7.4 Rigid body simulation*

Quaternions vs. Rotation Matrices

- unit quaternions, a better way to represent the orientation of a rigid body. Why?

(1) More compact: 4 numbers vs. 9 numbers

(2) Smooth transition can be achieved by interpolating the quaternions, but difficult with the matrices

(3) Due to cumulation of rounding errors, both quaternions and rotation matrices can cease to be unitary and orthogonal. However, a quaternion can be easily normalized, whereas a rotation matrix is harder to bring back to being orthogonal

(4) Quaternions can avoid gimbal lock

Some materials used here are taken from David Baraff's notes: **Physically Based Modeling - *Rigid Body Simulation***

Quaternions vs. Rotation Matrices

- unit quaternions, a better way to represent the orientation of a rigid body.

- degree of redundancy is noticeably lower for quaternions than rotation matrices (hence, quaternions experience far less drift than rotation matrices)

- a formula for $\dot{R}(t)$

$$\dot{R}(t) = \omega(t)^* R(t)$$

When we integrate this equation we inevitably encounter drift

- drift problem can be easily corrected by renormalizing the quaternion to unit length

- Quaternions revisited:
    - quaternion definition
    - quaternion multiplication
    - rotation

Quaternions vs. Rotation Matrices

- quaternion definition

$$[x, v]$$

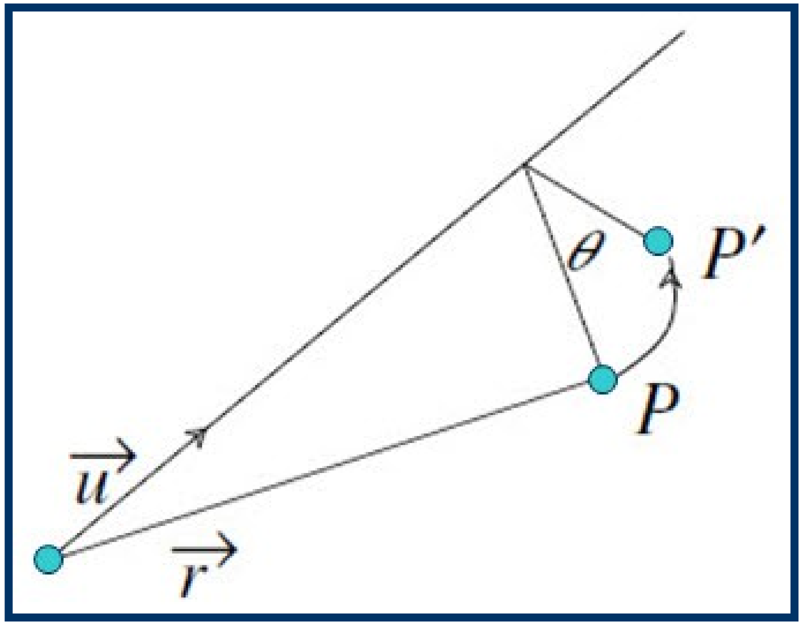
- quaternion multiplication

$$[s_1, v_1][s_2, v_2] = [s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2]$$

- rotation

$$[\cos(\theta/2), \sin(\theta/2)u]$$

- if *q₁* and *q₂* indicate rotations, then $q_2 q_1$ represents the composite rotation of q_1 followed by q_2



Quaternions vs. Rotation Matrices

- a formula for $\dot{q}(t)$

$$\dot{q}(t) = \frac{1}{2}\omega(t)\,q(t) \qquad (7\text{-}11)$$

where the multiplication $\omega(t)q(t)$ is a shorthand for multiplication between the quaternions $[0, \omega(t)]$ and q(t)

Proof:

Recall that the angular velocity $\omega(\mathrm{t})$ indicates that the body is instantaneously rotating about the $\omega(\mathrm{t})$ axis with magnitude $|\omega(\mathrm{t})|$. Suppose that a body were to rotate with a constant angular velocity $\omega(\mathrm{t})$. Then the rotation of the body after a period of time Δt is represented by the quaternion

$$[\cos\frac{|\omega(t)|\Delta t}{2}, \sin\frac{|\omega(t)|\Delta t}{2}\frac{\omega(t)}{|\omega(t)|}]$$

Quaternions vs. Rotation Matrices

Proof: (conti.)

Let us compute $\dot{q}(\mathrm{t})$ at some particular instant of time *t₀*. At times $t_0 + \Delta t$ (for small Δt), the orientation of the body is (to within first order) the result of first rotating by $q(\mathrm{t}_0)$ and then further rotating with velocity $\omega(\mathrm{t}_0)$ for $\Delta \mathrm{t}$ time.

Combining the two rotations, we get

$$q(t_0 + \Delta t) = [\cos \frac{|\omega(t_0)|\Delta t}{2}, \sin \frac{|\omega(t_0)|\Delta t}{2} \frac{\omega(t_0)}{|\omega(t_0)|}] \, q(t_0)$$

Substituting *t* for $t_0 + \Delta t$, we get

$$q(t) = [\cos \frac{|\omega(t_0)|(t - t_0)}{2}, \sin \frac{|\omega(t_0)|(t - t_0)}{2} \frac{\omega(t_0)}{|\omega(t_0)|}] q(t_0)$$

Differentiate *q(t)* at time *t₀,* we get

Quaternions vs. Rotation Matrices

Proof: (conti.)

$$\begin{aligned}\dot{q}(t) &= \frac{d}{dt}\left([\cos\frac{|\omega(t_0)|(t-t_0)}{2}, \sin\frac{|\omega(t_0)|(t-t_0)}{2}\frac{\omega(t_0)}{|\omega(t_0)|}]\,q(t_0)\right)\\ &= \frac{d}{dt}\left([\cos\frac{|\omega(t_0)|(t-t_0)}{2}, \sin\frac{|\omega(t_0)|(t-t_0)}{2}\frac{\omega(t_0)}{|\omega(t_0)|}]\right)q(t_0)\\ &= [0, \frac{|\omega(t_0)|}{2}\frac{\omega(t_0)}{|\omega(t_0)|}]\,q(t_0)\\ &= [0, \tfrac{1}{2}\omega(t_0)]\,q(t_0) = \tfrac{1}{2}[0, \omega(t_0)]\,q(t_0).\end{aligned}$$

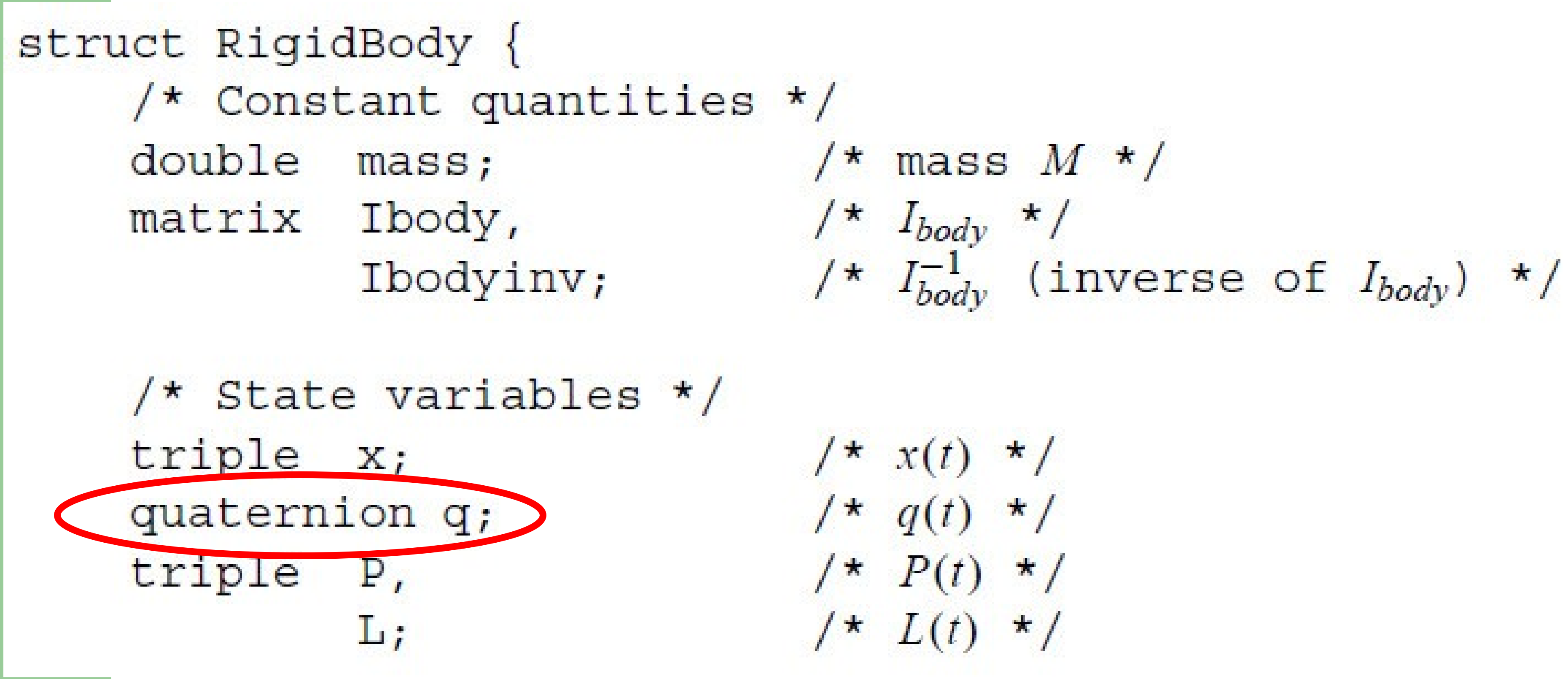
The product $[0, \omega(t_0)]q(t_0)$ is abbreviated to the form $\omega(t_0)q(t_0)$, thus, the general expression for $\dot{q}(t)$ is

$$\dot{q}(t) = \tfrac{1}{2}\omega(t)q(t).$$

Q.E.D.

Quaternions vs. Rotation Matrices

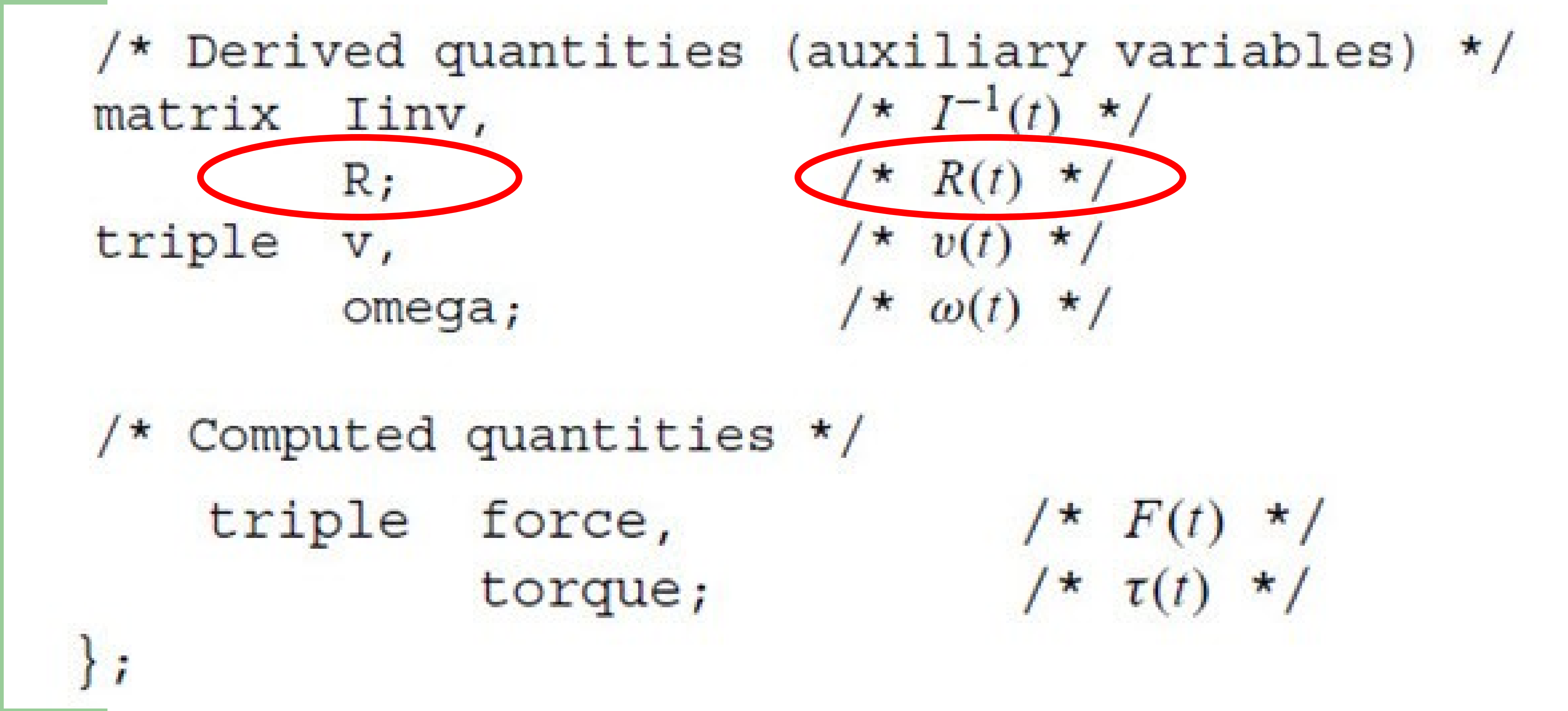
- to use a quaternion representation, need to redefine the type *RigidBody*:



```
struct RigidBody {
    /* Constant quantities */
    double  mass;               /* mass $M$ */
    matrix  Ibody,              /* $I_{body}$ */
            Ibodyinv;           /* $I_{body}^{-1}$ (inverse of $I_{body}$) */

    /* State variables */
    triple  x;                  /* $x(t)$ */
    quaternion q;               /* $q(t)$ */
    triple  P,                  /* $P(t)$ */
            L;                  /* $L(t)$ */
```

Quaternions vs. Rotation Matrices



```
/* Derived quantities (auxiliary variables) */
matrix  Iinv,                 /* $I^{-1}(t)$ */
        R;                    /* $R(t)$ */
triple  v,                    /* $v(t)$ */
        omega;                /* $\omega(t)$ */

/* Computed quantities */
    triple  force,                 /* $F(t)$ */
            torque;                /* $\tau(t)$ */
};
```

Quaternions vs. Rotation Matrices

- next, in *StateToArray*, replace the double loop

```
for(int i = 0; i < 3; i++)
    for(int j = 0; j < 3; j++)
       *y++ = rb->R[i,j];
```

Copy rotation matrix

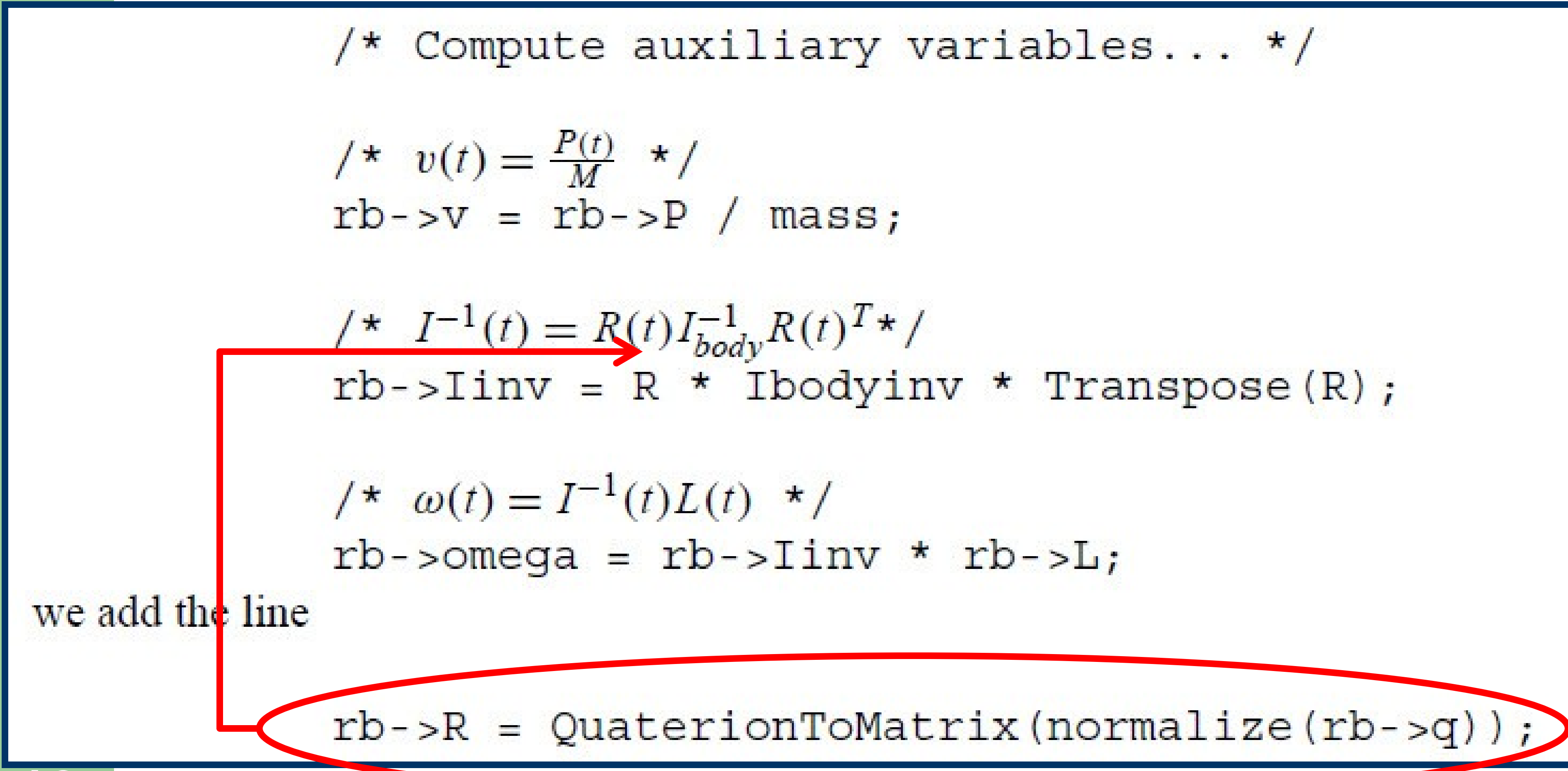
with

```
*y++ = rb->q.r;
*y++ = rb->q.i;
*y++ = rb->q.j;
*y++ = rb->q.k;
```

where quaternion is represented in terms of elements ‘r’ for the real part, and ‘i’, ‘j’, and ‘k’ for the vector part.

Quaternions vs. Rotation Matrices

- a similar change can be made in *ArrayToState*
- *ArrayToState* must also compute *R(t)* as an auxiliary variable: in the section



```
/* Compute auxiliary variables... */

/* $v(t) = \frac{P(t)}{M}$ */
rb->v = rb->P / mass;

/* $I^{-1}(t) = R(t) I_{body}^{-1} R(t)^T$ */
rb->Iinv = R * Ibodyinv * Transpose(R);

/* $\omega(t) = I^{-1}(t) L(t)$ */
rb->omega = rb->Iinv * rb->L;
```

we add the line

```
rb->R = QuaterionToMatrix(normalize(rb->q));
```

Quaternions vs. Rotation Matrices

- QuaterionToMatrix returns the matrix

$$\begin{pmatrix} 1-2v_y^2-2v_z^2 & 2v_xv_y-2sv_z & 2v_xv_z+2sv_y \\ 2v_xv_y+2sv_z & 1-2v_x^2-2v_z^2 & 2v_yv_z-2sv_x \\ 2v_xv_z-2sv_y & 2v_yv_z+2sv_x & 1-2v_x^2-2v_y^2 \end{pmatrix}$$

why?

- a brilliant way to convert a quaternion to a rotation matrix

Rotation of the vector p=(x, y, z) with the quaternion *q* is done by the operation $q[0,p]q^{-1}$. We want to determine the corresponding matrix which multiplied on $[x,y,z,1]^T$ from the left will yield the same result.

The product of two quaternions $q_v = [w,(a,b,c)]$ and $q_h = [s,(x,y,z)]$ is:

Quaternions vs. Rotation Matrices

$$q_v q_h = [w, (\mathrm{a}, \mathrm{b}, \mathrm{c})][s, (\mathrm{x}, \mathrm{y}, \mathrm{z})]$$
$$= [ws - ax - by - cz, (as + wx - cy + bz,$$
$$bs + wy + cx - az, cs + wz - bx + ay)]$$

Written as columns using sloppy notation this equals

$$q_v q_h = \begin{bmatrix} a \\ b \\ c \\ w \end{bmatrix} \times_q \begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix} = \begin{bmatrix} wx - cy + bz + as \\ cx + wy - az + bs \\ -bx + ay + wz + cs \\ -ax - by - cz + ws \end{bmatrix}$$

From this we can write the matrices corresponding to multi-plying from the left and from the right with a quaternion.

First we determine M_v such that $M_v q_h = q_v q_h$ where q_h and $q_v q_h$ are written as columns:

Quaternions vs. Rotation Matrices

$$\mathbf{M_v} = \begin{bmatrix} w & -c & b & a \\ c & w & -a & b \\ -b & a & w & c \\ -a & -b & -c & w \end{bmatrix}$$

Then we write M_h such that $M_h q_v = q_v q_h$:

$$\mathbf{M_h} = \begin{bmatrix} s & z & -y & x \\ -z & s & x & y \\ y & -x & s & z \\ -x & -y & -z & s \end{bmatrix}$$

We are now ready to write the matrix M such that $Mq = q[0, p]q^{-1}$. Using $q = [s, (x, y, z)]$ and $q^{-1} = [s, (-x, -y, -z)]$ we get:

Quaternions vs. Rotation Matrices

$$M = M_q M_{q^{-1}}$$

$$= \begin{bmatrix} s & -z & y & x \\ z & s & -x & y \\ -y & x & s & z \\ -x & -y & -z & s \end{bmatrix} \begin{bmatrix} s & -z & y & -x \\ z & s & -x & -y \\ -y & x & s & -z \\ x & y & z & s \end{bmatrix}$$

$$= \begin{bmatrix} 1-2(y^2+z^2) & 2xy-2sz & 2sy+2xz & 0 \\ 2xy+2sz & 1-2(x^2+z^2) & -2sx+2yz & 0 \\ -2sy+2xz & 2sx+2yz & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Q.E.D.

Quaternions vs. Rotation Matrices

$$M = M_q M_{q^{-1}}$$

$$= \begin{bmatrix} s & -z & y & x \\ z & s & -x & y \\ -y & x & s & z \\ -x & -y & -z & s \end{bmatrix} \begin{bmatrix} s & -z & y & -x \\ z & s & -x & -y \\ -y & x & s & -z \\ x & y & z & s \end{bmatrix}$$

$$= \begin{bmatrix} 1-2(y^2+z^2) & 2xy-2sz & 2sy+2xz & 0 \\ 2xy+2sz & 1-2(x^2+z^2) & -2sx+2yz & 0 \\ -2sy+2xz & 2sx+2yz & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Q.E.D.

Quaternions vs. Rotation Matrices

- convert from a rotation matrix to a quaternion

Recall that

$$M = \begin{bmatrix} 1-2(y^2+z^2) & 2xy-2sz & 2sy+2xz & 0 \\ 2xy+2sz & 1-2(x^2+z^2) & -2sx+2yz & 0 \\ -2sy+2xz & 2sx+2yz & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

First, we find *s* :

$$\begin{aligned} M_{11}+M_{22}+M_{33}+1 &= 4-4(x^2+y^2+z^2) \\ &= 4-4(1-s^2) \\ &= 4s^2 \end{aligned}$$

This yields s^2. Now the other values follow:

$$s = \pm\frac{1}{2}\sqrt{M_{11}+M_{22}+M_{33}+1}$$

Quaternions vs. Rotation Matrices

$$M = \begin{bmatrix} 1-2(y^2+z^2) & 2xy-2sz & 2sy+2xz & 0 \\ 2xy+2sz & 1-2(x^2+z^2) & -2sx+2yz & 0 \\ -2sy+2xz & 2sx+2yz & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x = \frac{M_{32}-M_{23}}{4s}$$

$$y = \frac{M_{13}-M_{31}}{4s}$$

$$z = \frac{M_{21}-M_{12}}{4s}$$

The sign of *x*, *y* and *z* depends on the sign of *s*. For *s*, we choose the positive square root.

(why? Note that positive square root and negative square root yield the same rotation, but the interpolation curve could be influenced by this choice.)

Quaternions vs. Rotation Matrices

- Hence, to convert from a rotation matrix to a quaternion:

```
quaternion matrixToQuaternion(const matrix &m)
{
    quaternion   q;
    double       tr, s;

    tr = m[0,0] + m[1,1] + m[2,2];

    if(tr >= 0)
    {
        s = sqrt(tr + 1);
        q.r = 0.5 * s;
        s = 0.5 / s;
        q.i = (m[2,1] - m[1,2]) * s;
        q.j = (m[0,2] - m[2,0]) * s;
        q.k = (m[1,0] - m[0,1]) * s;
    }
```

The matrix *m* is structured so that *m*[0, 0], *m*[0, 1] and *m*[0, 2] form the first row of *m*

```
else
{
    int i = 0;

    if(m[1,1] > m[0,0])
        i = 1;
    if(m[2,2] > m[i,i])
        i = 2;

    switch (i)
    {
    case 0:
        s = sqrt((m[0,0] - (m[1,1] + m[2,2])) + 1);
        q.i = 0.5 * s;
        s = 0.5 / s;
        q.j = (m[0,1] + m[1,0]) * s;
        q.k = (m[2,0] + m[0,2]) * s;
        q.r = (m[2,1] - m[1,2]) * s;
        break;
    case 1:
        s = sqrt((m[1,1] - (m[2,2] + m[0,0])) + 1);
        q.j = 0.5 * s;
        s = 0.5 / s;
```

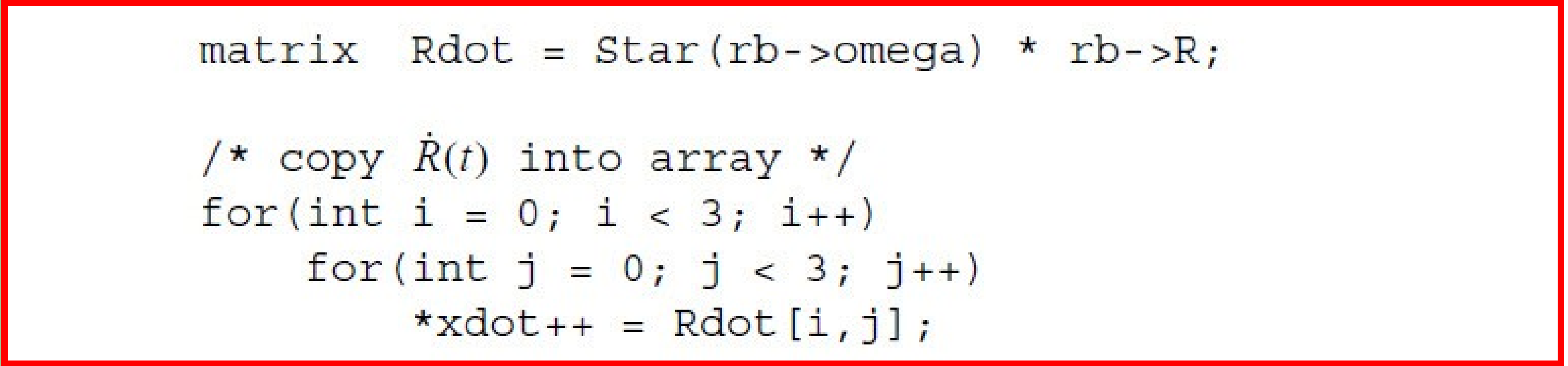
Quaternions vs. Rotation Matrices

```
            q.k = (m[1,2] + m[2,1]) * s;
            q.i = (m[0,1] + m[1,0]) * s;
            q.r = (m[0,2] - m[2,0]) * s;
            break;
        case 2:
            s = sqrt((m[2,2] - (m[0,0] + m[1,1])) + 1);
            q.k = 0.5 * s;
            s = 0.5 / s;
            q.i = (m[2,0] + m[0,2]) * s;
            q.j = (m[1,2] + m[2,1]) * s;
            q.r = (m[1,0] - m[0,1]) * s;
        }

    }
    return q;
}
```

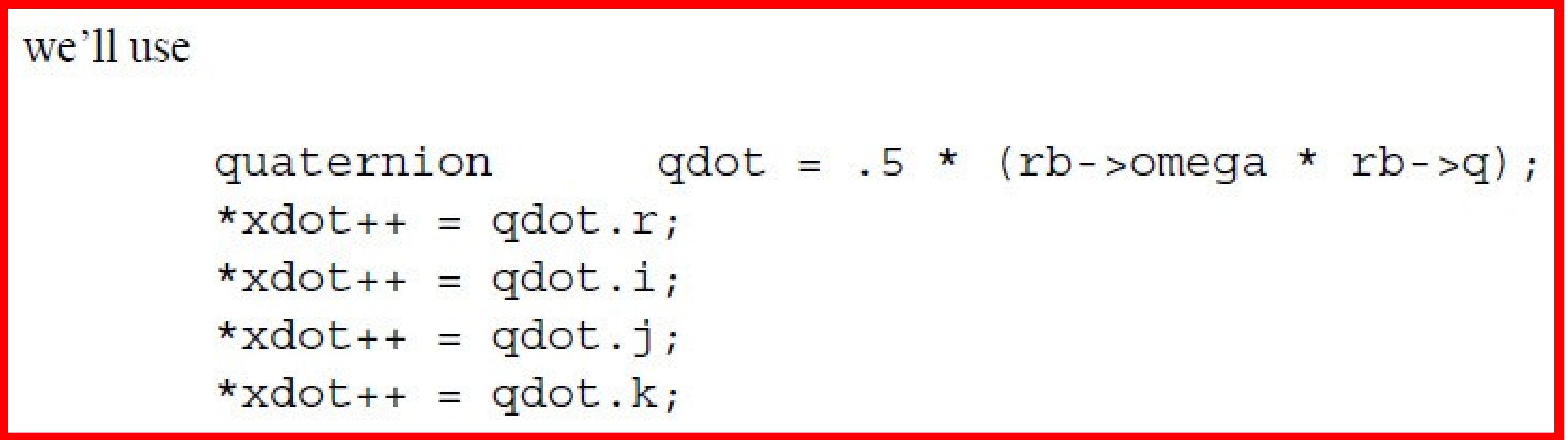
Quaternions vs. Rotation Matrices

- *ArrayToBodies* and *BodiesToArray* don’t need changes
- constant *STATE_SIZE* changes from 18 to 13
- *ddtStateToArray* needs changes. Instead of



```
matrix  Rdot = Star(rb->omega) * rb->R;

/* copy $\dot{R}(t)$ into array */
for(int i = 0; i < 3; i++)
    for(int j = 0; j < 3; j++)
        *xdot++ = Rdot[i,j];
```

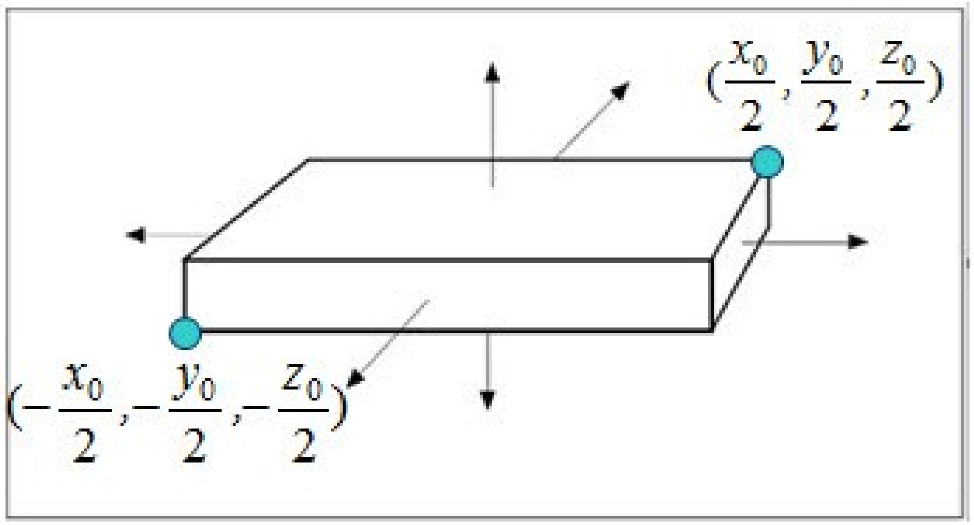


we’ll use

```
    quaternion      qdot = .5 * (rb->omega * rb->q);
    *xdot++ = qdot.r;
    *xdot++ = qdot.i;
    *xdot++ = qdot.j;
    *xdot++ = qdot.k;
```

Examples – Inertia Tensor of a block

- How should the inertia tensor of the following block be computed?



$(\frac{x_0}{2}, \frac{y_0}{2}, \frac{z_0}{2})$
$(-\frac{x_0}{2}, -\frac{y_0}{2}, -\frac{z_0}{2})$

$$\rho(x, y, z) \equiv 1$$

$$M = x_0 y_0 z_0$$

- Again, what is an inertia tensor?
  Given an object with mass *m*, what force do I need to apply to get an acceleration *a*? $\boxed{F = ma}$ **(7-11)**

  If I apply a force 2**F*, I get an acceleration 2**a*. If the object gets heavier, I need to apply more force to get the same acceleration.

Examples – Inertia Tensor of a block

Now I want to ask the same question for *rotation.* First let's assume we have a world with known axes x, y, and z. And let's define the *rotational velocity ω* for an object as a three element-vector ($\omega_x, \omega_y, \omega_z$) that contains the rate at which the object is spinning around each axis.

If all three are zero, the object isn't spinning at all. If ω_x is the only non-zero element, the object is spinning nicely around the x-axis, etc.

So now I want to know what *torque* I need to apply to accelerate the object's rotation by some amount *dω.* For example, maybe the object is spinning around the x axis with rotational velocity ($\omega_x, 0, 0$), and I want to stop it in *t* seconds. *So I want to know what torque* I need to apply to get a rotational acceleration of ($-\omega_x/t$).

Examples – Inertia Tensor of a block

This is where the *inertia tensor* comes in. Just like we have $F = ma$ for linear acceleration and force, we have the following equation for rotation :

$$\boxed{\tau = I * d\omega} \tag{7-12}$$

...where τ is torque (a 3-element vector, indicating the torque around the x, y, and z axes), $d\omega$ is the rotational acceleration (a 3-element vector), and I is the magic inertia tensor (a 3-by-3 matrix). Note how it looks just like equation (7-11)!

In matrix form, the above equation looks like :

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} d\omega_x \\ d\omega_y \\ d\omega_z \end{bmatrix} \tag{7-13}$$

Examples – Inertia Tensor of a block

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} d\omega_x \\ d\omega_y \\ d\omega_z \end{bmatrix} \qquad (7\text{-}13)$$

what do the individual elements of the inertia tensor *mean* in the real world? The easy ones are the diagonal elements: I_{xx} , I_{yy} and I_{zz} . In fact, let's take the simple case, where all the other elements in I are zero. This happens to be true for any object that's symmetric around all three axes, like a big sphere, or a big cube that's sitting nicely on the axes. We'll deal with the other elements later.

In this case, if we multiply the first row of equation (7-13) out, we get :

$$\boxed{\tau_x = I_{xx} * d\omega_x} \qquad (7\text{-}14)$$

Examples – Inertia Tensor of a block

$$\boxed{\tau_x = I_{xx} * d\omega_x} \qquad (7\text{-}14)$$

This looks a *lot* like *F = ma.*

Can think of *Ixx* as the rotational inertia around the *x* axis.

So for rotation around the *x* axis, *Ixx* behaves just like the *m* in *F=ma.*

The bigger *Ixx* is, the more torque we have to apply around the x axis to get it to spin.

Or, conversely, for a given torque, a bigger *Ixx* means we'll get *less* rotation around the *x* axis. Just like *m*.

What would it mean for *Ixx* to be zero?

It would mean that for a given torque, we would get an *infinite* rotational acceleration around the x axis.

For real objects, none of the diagonal elements in *I* can be zero, since we know there's no real object where a tiny touch will set it spinning at an infinite rate.

Examples – Inertia Tensor of a block

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} d\omega_x \\ d\omega_y \\ d\omega_z \end{bmatrix} \qquad (7\text{-}13)$$

what's the deal with the off-diagonal elements, for example I_{xy} ?

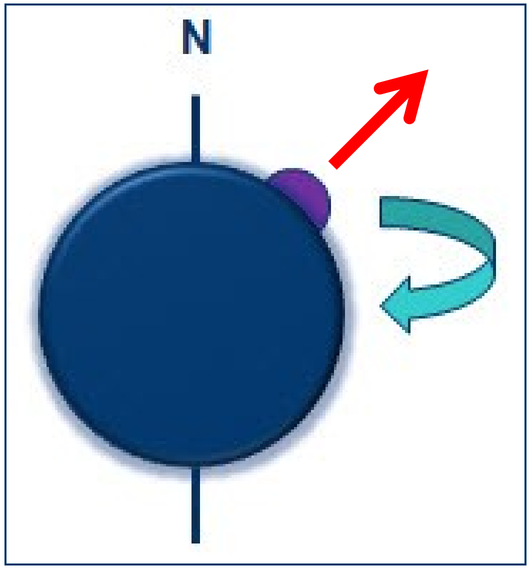
What this element (I_{xy}) tells us is how much our object will be accelerated around the *y* axis when we apply torque around the *x* axis.

How's that possible? How could spinning an object around one axis make it rotate around another axis too?

For symmetric objects, it can't. If we take a uniform cube in space and we torque it around the x axis, intuition suggests that it only spins around the x axis.

Examples – Inertia Tensor of a block

But now let's say I attach a big-ass weight somewhere on the sphere.



For example, let's say I'm spinning the Earth around the axis that runs from the north pole to the south pole (and we're assuming of course that the Earth is a uniform sphere), and I attach a big weight to Canada.

Now suddenly I've accelerated that big weight, and it wants to "pull" that part of the world along with it.

This is going to make the Earth want to "tilt", so the north pole moves along with big heavy Canada.

This is a rotational acceleration along an axis other than the one I torqued!

Examples – Inertia Tensor of a block

Remember that off-diagonal elements are always zero for a perfectly symmetric object

Actually, for any object, there's some basis along which the inertia tensor is diagonal (all the off-diagonal elements are zero).

That means that there's some set of perpendicular axes (*x*,*y*,*z*) around which we can torque the object and get rotational acceleration only along the axis we torqued…

Another way of saying this is that there's some set of "principal axes" around which an object will "spin stably", meaning it can spin around those axes without "wobbling" off-axis.

Finding those axes is straightforward, but that is not our concern here, so I'm not going to talk about it.

Examples – Inertia Tensor of a block

Now, how the inertia tensor is defined…

For the diagonal elements, say I_{xx}, we have

$$\boxed{I_{xx} = \int_V \rho(x,y,z)(y^2+z^2)dV} \tag{7-14}$$

where $\rho(x,y,z)$ is the density of the object, *V* is the volume of the object, and *y* and *z* are the positions of a particular point along the *y* and *z* axes.

What does the *I*xx equation (7-14) mean?

We're taking an integral over the whole volume, and we're summing the squared distance of every point from the *x* axis (y^2+z^2). So for a given amount of mass, we are going to have a bigger *I*xx value if most of my object is far away from the *x* axis.

Examples – Inertia Tensor of a block

This means to get a given acceleration around the *x* axis, we need to apply more torque around the *x* axis.

How can we understand that intuitively?

The classic example for understanding this comes from figure skating… how does an ice skater get ready to spin super fast on the ice? What does he/she do? he puts his arms way above his head.

What does this have to do with the inertia tensor? If axis of the skater's body is the *x* axis, he is spinning himself around this axis with some torque (whatever he can generate with his legs). Eq. (7-14) shows that the lower his I_{xx} value is, the more rotational acceleration he'll get for that torque. He can make his I_{xx} value smaller by putting his mass closer to the *x* axis… and he does that by putting his arms –which carry some of his weight – along the axis of his body. Amazing.

Examples – Inertia Tensor of a block

Nest, the off-diagonal elements. Equation for I_{xy} looks like this :

$$\boxed{I_{xy} = -\int_V \rho(x,y,z)xy\, dV} \qquad \text{(7-15)}$$

where $\rho(x,y,z)$ is the density of the object, *V* is the volume of the object, and *x* and *y* are the distance of a particular point from the *yz* and the *xz* planes.

The off-diagonal elements are called *products of inertia*, they are a measure of the imbalance in the mass distri-bution.

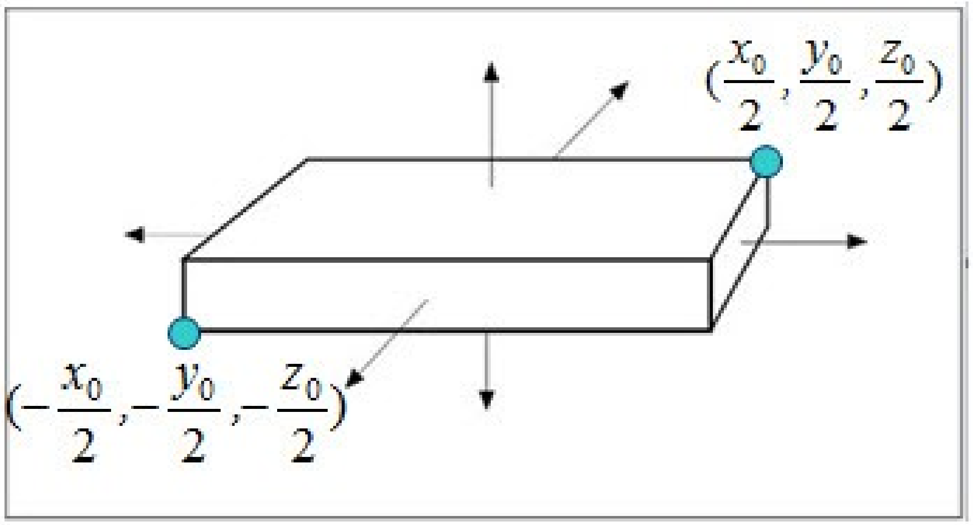
Products of inertia can be positive, negative, or zero. If the object is symmetric, every point on one side of each axis is going to “cancel out” the corresponding point on the other side of the axis. Hence we get off-diagonal elements that are zero.

Examples – Inertia Tensor of a block

$$\boxed{I_{xy} = -\int_V \rho(x,y,z)xy\, dV} \qquad \textbf{(7-15)}$$

*I*xy is exactly the same as *I*yx. This tells us that all inertia tensors are symmetric, which makes them numerically friendly for many applications.

- So, how should the inertia tensor of the following block be computed?


$(\frac{x_0}{2}, \frac{y_0}{2}, \frac{z_0}{2})$
$(-\frac{x_0}{2}, -\frac{y_0}{2}, -\frac{z_0}{2})$

$$\rho(x,y,z) \equiv 1$$

$$M = x_0 y_0 z_0$$

Examples – Inertia Tensor of a block

$$I_{xx} = \int_{\frac{-x_0}{2}}^{\frac{x_0}{2}} \int_{\frac{-y_0}{2}}^{\frac{y_0}{2}} \int_{\frac{-z_0}{2}}^{\frac{z_0}{2}} \rho(x,y,z)(y^2+z^2)\,dx\,dy\,dz$$

$$= \int_{\frac{-x_0}{2}}^{\frac{x_0}{2}} \int_{\frac{-y_0}{2}}^{\frac{y_0}{2}} y^2 z + \frac{z^3}{3}\bigg|_{z=\frac{-z_0}{2}}^{z=\frac{z_0}{2}} dx\,dy$$

$$= \int_{\frac{-x_0}{2}}^{\frac{x_0}{2}} \int_{\frac{-y_0}{2}}^{\frac{y_0}{2}} y^2 z_0 + \frac{z_0^3}{12}\,dx\,dy$$

$$= \int_{\frac{-x_0}{2}}^{\frac{x_0}{2}} \frac{y_0^3}{3} z_0 + \frac{z_0^3}{12} y_0\bigg|_{y=\frac{-y_0}{2}}^{y=\frac{y_0}{2}} dx$$

$$= \int_{\frac{-x_0}{2}}^{\frac{x_0}{2}} \frac{y_0^3 z_0}{12} + \frac{z_0^3 y_0}{12}\,dx$$

$$= \frac{x_0 y_0 z_0}{12}(y_0^2 + z_0^2) = \frac{M}{12}(y_0^2 + z_0^2).$$

Examples – Inertia Tensor of a block

Similarly, $I_{yy} = \frac{M}{12}(x_0^2 + z_0^2)$ and $I_{zz} = \frac{M}{12}(x_0^2 + y_0^2)$.

- The off-diagonal terms, such as *I*$_{xy}$, are

$$I_{xy} = \int_{\frac{-x_0}{2}}^{\frac{x_0}{2}} \int_{\frac{-y_0}{2}}^{\frac{y_0}{2}} \int_{\frac{-z_0}{2}}^{\frac{z_0}{2}} \rho(x,y,z)(xy)\,dx\,dy\,dz$$

$$= \int_{\frac{-x_0}{2}}^{\frac{x_0}{2}} \int_{\frac{-y_0}{2}}^{\frac{y_0}{2}} \int_{\frac{-z_0}{2}}^{\frac{z_0}{2}} xy\,dx\,dy\,dz = 0$$

and similarly for the others. Thus, the inertia tensor of the block is

$$I_{body} = \frac{M}{12}\begin{pmatrix} y_0^2 + z_0^2 & 0 & 0 \\ 0 & x_0^2 + z_0^2 & 0 \\ 0 & 0 & x_0^2 + y_0^2 \end{pmatrix}$$

End of Physically Based Animation III