need to obtain an expression for the second derivative of the quaternion by eliminating the angular velocities from Euler's equation of motion (17.35). The resulting quaternion acceleration can be used in a differential equation solver.

If we start with

$$\frac{d}{dt}\hat{q}^*\dot{\hat{q}} = \dot{\hat{q}}^*\dot{\hat{q}} + \hat{q}^*\ddot{\hat{q}}, \tag{17.48}$$

multiply by $\hat{q}$ and rearrange the terms, we obtain

$$\ddot{\hat{q}} = \hat{q}(\frac{d}{dt}\hat{q}^*\dot{\hat{q}} - \dot{\hat{q}}^*\dot{\hat{q}}). \tag{17.49}$$

Some messy algebra shows that (17.49) can be written in matrix form as

$$\begin{bmatrix} \ddot{q}_0 \\ \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} = \frac{1}{2}\mathcal{W}^T \begin{bmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \\ -2\Sigma\dot{q}_m^2 \end{bmatrix}. \tag{17.50}$$

The dynamical behavior of a rigid body now can be expressed in terms of a quaternion state vector $(q_0, \dot{q_0}, q_1, \dot{q_1}, q_2, \dot{q_2}, q_3, \dot{q_3}, t)$. The following steps summarize the algorithm for computing the quaternion acceleration for the rigid body:

1. Use the state vector to compute the angular velocity $(\omega_1, \omega_2, \omega_3)$ in the body frame using (17.42);
2. Project the external torques onto the body frame and compute $\dot{\boldsymbol{\omega}}$ using Euler's equation (17.35);
3. Compute the quaternion acceleration using (17.50).

These steps are implemented in the Free Rotation and the Gyroscope models in this chapter's source code directory.

The Free Rotation model uses many of the techniques developed in the Feynman Plate model. The big change is its use of an adaptive stepsize ODE solver rather than an explicit numerical algorithm. Although the initialization page sets the quaternion rate using initial angular velocity components, the angular velocity is no longer a dynamical variable and is computed as needed from the quaternion components. The following preliminary code computes the quaternion acceleration that is used by the ODE solver to advance the rotation.

```
// compute omega in body frame
double w1 = 2*(-q[1]*qDot[0]+q[0]*qDot[1]+q[3]*qDot[2]-q[2]*qDot[3]);
double w2 = 2*(-q[2]*qDot[0]-q[3]*qDot[1]+q[0]*qDot[2]+q[1]*qDot[3]);
double w3 = 2*(-q[3]*qDot[0]+q[2]*qDot[1]-q[1]*qDot[2]+q[0]*qDot[3]);
// evaluate Euler's equations of motion in body frame
double w1dot = (I2-I3)*w3*w2/I1;
double w2dot = (I3-I1)*w1*w3/I2;
double w3dot = (I1-I2)*w2*w1/I3;
// compute quaternion acceleration
double[] qAcc=new double[4];
double sum = 0;
for(int i = 0;i<4;i++) {// sum the q dot values
  sum += qDot[i]*qDot[i];
}
sum = -2.0*sum;
qAcc[0] = 0.5*(-q[1]*w1dot-q[2]*w2dot-q[3]*w3dot+q[0]*sum);
```

```
qAcc[1] = 0.5*( q[0]*w1dot-q[3]*w2dot+q[2]*w3dot+q[1]*sum);
qAcc[2] = 0.5*( q[3]*w1dot+q[0]*w2dot-q[1]*w3dot+q[2]*sum);
qAcc[3] = 0.5*(-q[2]*w1dot+q[1]*w2dot+q[0]*w3dot+q[3]*sum);
```

The Free Rotation model main window displays a space frame view showing a constant angular momentum arrow and a body group containing a block and 123-axis arrows. Note how the body group orientation is set using a quaternion in the fixed relation code page and how the methods within the body are used to transform vectors to and from the body frame. The tips of the body frame axis arrows trace space frame trajectories to illustrate stable and unstable rotation of the block about the principal axes. A second window displays the body frame view with space frame xyz-axes orbiting the block.

Exercise 17.12. Free Rotation model

Show that the Free Rotation model gives the same results as the Feynman Plate model used in Problem 17.4. Does the body frame view match the result of Problem 17.3?

■

Problem 17.6. Torque-free rotation about a principal axis

Run the Free Rotation model. If the angular velocity vector coincides with a body's principal axis, the angular momentum and the angular velocity coincide. The body should then rotate steadily about the corresponding principal axis because the net torque is zero and the angular momentum is constant. Does the simulation show this result for all three axes if the moments of inertia are unequal? Perturb the angular velocity. Are rotations about the three principal axes stable or unstable? How do the angular velocity and angular momentum vectors move in the body frame?

Repeat this simulation with a different set of moments of inertia and verify that $\omega_3(t)$ is constant if $I_1 = I_2 = I_3$. ■

Problem 17.7. Free rotation of a thin cylindrical rod

Model the free rotation of a thin cylindrical rod. Show that $\boldsymbol{\omega}$ precesses about the axis of the rod. Why does $\boldsymbol{\omega}$ precess? Why does $\mathbf{L}$ not precess? How is the frequency of precession related to the moment of inertia of the rod?

■

It is easy to add torque to the Free Rotation model to simulate a gyroscope. An ideal gyroscope is a rigid body that spins about an axle while being supported by a frictionless pivot. Because the pivot point is not the center of mass, there is an external torque τ due to the weight Mg of the gyroscope. Because we must compute the torque's vector components in the body frame to use Euler's equation of motion, the ODE preliminary code transforms the gravitational force from the space frame (gravity acts in the z direction) to the body frame before computing the cross product $\tau = \mathbf{r} \times \mathbf{F}$.

The Gyroscope model in the source code directory shows a spinning ellipsoid with radius R and thickness $R/5$. The gyroscope axle is the body's axis of symmetry (the body frame's $\hat{3}$-axis) and the ellipsoid center of mass is located a distance z_0 from the pivot along this axis. The model draws a trail showing the gyroscope's precession but the spinning about the $\hat{3}$-axis may appear

incorrect because the shaft rotates many times between screen updates. This aliasing effect often is seen in movies showing carriage wheel spokes rotating backward to the direction of travel. The model assumes a massless shaft, a total ellipsoid mass M of one, and an acceleration of gravity of one. A second window plots the elevation angle of the axel and the angular momentum vector to allow comparison with analytic approximations.

Problem 17.8. Uniform precession

If the angular velocity about the axis of symmetry is large, there are two two possible rates of steady precession. These precession rates $\dot{\phi}$ are approximately

$$\dot{\phi} \approx \frac{I_3}{I_s} \frac{w_3}{\cos\theta_0}, \tag{17.51a}$$

and

$$\dot{\phi} \approx \frac{mgl}{I_3 \omega_3}, \tag{17.51b}$$

where θ_0 is the angle between the vertical and the axis of gyroscope and mgl is the weight times the distance from the pivot to the center of mass. Demonstrate both precession rates. ■

17.7 Projects

Project 17.1. Rotating reference frames

Model the projectile motion of a ball thrown into the air as seen from a rotating platform. Do so by solving the equations of motion in an inertial reference frame and transforming the trajectory to the non-inertial rotating frame.

Project 17.2. Gyroscope using Euler angles

Develop a gyroscope model using Euler angles rather than quaternions. Compare the model to the quaternion model developed in Section 17.6 and to the approximate analytical solution for large spin. Show that the model can fail for certain initial conditions.

Project 17.3. Falling box

Do a two-dimensional simulation of a rotating and falling box hitting and rebounding from a floor. If you are bold, do the simulation in three dimensions by adding translational center of mass coordinates to the quaternion state vector. Does the average kinetic energy of translation equal the average energy of rotation over many bounces?

References and Suggestions for Further Reading

M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press (1987).

Didier H. Besset, *Object-Oriented Implementation of Numerical Methods*, Morgan Kaufmann (2001).

David H. Eberly, *Game Physics*, Morgan Kaufmann (2004)

Denis J. Evans, "On the representation of orientation space," Mol. Phys. **34** (2) 317–325 (1977).

R. P. Feynman, *Surely You are Joking, Mr. Feynman!*, W. W. Norton (1985). See pp. 157–158 for a discussion of the motion of the motion of the rotating plate. Feynman had fun doing physics.

James D. Foley, Andries van Dam, Steven Feiner, and John Hughes, *Computer Graphics: Principles and Practice*, second edition, Addison-Wesley (1990).

Herbert Goldstein, Charles P. Poole, and John L. Safko, *Classical Mechanics*, third edition, Addison-Wesley (2002). Chapter 4 discusses the kinematics of rigid body motion.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992).

Jerry Marion and Stephen Thornton, *Classical Dynamics*, fifth edition, Brooks/Cole (2003).

D. C. Rapaport, "Molecular dynamics simulation using quaternions," J. Comp. Phys. **60**, 306–314 (1985).

D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, second edition, Cambridge University Press (2004). Chapter 8 discusses the molecular dynamics of rigid molecules.

Philip J. Schneider and David H. Eberly, *Geometric Tools for Computer Graphics*, Morgan Kaufmann (2003)

Ken Shoemake,"Animating rotation with quaternion curves," ACM Transactions in Graphics **19** (3), 256–276 (1994).

Keith R. Symon, *Mechanics*, Addison-Wesley (1971).

Slavomir Tuleja et al., "Feynman's wobbling plate," submitted to Am. J. Phys.

John R. Taylor, *Classical Mechanics*, University Science Books (2005).

James M. Van Verth and Lars M. Bishop, *Essential Mathematics for Games and Interactive Applications*, Morgan Kaufmann (2004).