Non Spherical Particles

When the particles were assumed spherical with smooth surface, computational costs associated with representing their geometries, tracking them and finding the contact plane/point are low.

However many practical granular flows involve non-spherical particles that may be near round or with irregular shapes.

Using non-spherical in the DEM as compared with spherical particles, requires additional computational costs due to implementing more complex algorithms for contact detection, tracking particles and contact force calculations. This is compulsory for performing accurate DEM simulation, even though the costs are high.

Shape Representation:

In the implementation of DEM for non-spherical particles, four issues should be addressed:

1. The kinematics of the particle
2. The method of representing particle shape to DEM
3. Calculation of the contact force
4. Finding the contact plane between particles.

1. Equation of rotational motion of a non-spherical particle is different from that of a sphere. A method should be therefore considered for defining particle orientation in 3D space and solving Euler's equations.

1. A sphere can be represented by its center point and radius in the computer, while shape representation of a non-spherical particle (exact or approximate) requires additional data.

1. Shape representation is still one of the key challenges in the DEM modeling of granular flows.

1. According to the way that particle shape is represented, the method of finding a contact plane between particles and force calculation should be changed.

Multi-Element Approach:

In the multi-element particle approach, complex surface of the particle is constructed from smaller sub-elements (with simple shapes) that are connected to each other.

Among different methods in this group, the multi-sphere method, also called glued spheres or clustered spheres, is one of the most common one. The complex surface of a particle can be formed by overlapped spheres.

By increasing the number of spheres used in the construction of the complex particle, the real surface would be represented more accurately.

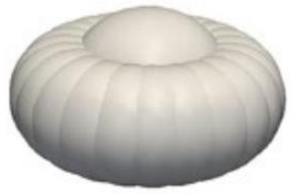
However, regardless of the number of spheres, the resultant surface would be bumpy and without a sharp edge or vertex

A DEM code for spherical particle can be easily extended to handle multi-sphere method. The contact search and contact force calculations are similar, although calculation of force needs some modifications.

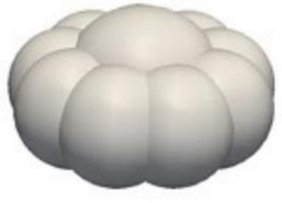
The multi-sphere method is not limited to a particular shape or a family of geometric shapes and theoretically can produce any complex surface, except surfaces with sharp edges.

By increasing the number of spheres in a particle, the particle surface is represented more accurately, while at the same time it increases the computational costs and errors due to multiple contact points.

Large number of spheres



Small number of spheres



Kinematics and Dynamics of Rigid Bodies:

A non-spherical particle may be a single body whose surface is defined by analytical equations or a body composed of sub-elements that are rigidly connected to each other.

Therefore, the basic principles of the kinematics and dynamics of the non-spherical rigid body apply to both groups.

- The position of a non-spherical rigid body in the space is defined by the location of its center of mass and its orientation.
- In Cartesian coordinates, two different coordinates systems are required. First, an orthogonal body-fixed frame (local coordinates) located on the center of mass and moves and rotates with the rigid body.1 Second, an orthogonal space-fixed frame (global coordinates) that is inertial and it does not move or rotate in the space.
- In a DEM simulation, all variable such as position, orientation, and velocities of the particle, as well as location of walls, are defined in this space-fixed frame.
- The transformation of vector variables between the body-fixed and space-fixed frames are frequently performed to compute contact force and torque as well as to solve equations of motion.
- The location and orientation of the body-fixed frame (hence the body itself) is defined relative to the space-fixed frame. Therefore, a convention should be used to transform variables from the space-fixed frame to the body-fixed frame and vice versa.

Euler Angles and Transformation Matrix:

In 2D space, three scalar variables (three degrees of freedom) are required to define the rigid body motion: two for coordinates of the center of mass and one for the rotation angle θ.

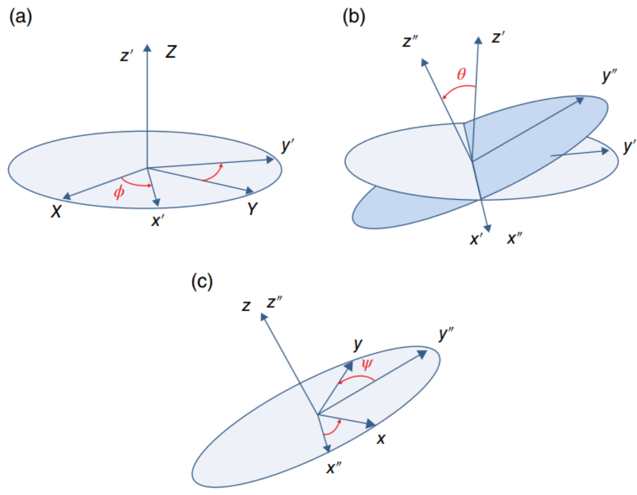
The rotation by the angle θ around the center of mass is simply done using the following rotation matrix:

$$A_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

In 3D space, six scalar variables (six degrees of freedom) are required for defining the rigid body motion: three for the coordinates of the center of mass and three angles for

There is a variety of methods to define the orientation of a body in 3D space. The most well-known is by using Euler angles.

According to the classical Euler angles, the orientation of the body-fixed frame is defined by three Euler angles ϕ, θ, and ψ.



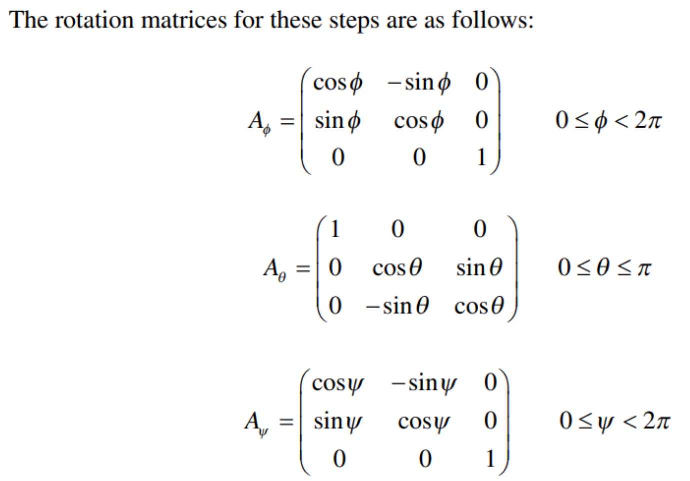
Euler Angles:

Consider the spaced-fixed frame XYZ. It is desirable to perform a set of rotations to transform this frame to the body-fixed frame xyz. For this purpose, the following steps should be carried out in sequence:

1. First, a rotation is done by ϕ around Z-axis that results in a new coordinate system x′y′z′.

1. Then, in the x′y′z′ coordinates system, the second rotation is done by θ around x′-axis, which gives a new coordinate system x″y″z″.

1. Finally, in the x″y″z″ coordinates system, the third rotation is done by ψ around z″-axis and gives the body-fixed frame coordinates xyz.

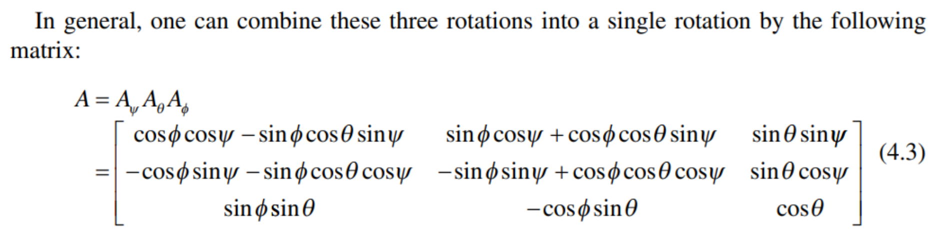


The rotation matrices for these steps are as follows:

$$A_\phi = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad 0 \le \phi < 2\pi$$

$$A_\theta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \qquad 0 \le \theta \le \pi$$

$$A_\psi = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad 0 \le \psi < 2\pi$$

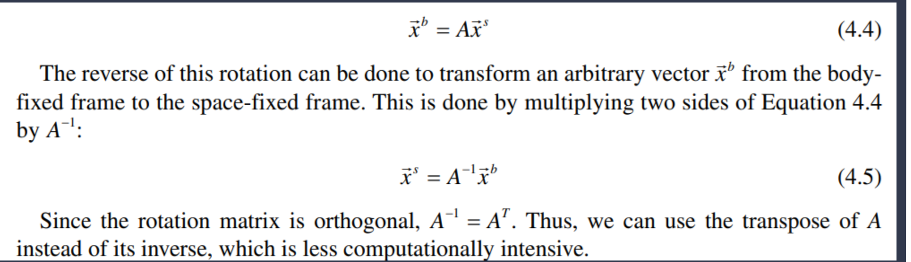


In general, one can combine these three rotations into a single rotation by the following matrix:

$$\begin{aligned} A &= A_\psi A_\theta A_\phi \\ &= \begin{bmatrix} \cos\phi\cos\psi - \sin\phi\cos\theta\sin\psi & \sin\phi\cos\psi + \cos\phi\cos\theta\sin\psi & \sin\theta\sin\psi \\ -\cos\phi\sin\psi - \sin\phi\cos\theta\cos\psi & -\sin\phi\sin\psi + \cos\phi\cos\theta\cos\psi & \sin\theta\cos\psi \\ \sin\phi\sin\theta & -\cos\phi\sin\theta & \cos\theta \end{bmatrix} \end{aligned} \qquad (4.3)$$

Euler Angles:

A is the transformation matrix that transforms an arbitrary vector **xs** in the space-fixed frame to **xb** in the body-fixed frame by the following equation:



$$\vec{x}^b = A\vec{x}^s \tag{4.4}$$

The reverse of this rotation can be done to transform an arbitrary vector $\vec{x}^b$ from the body-fixed frame to the space-fixed frame. This is done by multiplying two sides of Equation 4.4 by A^{-1}:

$$\vec{x}^s = A^{-1}\vec{x}^b \tag{4.5}$$

Since the rotation matrix is orthogonal, $A^{-1} = A^T$. Thus, we can use the transpose of A instead of its inverse, which is less computationally intensive.

Each element of the transformation matrix A is a direct cosine of the angle between an axis of the body-fixed frame and the space-fixed frame.

Therefore, the transformation matrix also describes the relative orientation of the two-coordinate systems with nine parameters. For instance, the first column of A, transforms the unit vector i of the space-fixed frame to the body-fixed frame. Similarly, the second column transforms the unit vector j and the third column, the unit vector k

We are also interested in knowing the relation between the angular velocity of body in the body-fixed frame and incremental change (time derivatives) of Euler angles. The absolute angular velocity in the body-fixed frame is the vector sum of time derivatives change of Euler angles:

$$\omega^b = \dot{\phi} + \dot{\theta} + \dot{\psi} \tag{4.6}$$

It is relatively easy to show that [31]:

$$\omega_x^b = \dot{\phi}\sin\theta\sin\psi + \dot{\theta}\cos\psi \tag{4.7a}$$

$$\omega_y^b = \dot{\phi}\sin\theta\cos\psi - \dot{\theta}\sin\psi \tag{4.7b}$$

$$\omega_z^b = \dot{\psi} + \dot{\phi}\cos\theta \tag{4.7c}$$

If we solve these three equations for $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$, we get time derivatives of Euler angles in terms of fixed-body angular velocity:

$$\dot{\phi} = \frac{\omega_x^b \sin\psi + \omega_y^b \cos\psi}{\sin\theta} \tag{4.8a}$$

$$\dot{\theta} = \omega_x^b \cos\psi - \omega_y^b \sin\psi \tag{4.8b}$$

$$\dot{\psi} = \omega_z^b - \dot{\phi}\cos\theta \tag{4.8c}$$

Equations 4.8a–4.8c are important relations, since the angular equation of motion is solved in the orthogonal body-fixed frame and ωb is obtained, while non-orthogonal Euler angles are used to represent the orientation. Thus, these equations are used to obtain new orientation of the body in each time-step of simulation by integrating time derivatives of Euler angles.

Equations of Motion:

The motion of a body in 3D space is defined by translational motion of the center of mass and rotational motion about the center of mass.

The general form of the translational equation of motion of the center mass is:

$$m_i\vec{a}_i = m_i\frac{d\vec{v}_i}{dt} = \vec{f}_i$$

Where 'f' is the sum of forces acting on the center of mass of the body in the space-fixed frame and 'v' is the translational velocity of the center of mass in the space-fixed frame.
We can solve the translational motion of the center of mass of a non-spherical body similar to a spherical particle.

Rotational Motion of a 3D body:

The rotational motion of a rigid body in 3D space is given by:

$$\dot{\vec{L}} = \vec{M} \tag{4.10}$$

where $\dot{\vec{L}}$ is the rate of change of angular momentum and $\vec{M}$ is sum of torques acting on a particle. The reference point to evaluate torques and angular momentum should be similar and it can be any arbitrary point that moves with the center of mass of the body. It is more common to choose the center of mass as the reference point. Then, the angular momentum reads as:

$$\vec{L}^b = I\,\vec{\omega}^b \tag{4.11}$$

where I is the inertia tensor of the body at its center of mass and is defined as:

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yx} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \tag{4.12}$$

Diagonal elements of this matrix are called centroidal mass moment of inertia and off- diagonal elements are called centroidal mass product of inertia.

Rotational Motion (Euler's Equations):

The derivative of angular momentum of the rigid body is obtained by:

$$\dot{\vec{L}}^b = I\dot{\vec{\omega}}^b + \vec{\omega}^b \times I\vec{\omega}^b$$

Therefore, the equation of rotational motion around its center of mass in the body-fixed frame becomes:

$$\vec{M}^b = I\dot{\vec{\omega}}^b + \vec{\omega}^b \times I\vec{\omega}^b$$

and in matrix form:

$$\begin{bmatrix} M_x^b \\ M_y^b \\ M_z^b \end{bmatrix} = I \begin{bmatrix} \dot{\omega}_x^b \\ \dot{\omega}_y^b \\ \dot{\omega}_z^b \end{bmatrix} + \begin{bmatrix} 0 & -\omega_z^b & \omega_y^b \\ \omega_z^b & 0 & -\omega_x^b \\ -\omega_y^b & \omega_x^b & 0 \end{bmatrix} I \begin{bmatrix} \omega_x^b \\ \omega_y^b \\ \omega_z^b \end{bmatrix}$$

This set of equations should be solved to obtain the rotational velocity of body in the fixed- body frame.

If the reference point is kept on the center of mass and the body-fixed coordinates is aligned to coincide with principal axes of the body, then the off-diagonal elements of the inertia tensor vanish and the given matrix equation turns into well known Euler's Equations:

$$M_x^b = \hat{I}_1\dot{\omega}_x^b + \left(\hat{I}_3 - \hat{I}_2\right)\omega_y^b\omega_z^b$$

$$M_y^b = \hat{I}_2\dot{\omega}_y^b + \left(\hat{I}_1 - \hat{I}_3\right)\omega_x^b\omega_z^b$$

$$M_z^b = \hat{I}_3\dot{\omega}_z^b + \left(\hat{I}_2 - \hat{I}_1\right)\omega_x^b\omega_y^b$$

where Î 1 , Î 2 , and Î 3 are principal central moments of inertia about the center mass of body.

Since Euler's equations are simpler and involve less non-linear terms, we choose the orientation of the body-fixed frame in a way that it coincides with the principal axes of the body.

Rotational Motion:

1. Method of Calculating the new orientation of the non spherical particle by calculating the derivatives of Euler angles and then Using Numerical Integration to compute the new orientation cannot be used because of the problem of singularity.

1. We discuss two methods here: using the transformation matrix to define the orientation instead of Euler angles and using quaternions to describe orientation.

1. As we discussed earlier, elements of the transformation matrix are direct cosine of angles between axes of the body-fixed frame and axes of the space-fixed frame. Thus, the transformation matrix describes the orientation of the body-fixed frame relative to the space-fixed frame with nine parameters.

If the time derivative of the transformation matrix (dA/dt) is calculated, a new transformation matrix (new orientation of body) at time t + Δt is obtained by numerical integration.

In contrast to the Euler angles method, in which the time derivative of Euler angles is used to obtain new orientation, time derivative of the transformation matrix is used in this method.

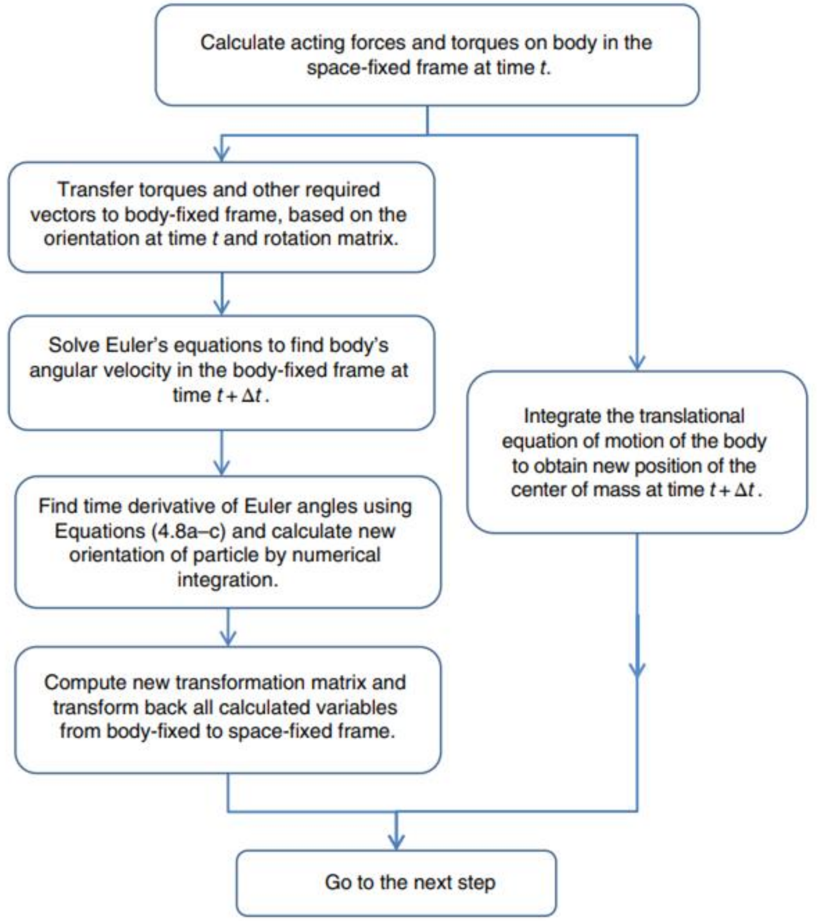
This solution remedies the problem of singularity associated with the derivative of Euler angles for $\theta = 0$. The transformation matrix uses nine parameters for describing the orientation of body in 3D space. This means that it requires six constraints (independent equations) to make the system determined.

These constraints come from the orthogonal property of the transformation matrix.

If dA/dt is integrated to obtain the new orientation of the body in each time step, very small numerical errors are introduced in the elements of A. These small numerical errors are accumulated over time and matrix A loses its orthogonality and hence it would be no longer a transformation matrix.

Step-wise procedure for numerical integration is complex and can be implemented rather hard into a regular DEM code. In addition, it involves too many matrix operations, which degrades the computational efficiency.

Rotational Motion:



Calculate acting forces and torques on body in the space-fixed frame at time t.
Transfer torques and other required vectors to body-fixed frame, based on the orientation at time t and rotation matrix.
Solve Euler's equations to find body's angular velocity in the body-fixed frame at time $t+\Delta t$.
Find time derivative of Euler angles using Equations (4.8a–c) and calculate new orientation of particle by numerical integration.
Compute new transformation matrix and transform back all calculated variables from body-fixed to space-fixed frame.
Integrate the translational equation of motion of the body to obtain new position of the center of mass at time $t+\Delta t$.
Go to the next step

Figure 4.4 Steps to integrate/solve equations of motion of a non-spherical rigid particle in 3D space. This flow chart corresponds to steps 5–7 in Figure 3.1

Rotational Motion: Method of Quaternions

Quaternion is a vector in 4D space that extends a complex number into a higher dimension. A quaternion is defined using a scalar value s and a 3D vector **v** as follows:

$$q=(s,\vec{v}) \quad \text{or} \quad q=s\tilde{e}_0+v_1\tilde{e}_1+v_2\tilde{e}_2+v_3\tilde{e}_3$$

where $(\tilde{e}_0,\tilde{e}_1,\tilde{e}_2,\tilde{e}_3)$ are elements of the quaternion space that should hold the following relation:

$$\tilde{e}_0^2+\tilde{e}_1^2+\tilde{e}_2^2+\tilde{e}_3^2=1 \tag{4.20}$$

The product of two quaternions is defined as follows:

$$q_1.q_2=(s_1,\vec{v}_1).(s_2,\vec{v}_2)=(s_1s_2-\vec{v}_1.\vec{v}_2,\ s_1\vec{v}_2+s_2\vec{v}_1+\vec{v}_1\times\vec{v}_2)$$

Quaternion product is not commutative.

We can also define the quaternion in terms of a rotation angle α around vector u:

$$q_\alpha=\left(\cos\frac{\alpha}{2},\vec{u}\sin\frac{\alpha}{2}\right)$$

The following expression is used to rotate and arbitrary vector $\vec{r}$ by angle α around vector $\vec{u}$

$$\vec{r}'=q_\alpha.(0,\vec{r}).q_\alpha^*$$

(4

where q_α^* is the conjugate of q_α and is defined as:

$$q_\alpha^*=\left(\cos\frac{\alpha}{2},-\vec{u}\sin\frac{\alpha}{2}\right)$$

We can also define the components of the unit quaternion in terms of Euler angles:

$$\tilde{e}_0=\cos\frac{\theta}{2}\cos\frac{\phi+\psi}{2}$$

$$\tilde{e}_1=\sin\frac{\theta}{2}\cos\frac{\phi-\psi}{2}$$

$$\tilde{e}_2=\sin\frac{\theta}{2}\sin\frac{\phi-\psi}{2}$$

$$\tilde{e}_3=\cos\frac{\theta}{2}\sin\frac{\phi+\psi}{2}$$

Rotational Motion: Method of Quaternions

In this way, the 3D orientation of the body-fixed frame is mapped into the components of the unit quaternion.

Therefore, quaternions are capable of representing the orientation of a body in 3D space by four parameters (contrary to the transformation matrix with nine parameters).

The rotation matrix A can be expressed in terms of components of the unit quaternion:

$$A = \begin{bmatrix} \tilde{e}_0^2 + \tilde{e}_1^2 - \tilde{e}_2^2 - \tilde{e}_3^2 & 2\left(\tilde{e}_1\tilde{e}_2 + \tilde{e}_0\tilde{e}_3\right) & 2\left(\tilde{e}_1\tilde{e}_3 - \tilde{e}_0\tilde{e}_2\right) \\ 2\left(\tilde{e}_1\tilde{e}_2 - \tilde{e}_0\tilde{e}_3\right) & \tilde{e}_0^2 - \tilde{e}_1^2 + \tilde{e}_2^2 - \tilde{e}_3^2 & 2\left(\tilde{e}_2\tilde{e}_3 + \tilde{e}_0\tilde{e}_1\right) \\ 2\left(\tilde{e}_1\tilde{e}_3 + \tilde{e}_0\tilde{e}_2\right) & 2\left(\tilde{e}_2\tilde{e}_3 - \tilde{e}_0\tilde{e}_1\right) & \tilde{e}_0^2 - \tilde{e}_1^2 - \tilde{e}_2^2 + \tilde{e}_3^2 \end{bmatrix}$$

To transform an arbitrary vector, r, between the space-fixed frame and the body-fixed frame, one can use the transformation matrix or use the quaternion and its conjugate:

$$\vec{r}^{\,b} = A\vec{r}^{\,s}$$

$$\left(0, \vec{r}^{\,b}\right) = q^*.\left(0, \vec{r}^{\,s}\right).q$$

and from the body-fixed frame to the space-fixed frame:

$$\vec{r}^{\,s} = A^T\vec{r}^{\,b}$$

$$\left(0, \vec{r}^{\,s}\right) = q.\left(0, \vec{r}^{\,b}\right).q^*$$

In addition to the transformation relations, we also need a time derivative of the quaternion to update the instantaneous orientation of the body-fixed frame. The time derivative of quaternion is given by:

$$\dot{q} = \frac{1}{2}q.\left(0, \vec{\omega}^b\right) = \frac{1}{2}\begin{bmatrix} -\tilde{e}_1\omega_x^b - \tilde{e}_2\omega_z^b - \tilde{e}_3\omega_z^b \\ \tilde{e}_0\omega_x^b + \tilde{e}_2\omega_z^b - \tilde{e}_3\omega_y^b \\ \tilde{e}_0\omega_y^b - \tilde{e}_1\omega_z^b + \tilde{e}_3\omega_x^b \\ \tilde{e}_0\omega_z^b + \tilde{e}_1\omega_y^b - \tilde{e}_2\omega_x^b \end{bmatrix}$$

Multi Spherical Particles:

1. The multi-sphere method is another method for generating non-spherical particles.

1. In spite of superellipsoids, which are non-spherical axisymmetric particles, the multi-sphere method can generate particles with irregular shapes.

1. Based on this method, spheres with smaller size are glued to each other to approximately fill the volume/surface of the real particle.

1. These spheres are allowed to overlap to any extent. In this way, any number of spheres, with different sizes and overlaps, can be used.

1. After positioning spheres in the non-spherical particle and producing the final shape, the relative position of spheres within the particle will not change.

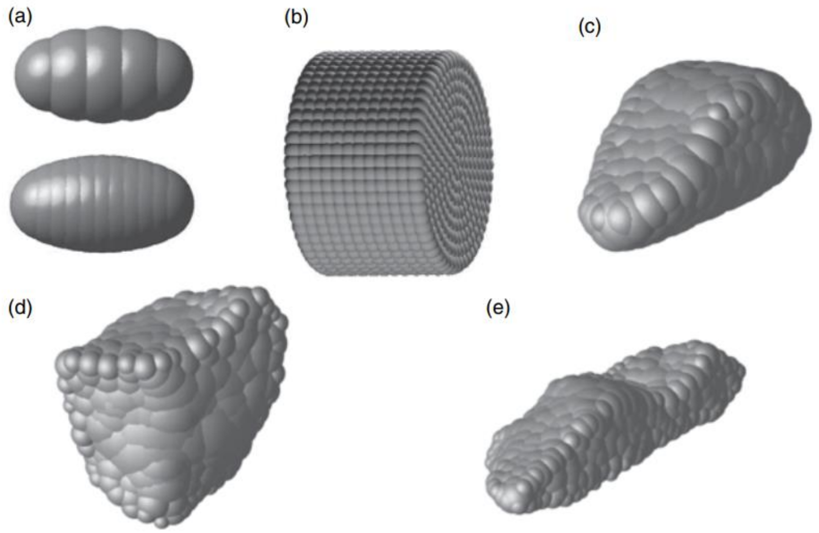
1. Thus, the dynamics of the rigid body can be applied to multi-sphere particles.

The particle shape/surface can be more accurately approximated by increasing the number of spheres and reduce the level of bumpiness of the surface.

On the other hand, the computational cost of simulation also increases with increasing the number of spheres.

Although the accuracy of representation of the surface is improved by using a large number of spheres, the mechanical behavior of the particle is not necessarily improved due to additional dominant errors introduced with increasing the number of spheres in the particle.

We are not still definite about an optimal number of spheres as it changes from one case to another.



Multi Spherical Particles:

$$\vec{r}_{ik}^{\,s} = \vec{r}_{CM,i}^{\,s} + A^T \vec{r}_{ik}^{\,b}$$

$$\vec{v}_{ik}^{\,s} = \vec{v}_{CM,i}^{\,s} + A^T \left(\vec{\omega}_i^{\,b} \times \vec{r}_{ik}^{\,b}\right)$$

The net contact force acting the particle, $\vec{f}_{cont,i}^{\,s}$, is sum of contact forces acting on all spheres in that particle:

$$\vec{f}_{cont,i}^{\,s} = \sum_{jl \in CL_i} \vec{f}_{cont,ik,jl}^{\,s} \tag{4.87}$$

where $\vec{f}_{cont,ik,jl}^{\,s}$ is the contact force between sphere *ik* in particle *i* and sphere *jl* in particle *j*. Since the contact force acts on the surface and it is considered on the center of mass of the particle, a torque should be considered around this point. The net contact torque on particle in the space-fixed frame is obtained by:

$$\vec{M}_{cont,i}^{\,s} = \sum_{jl \in CL_i} \vec{R}_{C,ik,jl}^{\,s} \times \vec{f}_{cont,ik,jl}^{\,s} \tag{4.88}$$

For solving linear and rotational equations of motion, the total mass of particle and principal moments of inertia should be known. It is common to calculate these values from properties of the spheres inside the particle. The following equations are used to obtain the total mass and principal moments of inertia of a multi-sphere particle:

$$m_i = \sum_{k=1}^{NS} m_{ik} \tag{4.89}$$

$$\hat{I}_i = \begin{pmatrix} \sum I_{ik} + \sum m_{ik}(y_{ik}^2 + z_{ik}^2) & 0 & 0 \\ 0 & \sum I_{ik} + \sum m_{ik}(x_{ik}^2 + z_{ik}^2) & 0 \\ 0 & 0 & \sum I_{ik} + \sum m_{ik}(x_{ik}^2 + y_{ik}^2) \end{pmatrix} \tag{4.90}$$

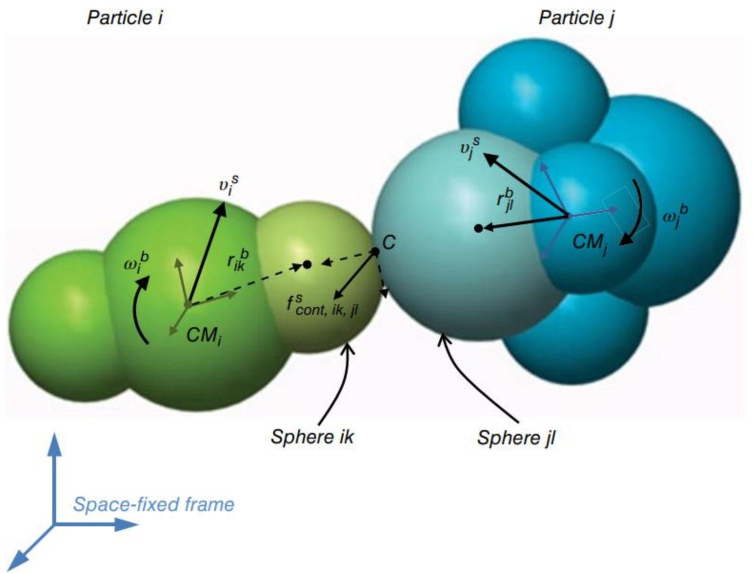


Figure 4.13 Two non-spherical particles *i* and *j* are in contact via their constituent spheres *ik* and *jl*

Contact Force in Multi-spherical particles:

$$\vec{f}^{\,s}_{cont,ij} = \frac{1}{NCP_{ij}} \sum \vec{f}^{\,s}_{cont,ik,jl}$$

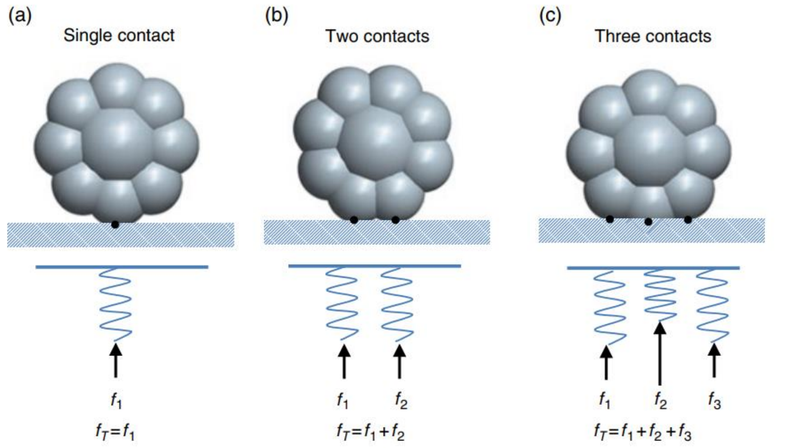


Figure 4.15 The schematic of possible contact conditions between a multi-sphere particle and a flat wall. With more contact points between particle and wall, the equivalent spring would be stiffer

The Net Force between the Particles due to all the collisions is the sum of Individual Forces acting on the individual spheres.

Implementation in LIGGHTS

Variable Description:

xcm - Position of Center of Mass
vcm - Velocity of Center of Mass
fcm - Net Force Acting on Center of Mass
torquecm - Net Torque on CoM

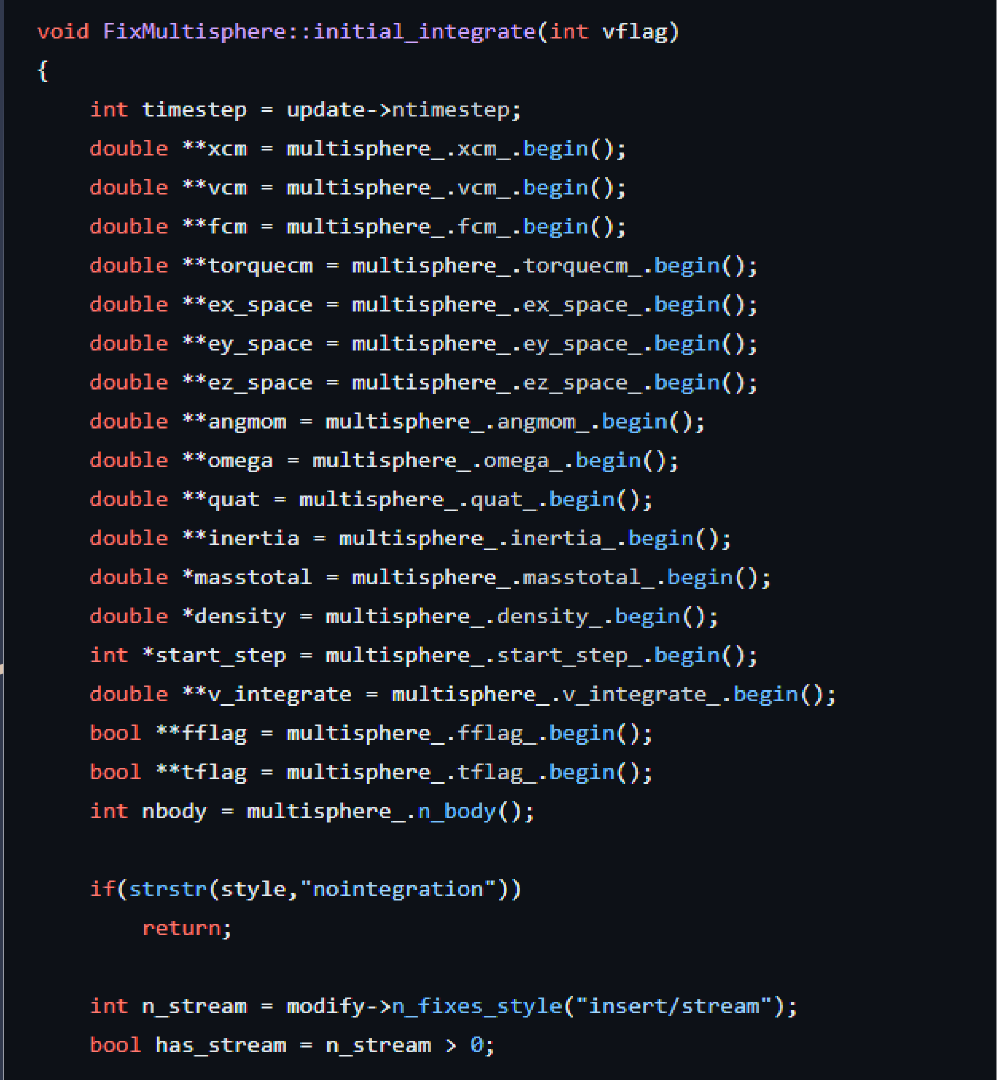
ex_space, ey_space, ez_space -
Vectors representing the rows of Rotation Matrix A which describes the orientation of the Body Particle

angmom - Angular Momentum at that instant

omega - Angular Velocity at that instant

quat - Quaternion representing the Orientation of Body in the Space Fixed Frame.

inertia - Inertia Tensor



```
void FixMultisphere::initial_integrate(int vflag)
{
    int timestep = update->ntimestep;
    double **xcm = multisphere_.xcm_.begin();
    double **vcm = multisphere_.vcm_.begin();
    double **fcm = multisphere_.fcm_.begin();
    double **torquecm = multisphere_.torquecm_.begin();
    double **ex_space = multisphere_.ex_space_.begin();
    double **ey_space = multisphere_.ey_space_.begin();
    double **ez_space = multisphere_.ez_space_.begin();
    double **angmom = multisphere_.angmom_.begin();
    double **omega = multisphere_.omega_.begin();
    double **quat = multisphere_.quat_.begin();
    double **inertia = multisphere_.inertia_.begin();
    double *masstotal = multisphere_.masstotal_.begin();
    double *density = multisphere_.density_.begin();
    int *start_step = multisphere_.start_step_.begin();
    double **v_integrate = multisphere_.v_integrate_.begin();
    bool **fflag = multisphere_.fflag_.begin();
    bool **tflag = multisphere_.tflag_.begin();
    int nbody = multisphere_.n_body();

    if(strstr(style,"nointegration"))
        return;

    int n_stream = modify->n_fixes_style("insert/stream");
    bool has_stream = n_stream > 0;
```

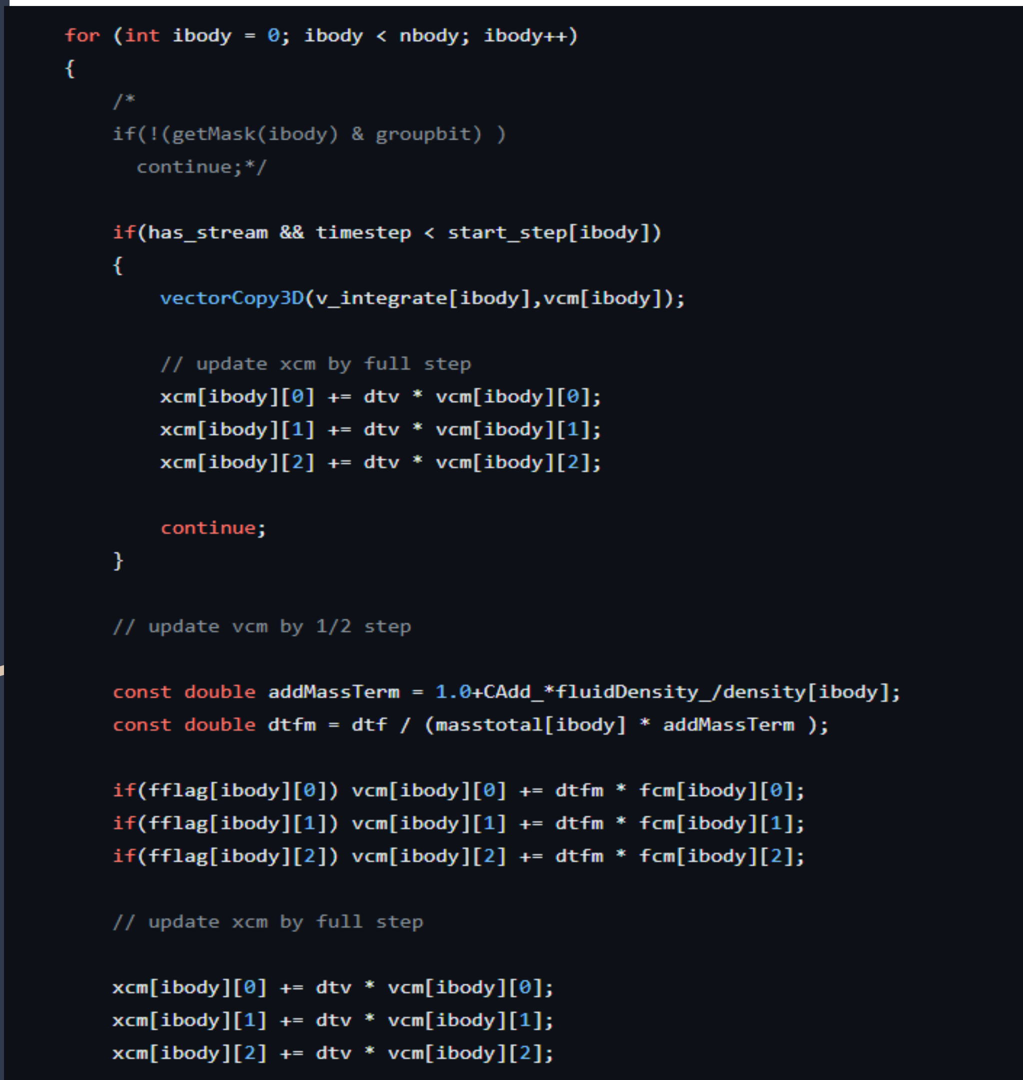
Implementation in LIGGHTS

Algorithm:

1. Compute the updated Center of Mass Position using the Translational Motion of the Body.

1. Then update the Velocity using the Net Translational Acceleration acting on the Body.

1. Update the New Angular Momentum of the Body using Net Torque in the Space Fixed Frame



```
for (int ibody = 0; ibody < nbody; ibody++)
{
    /*
    if(!(getMask(ibody) & groupbit) )
      continue;*/

    if(has_stream && timestep < start_step[ibody])
    {
        vectorCopy3D(v_integrate[ibody],vcm[ibody]);

        // update xcm by full step
        xcm[ibody][0] += dtv * vcm[ibody][0];
        xcm[ibody][1] += dtv * vcm[ibody][1];
        xcm[ibody][2] += dtv * vcm[ibody][2];

        continue;
    }

    // update vcm by 1/2 step

    const double addMassTerm = 1.0+CAdd_*fluidDensity_/density[ibody];
    const double dtfm = dtf / (masstotal[ibody] * addMassTerm );

    if(fflag[ibody][0]) vcm[ibody][0] += dtfm * fcm[ibody][0];
    if(fflag[ibody][1]) vcm[ibody][1] += dtfm * fcm[ibody][1];
    if(fflag[ibody][2]) vcm[ibody][2] += dtfm * fcm[ibody][2];

    // update xcm by full step

    xcm[ibody][0] += dtv * vcm[ibody][0];
    xcm[ibody][1] += dtv * vcm[ibody][1];
    xcm[ibody][2] += dtv * vcm[ibody][2];
```

Implementation in LIGGGHTS

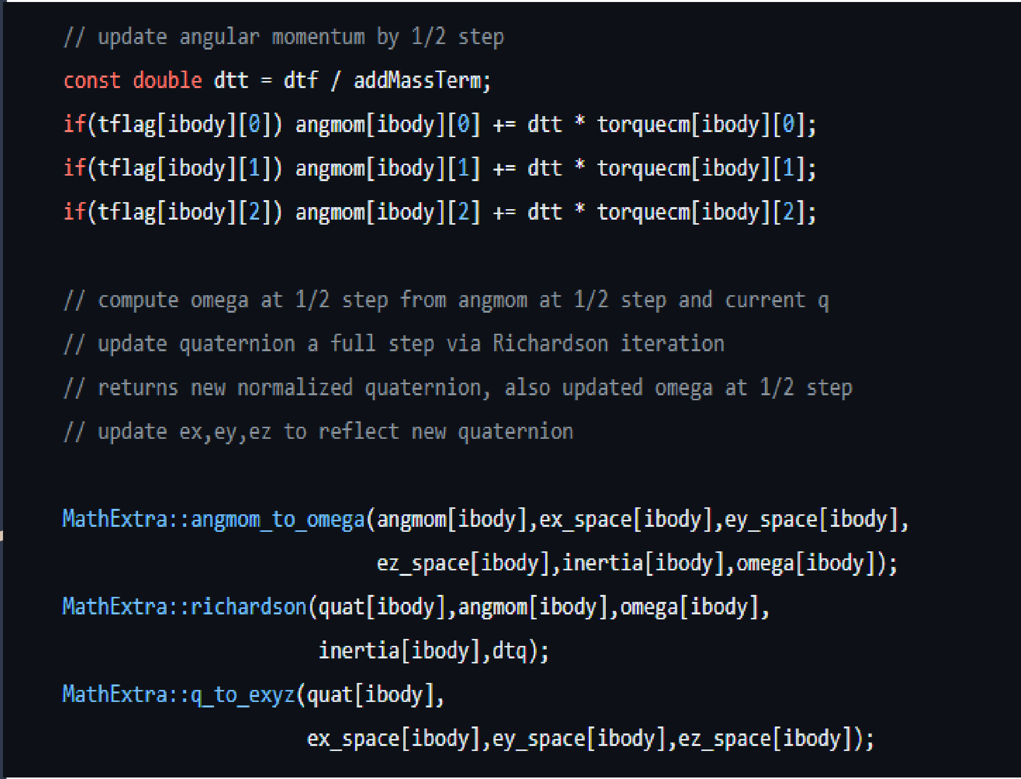
Algorithm:

1. Using the New Angular Momentum Calculated in the Space Fixed Frame calculate the New Angular Velocity in the Space Fixed Frame. Now convert this Angular Velocity in the Space Fixed Frame to the Body fixed Frame Angular Velocity using the vectors ex, ey, ez of the Rotation Matrix.

   This is done in the function:

   angmom_to_omega()

1. Then using Richardson Integration update the Quaternion by calculating the derivative of the quaternion using the body frame angular velocity. (richardson())



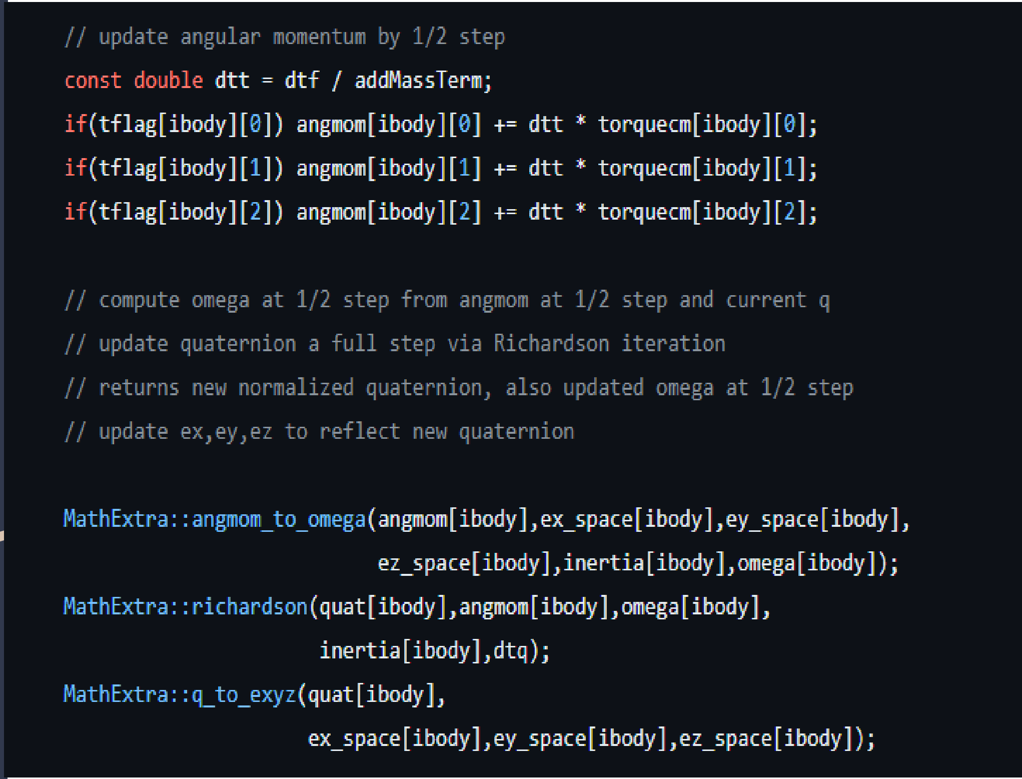
```
// update angular momentum by 1/2 step
const double dtt = dtf / addMassTerm;
if(tflag[ibody][0]) angmom[ibody][0] += dtt * torquecm[ibody][0];
if(tflag[ibody][1]) angmom[ibody][1] += dtt * torquecm[ibody][1];
if(tflag[ibody][2]) angmom[ibody][2] += dtt * torquecm[ibody][2];

// compute omega at 1/2 step from angmom at 1/2 step and current q
// update quaternion a full step via Richardson iteration
// returns new normalized quaternion, also updated omega at 1/2 step
// update ex,ey,ez to reflect new quaternion

MathExtra::angmom_to_omega(angmom[ibody],ex_space[ibody],ey_space[ibody],
                           ez_space[ibody],inertia[ibody],omega[ibody]);
MathExtra::richardson(quat[ibody],angmom[ibody],omega[ibody],
                      inertia[ibody],dtq);
MathExtra::q_to_exyz(quat[ibody],
                     ex_space[ibody],ey_space[ibody],ez_space[ibody]);
```

Implementation in LIGGHTS

Algorithm:

Now compute the New Rotation Matrix for the New Orientation of the Body which has been represented by the Newly calculated Quaternion in the richardson() function.

This step is done using the q_to_exyz() Function.



```
// update angular momentum by 1/2 step
const double dtt = dtf / addMassTerm;
if(tflag[ibody][0]) angmom[ibody][0] += dtt * torquecm[ibody][0];
if(tflag[ibody][1]) angmom[ibody][1] += dtt * torquecm[ibody][1];
if(tflag[ibody][2]) angmom[ibody][2] += dtt * torquecm[ibody][2];

// compute omega at 1/2 step from angmom at 1/2 step and current q
// update quaternion a full step via Richardson iteration
// returns new normalized quaternion, also updated omega at 1/2 step
// update ex,ey,ez to reflect new quaternion

MathExtra::angmom_to_omega(angmom[ibody],ex_space[ibody],ey_space[ibody],
                           ez_space[ibody],inertia[ibody],omega[ibody]);
MathExtra::richardson(quat[ibody],angmom[ibody],omega[ibody],
                      inertia[ibody],dtq);
MathExtra::q_to_exyz(quat[ibody],
                     ex_space[ibody],ey_space[ibody],ez_space[ibody]);
```