Animation & CGI Motion: Rigid Body Simulation

Theme 3 Milestone 1

Academic Honesty Policy

You are permitted and encouraged to discuss your work with other students. You may work out equations in writing on paper or a whiteboard. You are encouraged to use the discussion boards to converse with other students, the TA, and the instructor.

HOWEVER, you may NOT share source code or hardcopies of source code. Refrain from activities or the sharing of materials that could cause your source code to APPEAR TO BE similar to another student's enrolled in this course. We will be monitoring source code for individuality. Cheating will be dealt with severely. Cheaters will be punished. Source code should be yours and yours only. Do not cheat.

1 Introduction

In Theme III we will explore methods for simulating rigid bodies – objects in which the distance between any two points is fixed for all time. Rigid bodies have a number of interesting applications ranging from their use in robot motion planning to studying the time evolution of tops. Rigid bodies are also one of the most common primitives simulated in video game physics engines.

Rigid bodies see heavy use as they posses many favorable numeric properties; simulating similar systems with masses and springs or finite elements will lead to very stiff systems and hence strict time step limits. As a *reduced coordinate* representation, rigid bodies also have the potential to yield large savings due to the reduction in the number of degrees of freedom.

The use of *reduced* or *generalized* coordinates can be viewed as a constraint enforcement method. For example, one can encode the position of a pendulum with one variable that changes with time - the deflection of the pendulum arm from the vertical. The position of the pendulum bob can be recovered using simple trigonometry, the angle of deflection, and the length of the arm. By construction, the inextensibility of the arm cannot be violated. In contrast, if one were to use cartesian coordinates, the inextensibility of the constraint would have to be enforced in some manner. Usually reduced coordinates are difficult to write down for complex constrained systems, but for rigid bodies they prove both manageable and advantageous.

In this first milestone we will explore the dynamics of rigid bodies in the absence of collisions.

2 New XML Features

This milestone introduces the following new simulation commands:

1. The *simtype* node specifies the type of this simulation:

   ```
   <simtype type="rigid-body"/>
   ```

   The valid values for *type* are "particle-system" and "rigid-body". Setting the type to "particle-system" indicates that this scene file describes a particle simulation as implemented in the previous two themes. In this theme, we will use a setting of "rigid-body".

2. The *rigidbodyintegrator* node specifies both the time integration technique and the time-step to use for this rigid body simulation:

   ```
   <rigidbodyintegrator type="explicit-euler" dt="0.001"/>
   ```

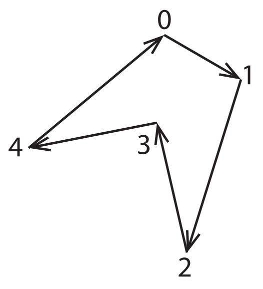

Figure 1: The hull of a rigid body. Note the clockwise specification of vertices.

The *type* attribute specifies which time integration technique to use: valid values are "explicit-euler" and "symplectic-euler". The *dt* attribute specifies the time-step to use. *dt* expects a positive scalar.

3. The *rigidbodyvertex* node creates a new vertex that the user can assign to a rigid body:

   ```
   <rigidbodyvertex x="-5" y="-1" m="1"/>
   ```

   The *x* and *y* attributes specify the position of the vertex. Both *x* and *y* expect scalar values. The *m* attribute specifies the mass of the vertex and must be a positive scalar.

4. The *rigidbody* node creates a new rigid body:

   ```
   <rigidbody p="0" p="1" p="2" p="3" vx="0.0" vy="0.0" omega="0.0" r="0.1"/>
   ```

   Each *p* attribute references a *rigidbodyvertex* and adds a new vertex to this *rigidbody*'s hull. The vertices that define the (generally non-convex) hull of this *rigidbody* are specified in clockwise order. The first and final vertex are also connected. See Figure 1. The center of mass, the total mass, and the moment of inertia of the rigid body are computed from the positions and masses of these vertices. The *vx* and *vy* attributes specify the initial velocity of the center of mass of this *rigidbody.* The *omega* attribute specifies the initial angular velocity about the center of mass of this *rigidbody.* The *r* attribute specifies the radius of this *rigidbody* for rendering and collision-detection purposes.

5. The *rigidbodyspringforce* node creates a spring force that acts on rigid bodies:

   ```
   <rigidbodyspringforce i="0" pix="1.0" piy="1.0" j="1" pjx="-1.0" pjy="1.0" k="1.0" l0="4.0"/>
   ```

   The *i* and *j* nodes specify which rigid bodies this force acts between. If either value is set to -1, this indicates that the corresponding endpoint is fixed in space and not attached to a rigid body. If an endpoint is attached to a rigid body, the *pix*, *piy*, *pjx*, and *pjy* nodes specify where on each rigid body the spring is attached relative to the center of mass (assuming the original orientation). If instead the corresponding endpoint is specified as fixed, these values specify the position in space where the spring endpoint is fixed to. The *k* node specifies the stiffness of the spring and the *l0* node specifies the rest-length of the spring.

6. The *rigidbodygravityforce* node creates a (constant near-earth) gravity force that acts on all rigid bodies:

   ```
   <rigidbodygravityforce fx="0.0" fy="-9.81"/>
   ```

   The *fx* and *fy* attributes specify the *x* and *y* components of the force, respectively.

7. The *rigidbodywindforce* node creates a wind-like force that acts on all rigid bodies:

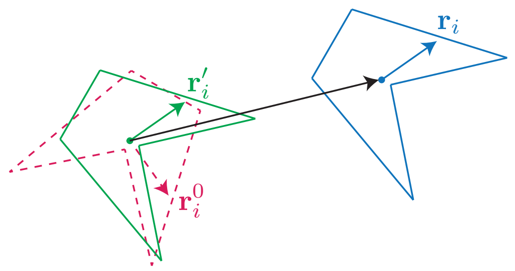

Figure 2: The *body space* and *world space* frames. $\mathbf{r}'_i = \mathbf{R}(t)\mathbf{r}_i^0$ and $\mathbf{r}_i = \mathbf{R}(t)\mathbf{r}_i^0 + \mathbf{X}(t)$.

```
<rigidbodywindforce beta="1.0" fx="1.0" fy="0.0" pointsperedge="2" />
```

The positive scalar *beta* attribute scales the overall strength of the force. The *fx* and *fy* attributes specify the x and y components of the force, respectively. The integer *pointsperedge* attributes specifies the number of quadrature points per edge to use when computing this force (see description below).

This milestone introduces the following new rendering commands:

1. The *rigidbodycolor* node sets the color of an entire rigid body:

   ```
   <rigidbodycolor body="1" r="0.0" g="0.4" b="0.0"/>
   ```

   The *body* attribute specifies which rigid body this color applies to. The r, g, and b nodes specify the color of the body. Their values must be between 0 and 1.

2. The *rigidbodyspringcolor* node sets the color of a spring:

   ```
   <rigidbodyspringcolor spring="4" r="0.2" g="0.0" b="0.8"/>
   ```

   The *spring* attribute specifies which spring this color applies to. The r, g, and b nodes specify the color of the spring. Their values must be between 0 and 1.

3 Rigid Body Dynamics

A rigid body is a collection of masses (or a continuum, but here we consider only a finite number of points) in which the distance between any two masses remains fixed for all time. Intuitively, the two transformations that preserve this invariant are translations and rotations. Therefore, not surprisingly, we will find that we can decompose the dynamics of a rigid body into two components: a component associated with the center of mass that behaves like a point mass, and a component associated with rotations about the center of mass.

3.1 Body Space, World Space, and Center of Mass

Consider a 2D rigid body composed of N points masses lying in a 2D plane. Recall that we define the center of mass of a collection of masses as

$$\mathbf{X} = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i} = \frac{\sum_i m_i \mathbf{r}_i}{M}$$

where m_i is the mass of the i^{th} particle, $\mathbf{r}_i$ is the position in *world space* of the i^{th} particle, and $M = \sum_i m_i$ is the total mass.

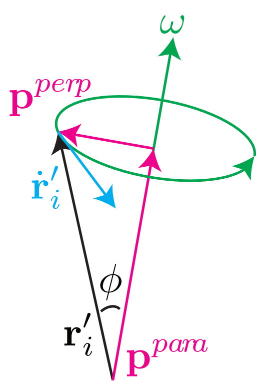

Figure 3: A rigid body rotating about its center of mass and axis ω.

Define *body space* as a 2D orthonormal coordinate system with the origin placed at the body's center of mass and in which the body is considered to have no rotation (see Figure 2). If $\mathbf{r}_i^0$ is the position of any point on the rigid body in *body space*, then the point's position in *world space* is given by

$$\mathbf{r}_i = \mathbf{R}(t)\mathbf{r}_i^0 + \mathbf{X}(t) \tag{1}$$

where $\mathbf{R}(t)$ is a 2×2 rotation matrix describing the orientation of the rigid body at time t, and $\mathbf{X}(t)$ is the position of the body's center of mass at time t. As $\mathbf{R}$ is purely a rotation, we can simply store a single angle θ to reconstruct $\mathbf{R}$.

What does this construction gain us? At any time t, if we know the orientation of the body θ and the center of mass' position $\mathbf{X}$, we can compute the position of any particle on the rigid body. Instead of simulating $2N$ degrees of freedom, we need only simulate 3. Furthermore, as our representation only admits rigid motions, that is rotations and translations, we do not need to enforce the rigidity constraint with soft constraints (e.g. stiff forces), hard constraints (e.g. Lagrange multipliers), or some other technique.

Before we can simulate a rigid body with this *reduced coordinate* representation, however, we need to ascertain how the velocity of a particle on the body relates to this reduced representation, and how forces act on this reduced representation.

3.2 Linear and Angular Velocity

We can compute the velocity of a particle attached to a rigid body by taking the time derivative of (1):

$$\dot{\mathbf{r}}_i = \dot{\mathbf{R}}(t)\mathbf{r}_i^0 + \dot{\mathbf{X}}(t)$$

Note that $\mathbf{r}_i^0$ is fixed throughout the simulation and thus its time derivative is 0. $\dot{\mathbf{X}}$ is simply the velocity of the rigid body's center off mass, but how do we compute $\dot{\mathbf{R}}$? Before proceeding, let us examine the columns of $\mathbf{R}$.

Component-wise, we can write $\mathbf{R}$ as:

$$\begin{pmatrix} R_{xx} & R_{yx} \\ R_{xy} & R_{yy} \end{pmatrix}$$

Consider the action of $\mathbf{R}$ on the cartesian x-axis:

$$\mathbf{R}\hat{\mathbf{x}} = \begin{pmatrix} R_{xx} & R_{yx} \\ R_{xy} & R_{yy} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} R_{xx} \\ R_{xy} \end{pmatrix}$$

That is, the first column of $\mathbf{R}$ is simply the *body space* x-axis rotated into world space! Similarly, we find that the second column of $\mathbf{R}$ is the *body space* y-axis rotated into world space. Therefore, if we can compute how these vectors evolve instantaneously, we can compute $\dot{\mathbf{R}}$. In order to compute these quantities, we will first derive a more general 3D result.

Consider a rigid body rotating about its center of mass and some axis ω with angular velocity $\omega = |\omega|$ (see Figure 3). Consider a vector pointing from the center of mass to some point on the body, say $\mathbf{r}'_i = \mathbf{R}\mathbf{r}^0_i = (\mathbf{r}_i - \mathbf{X}) = \mathbf{p}^{perp} + \mathbf{p}^{para}$, where $\mathbf{p}^{perp}$ is perpendicular to ω and $\mathbf{p}^{para}$ is parallel to ω. $\mathbf{p}^{para}$ is unaffected by the rotation, while $\mathbf{p}^{perp}$ will trace out a circle. Simple trigonometry gives that radius of the circle is $|\mathbf{r}_i - \mathbf{X}| \sin\phi$, where ϕ is the angle between $(\mathbf{r}_i - \mathbf{X})$ and ω. Thus, the instantaneous speed of any point on this circle is given by $|\omega||\mathbf{r}_i - \mathbf{X}| \sin\phi$. Furthermore, the direction of the velocity of this point is perpendicular to both ω and $(\mathbf{r}_i - \mathbf{X})$. What operation gives this magnitude and this direction? The cross product! Therefore, the velocity of $\mathbf{r}_i - \mathbf{X}$ is given by $\omega \times (\mathbf{r}_i - \mathbf{X})$.

As a notational convenience, note that we can represent a cross product as a matrix-vector multiplication:

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix} = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix} \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \mathbf{a}^* \mathbf{b}$$

Returning to the problem of computing $\dot{\mathbf{R}}$, first note that translations of the rigid body do not change $\mathbf{R}$, and thus we need only consider changes to $\mathbf{R}$ due to the angular velocity. Invoking the above result:

$$\dot{\mathbf{R}} = \begin{pmatrix} \omega \times \begin{pmatrix} R_{xx} \\ R_{xy} \end{pmatrix} & \omega \times \begin{pmatrix} R_{yx} \\ R_{yy} \end{pmatrix} \end{pmatrix} = \omega^* \mathbf{R}$$

Thus, the *world space* velocity of i^{th} particle in the rigid body is $\dot{\mathbf{r}}_i = \omega(t)^* \mathbf{R}(t) \mathbf{r}^0_i + \dot{\mathbf{X}}(t)$. That is, if in addition to $\mathbf{X}(t)$ and $\theta(t)$ we know $\dot{\mathbf{X}}(t)$ and $\omega(t)$, we can recover the velocity of any point on the rigid body! Therefore, we can represent the entire state of the rigid body with these four quantities.

If we wanted to simulate rigid bodies without any forces, the above would be enough; $\dot{\mathbf{X}}$ and ω would remain constant throughout the simulation, and we would just update $\mathbf{X}(t)$ and $\theta(t)$ using these constant values. This situation is not terribly interesting, however, so we will now see how Newton's second law ($\mathbf{F} = m\mathbf{a}$) looks when phrased in terms of our reduced coordinates.

3.3 Forces and Torques

3.3.1 Center of Mass Acceleration

To determine how a force effects the center of mass' velocity, $\dot{\mathbf{X}}$, we will start by computing the total momentum of the rigid body. The total momentum $\mathbf{P}$ of a rigid body is given by:

$$\mathbf{P} = \sum_i m_i \dot{\mathbf{r}}_i = (\sum_i m_i) \frac{\sum_i m_i \dot{\mathbf{r}}_i}{\sum_i m_i} = M\dot{\mathbf{X}}$$

That is, the momentum of the system can be computed by viewing all of the body's mass as moving with the center of mass. Taking the derivative of this quantity gives us the center of mass' acceleration:

$$M\ddot{\mathbf{X}} = \sum_i m_i \ddot{\mathbf{r}}_i = \sum_i \mathbf{f}_i$$

$\mathbf{f}_i$ is the total force acting on mass i. We have to be a little careful going forward, as this force includes both external forces (e.g. gravity) as well as the internal forces acting within the rigid body. Let us assume that each point j exerts a force on each other point i to maintain the rigidity constraint. That is, we can write the force on particle i as $\mathbf{f}_i = \mathbf{f}^{ext}_i + \sum_{j \neq i} \mathbf{f}_{ij}$ where $\mathbf{f}^{ext}_i$ is the sum of the external forces acting on i, and $\mathbf{f}_{ij}$ is the internal force on i from point j. Substituting this sum into the rate of change of momentum:

$$M\ddot{\mathbf{X}} = \sum_i \mathbf{f}_i = \sum_i (\mathbf{f}^{ext}_i + \sum_{j \neq i} \mathbf{f}_{ij}) = \sum_i \mathbf{f}^{ext}_i + \sum_i \sum_{j \neq i} \mathbf{f}_{ij}$$

Let us examine $\sum_i \sum_{j \neq i} \mathbf{f}_{ij}$ in more detail. We can rewrite this sum and invoke Newton's third law ('for every action there is an opposite and equal reaction'), giving:

$$\sum_i \sum_{j \neq i} \mathbf{f}_{ij} = \sum_i \sum_{j>i} (\mathbf{f}_{ij} + \mathbf{f}_{ji}) = \sum_i \sum_{j>i} (\mathbf{f}_{ij} - \mathbf{f}_{ij}) = 0$$

That is, provided the internal constraint forces obey Newton's third law, to compute the acceleration of the center of mass, we simply sum the external forces acting on each particle of the body:

$$M\ddot{\mathbf{X}} = \sum_i \mathbf{f}_i^{ext} = \mathbf{F}$$

3.3.2 Angular Acceleration

To determine the angular acceleration of a rigid body, we will first compute the time derivative of the angular momentum measured with respect to the origin. The total angular momentum of rigid body is given by

$$\mathbf{L} = \sum_i \mathbf{r}_i \times \mathbf{p}_i$$

and thus we compute the time derivative as:

$$\dot{\mathbf{L}} = \sum_i \dot{\mathbf{r}}_i \times \mathbf{p}_i + \sum_i \mathbf{r}_i \times \dot{\mathbf{p}}_i$$

In the first term, the velocity of a particle is parallel to its momentum, so this term is 0. In the second term, $\dot{\mathbf{p}}_i = \mathbf{f}_i = \mathbf{f}_i^{ext} + \sum_{j \neq i} \mathbf{f}_{ij}$ is the total force acting on the point, which as in the previous section includes both the external forces on the point and the internal constraint-maintaining forces. Making this substitution, we find:

$$\dot{\mathbf{L}} = \sum_i \mathbf{r}_i \times \dot{\mathbf{p}}_i = \sum_i \mathbf{r}_i \times (\mathbf{f}_i^{ext} + \sum_{j \neq i} \mathbf{f}_{ij}) = \sum_i \mathbf{r}_i \times \mathbf{f}_i^{ext} + \sum_i \mathbf{r}_i \times \sum_{j \neq i} \mathbf{f}_{ij}$$

Let us examine the second term in more detail. We can rearrange the sum and invoke Newton's third law to obtain:

$$\sum_i \mathbf{r}_i \times \sum_{j \neq i} \mathbf{f}_{ij} = \sum_i \sum_{j \neq i} \mathbf{r}_i \times \mathbf{f}_{ij} = \sum_i \sum_{j>i} (\mathbf{r}_i \times \mathbf{f}_{ij} + \mathbf{r}_j \times \mathbf{f}_{ji}) = \sum_i \sum_{j>i} ((\mathbf{r}_i - \mathbf{r}_j) \times \mathbf{f}_{ij})$$

When is this sum always 0? When $\mathbf{r}_i - \mathbf{r}_j$ is parallel to $\mathbf{f}_{ij}$. That is, when all constraint forces are central and obey Newton's third law, the time rate of change in angular momentum is simply the sum of all external torques:

$$\dot{\mathbf{L}} = \sum_i \mathbf{r}_i \times \mathbf{f}_i^{ext}$$

We can further decompose the angular momentum into a component associated with the center of mass' motion and a component associated with motion (spin) about the center of mass. Let $\mathbf{r}'_i = R(t)\mathbf{r}_i^0$ denote the *body space* coordinate of particle i rotated into *world space* by the body's orientation. Substituting a particle's position expressed relative to the center of mass into the formula for angular momentum:

$$\begin{aligned}
\mathbf{L} &= \sum_i \mathbf{r}_i \times \mathbf{p}_i = \sum_i (\mathbf{X} + \mathbf{r}'_i) \times m_i(\dot{\mathbf{X}} + \dot{\mathbf{r}}'_i) \\
&= \sum_i \mathbf{X} \times m_i \dot{\mathbf{X}} + \sum_i \mathbf{X} \times m_i \dot{\mathbf{r}}'_i + \sum_i \mathbf{r}'_i \times m_i \dot{\mathbf{X}} + \sum_i \mathbf{r}'_i \times m_i \dot{\mathbf{r}}'_i \\
&= \mathbf{X} \times (\sum_i m_i)\dot{\mathbf{X}} + \mathbf{X} \times (\sum_i m_i \dot{\mathbf{r}}'_i) + (\sum_i m_i \mathbf{r}'_i) \times \dot{\mathbf{X}} + \sum_i \mathbf{r}'_i \times m_i \dot{\mathbf{r}}'_i \\
&= \mathbf{X} \times \mathbf{P} + \mathbf{X} \times (\sum_i m_i \dot{\mathbf{r}}'_i) + (\sum_i m_i \mathbf{r}'_i) \times \dot{\mathbf{X}} + \sum_i \mathbf{r}'_i \times \mathbf{p}'_i
\end{aligned}$$

Let us look more closely at the two terms in parenthesis. Expanding $\sum_i m_i \mathbf{r}'_i$ gives:

$$\sum_i m_i \mathbf{r}'_i = \sum_i m_i(\mathbf{r}_i - \mathbf{X}) = \sum_i m_i \mathbf{r}_i - \sum_i m_i \mathbf{X} = (\sum_i m_i)\frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i} - (\sum_i m_i)\mathbf{X} = M\mathbf{X} - M\mathbf{X} = 0$$

$\sum_i m_i \dot{\mathbf{r}}'_i$ is simply the time derivative of this function, and hence is also 0. Thus, the total angular momentum of a rigid body with respect to the origin is given by:

$$\mathbf{L} = \mathbf{X} \times \mathbf{P} + \sum_i \mathbf{r}'_i \times \mathbf{p}'_i = \mathbf{L}^{point} + \mathbf{L}^{spin}$$

Conveniently, the angular momentum decomposes into a component associated with the center of mass and the total momentum of the system, and a component associated with velocity relative to the center of mass. We will now compute $\dot{\mathbf{L}}^{point}$, and use this to obtain a simple formula for $\dot{\mathbf{L}}^{spin}$.

$$\dot{\mathbf{L}}^{point} = \dot{\mathbf{X}} \times \mathbf{P} + \mathbf{X} \times \dot{\mathbf{P}} = \mathbf{X} \times \mathbf{F}$$

Here we have used that $\dot{\mathbf{X}}$ is parallel to $\mathbf{P}$ and that $\dot{\mathbf{P}} = \mathbf{F}$ as proved in the previous section. We now compute $\dot{\mathbf{L}}^{spin}$ as

$$\begin{aligned}\dot{\mathbf{L}}^{spin} &= \dot{\mathbf{L}} - \dot{\mathbf{L}}^{point} = \sum_i \mathbf{r}_i \times \mathbf{f}_i^{ext} - \mathbf{X} \times \mathbf{F} \\ &= \sum_i (\mathbf{X} + \mathbf{r}'_i) \times \mathbf{f}_i^{ext} - \mathbf{X} \times \mathbf{F} = \mathbf{X} \times \sum_i \mathbf{f}_i^{ext} + \sum_i \mathbf{r}'_i \times \mathbf{f}_i^{ext} - \mathbf{X} \times \mathbf{F} \\ &= \mathbf{X} \times \mathbf{F} + \sum_i \mathbf{r}'_i \times \mathbf{f}_i^{ext} - \mathbf{X} \times \mathbf{F} = \sum_i \mathbf{r}'_i \times \mathbf{f}_i^{ext}\end{aligned}$$

This is a remarkable result! If we compute the angular momentum in the non-inertial (accelerating) reference frame of the center of mass, the rate of change of the angular momentum takes the simple form $\dot{\mathbf{L}}^{spin} = \sum_i \mathbf{r}'_i \times \mathbf{f}_i^{ext}$.

We now restrict ourselves to 2D, and without loss of generality consider a body undergoing only a rotational motion with instantaneous angular velocity ω (as we know how to compute torques in the non-inertial frame of the center of mass). We can compute the velocity of the i^{th} point as:

$$\dot{\mathbf{r}}'_i = \begin{vmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ 0 & 0 & \omega \\ x_i & y_i & 0 \end{vmatrix} = \omega \begin{pmatrix} -y_i \\ x_i \\ 0 \end{pmatrix}$$

The angular momentum of this particular point is given by (in 2D a scalar) $m_i \mathbf{r}'_i \times \dot{\mathbf{r}}'_i$. Expanding this cross product we find that:

$$m_i \mathbf{r}'_i \times \dot{\mathbf{r}}'_i = m_i \begin{vmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ x_i & y_i & 0 \\ -y_i & x_i & 0 \end{vmatrix} \omega = m_i \begin{pmatrix} 0 \\ 0 \\ x_i^2 + y_i^2 \end{pmatrix} \omega = m_i \mathbf{r}'_i \cdot \mathbf{r}'_i \omega$$

We can thus express the total angular momentum in the center of mass frame as

$$L^{spin} = \sum_i m_i \mathbf{r}'_i \times \dot{\mathbf{r}}'_i = \left(\sum_i m_i \mathbf{r}'_i \cdot \mathbf{r}'_i \right) \omega = I\omega$$

where $I = \sum_i m_i \mathbf{r}'_i \cdot \mathbf{r}'_i$ is called the moment of inertia. In 2D, this value is a constant throughout the simulation. To see that I is constant, rotate the entire body about the center of mass: $I = \sum_i m_i (\mathbf{R}\mathbf{r}'_i)^T \mathbf{R}\mathbf{r}'_i = \sum_i m_i (\mathbf{r}'_i)^T \mathbf{R}^T \mathbf{R} \mathbf{r}'_i = \sum_i m_i (\mathbf{r}'_i)^T \mathbf{r}'_i$, as a rotation is an orthogonal matrix ($\mathbf{R}^T = \mathbf{R}^{-1}$, and thus $\mathbf{R}^T\mathbf{R} = Id$).

In conclusion, we find that $\dot{\mathbf{L}}^{spin} = I\dot{\omega} = \sum_i \mathbf{r}'_i \times \mathbf{f}_i^{ext} = \Gamma$, which gives us a simple expression for $\dot{\omega}$. Note that here we rely on the fact that I is constant in time (a fact not true in 3D).

3.4 Equations of Motion

To summarize, the equations of motion for our reduced coordinate representation of a rigid body are given by:

$$\frac{d}{dt}\begin{pmatrix}\mathbf{X}\\ \theta\\ \dot{\mathbf{X}}\\ \omega\end{pmatrix} = \begin{pmatrix}\dot{\mathbf{X}}\\ \omega\\ \mathbf{F}/M\\ \Gamma/I\end{pmatrix}$$

In two dimensions, $\mathbf{X}$, $\dot{\mathbf{X}}$, and $\mathbf{F}$ are vectors of length 2. θ, ω, Γ, M, and I are scalars. From here, one can utilize his or her favorite time integration technique.

4 Required Features for Theme III

4.1 Computation of Total Mass, Center of Mass, Moment of Inertia

Please edit *RigidBodies/RigidBody.cpp* and complete the *computeTotalMass*, *computeCenterOfMass*, and *computeMomentOfInertia* methods. The interface for these methods is documented in the header file. The oracle will check and grade these values for all scenes. If your values are incorrect, the oracle will print the correct values.

4.2 Computation of Momentum

Please edit *RigidBodies/RigidBody.cpp* and complete the *computeTotalMomentum* method. The interface for this method is documented in the header file. The oracle will check and grade this value for all scenes. If your value is incorrect, the oracle will print the correct value.

4.3 Computation of Angular Momentum

In Section 3.3.2 we decomposed the total angular momentum of a rigid body into $\mathbf{L} = \mathbf{L}^{point} + \mathbf{L}^{spin}$. Please edit *RigidBodies/RigidBody.cpp* and complete the *computeCenterOfMassAngularMomentum* and *computeSpinAngularMomentum* methods. The interface for these methods is documented in the header file. The oracle will check and grade these values for all scenes. If your values are incorrect, the oracle will print the correct values.

4.4 Computation of Kinetic Energy Components

In Section 3.3.2, we separated the angular momentum into the components $L = L^{point} + L^{spring}$. Please derive a similar decomposition for the kinetic energy of a rigid body; that is, separate the kinetic energy into components $T = T^{point} + T^{spin}$. Please implement these functions in the *computeCenterOfMassKineticEnergy* and *computeSpinKineticEnergy* functions of *RigidBody.cpp*. These values are serialized and graded. If your values are incorrect, the oracle will print the correct values.

4.5 Explicit and Symplectic Euler

Please edit the *stepScene* methods of *RigidBodies/RigidBodyExplicitEuler.cpp* and *RigidBodies/RigidBodySymplecticEuler.cpp* to implement explicit Euler and symplectic Euler respectively. The documentation for these methods is included in the source files. After implementing these methods, you should be able to pass all tests under *theme3assets/inertiatests/*. As in the first theme, your symplectic Euler implementation should be implicit in velocity during the position update.

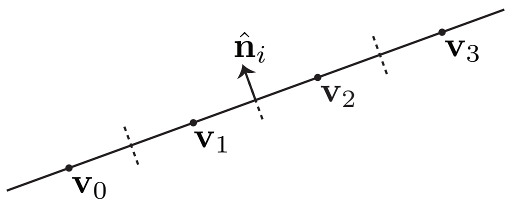

Figure 4: Subdivision of an edge for computing the wind force. Here the number of sample points is four.

4.6 Near-Earth Gravity Force

Please implement a force corresponding to the (pointwise) potential $U(\mathbf{r}_i) = -m_i \mathbf{g} \cdot \mathbf{r}_i$. As this force acts on all points of the rigid body, to simplify computations, work out the total potential energy, force, and torque on the rigid body due to this force:

$$\begin{aligned} U^{body} &= \sum_i U(\mathbf{r}_i) \\ \mathbf{F}^{body} &= \sum_i -\nabla U(\mathbf{r}_i) \\ \Gamma^{body} &= \sum_i \mathbf{r}_i' \times -\nabla U(\mathbf{r}_i) \end{aligned}$$

The resulting formulas are very simple and intuitive.

Please fill in the implementations of *computePotentialEnergy* and *computeForceAndTorque* in *RigidBodies/RigidBodyGravityForce.cpp* (note that the potential energy is serialized and graded). After implementing these methods, you should be able to pass all tests under *theme3assets/gravitytests/*.

4.7 Spring Force

Please implement a force corresponding to the potential $U(\mathbf{r}_i, \mathbf{r}_j) = \frac{1}{2}k(|\mathbf{r}_j - \mathbf{r}_i| - l_0)^2$, where $\mathbf{r}_i$ and $\mathbf{r}_j$ are either points on a rigid body or fixed points in space. If the endpoint is a rigid body, you will have to compute both the force and the corresponding torque. Please fill in the implementations of *computePotentialEnergy* and *computeForceAndTorque* in *RigidBodies/RigidBodySpringForce.cpp* (note that the potential energy is serialized and graded). The interfaces for these functions are documented in the corresponding header. After implementing these methods, you should be able to pass all tests under *theme3assets/springtests/*.

Note: Please be careful with the special case where the spring length is 0 and the rest length l_0 is 0 (can you see where in the force computation this will cause trouble?). You will have to explicitly check for this case and exert no force when it is detected.

4.8 Wind Force

We will implement a force that approximates the action of 'wind' on a rigid body. The magnitude of this force should scale with the difference between the wind's velocity and the velocity of some point on the edge in the normal direction, that is with $(\mathbf{v}^{wind} - \mathbf{v}_j) \cdot \hat{\mathbf{n}}_i$. The velocity along an edge of a rigid body is not constant, so we will have to subdivide the edge into at least two regions when computing the action of this force. See Figure 4. To summarize, given a rigid body with N edges and P sample points per edge:

$$\mathbf{F}^{wind} = \sum_{i=0}^{N-1} \sum_{j=0}^{P-1} \beta \frac{l_i}{P} (\mathbf{v}^{wind} - \mathbf{v}_j) \cdot \hat{\mathbf{n}}_i \hat{\mathbf{n}}_i$$

β scales the magnitude of the force, and l_i/P is the fraction of length assigned to each sample point of the i^{th} edge. As the vertices of the rigid body are specified in clockwise order, one can easily compute the outward normal $\hat{\mathbf{n}}_i$ for a given edge.

When implementing this force, you will have to compute the torque at each sample point on each edge. Please implement this force in the source file *RigidBodies/RigidBodyWindForce.cpp*.

5 Creative Scene

As part of your final submission for this milestone, please include a scene of your design that best shows off your program. Your scene will be judged by a secret of committee of top scientists using the highly refined criteria of:

1. How well the scene shows off this milestones magic ingredients (a la Iron Chef).
2. Aesthetic considerations. The more beautiful, the better.
3. Originality.

Top examples will be posted to the discussion board. Please remember to generate this scene before submitting, as the Codio box will become read-only after you submit.

5.1 Making Movies

Please remember to generate this scene before submitting, as the Codio box will become read-only after you submit. The FOSSSim starter code comes with a PNG outputting utility that can help you create movies from your simulation. To enable it, create a folder named "pngs" in the directory that you are calling your executable from (i.e. your current directory) and run your code with the

```
 -g 1
```

flag enabled. This will prompt your program to automatically save a PNG file for each frame of the simulation. This wont work if the "pngs" folder does not exist so you need to create it first before you run the program.

After the simulation finishes, you'll find all the frames in the pngs folder. Then you can make videos out of them with command-line tools such as mencoder and ffmpeg. Both mencoder and ffmpeg are easy-to-use tools available on the Codio boxes. You can execute this command:

```
mencoder mf://pngs/*.png -mf fps=24 -ovc lavc -lavcopts vcodec=msmpeg4v2 -oac copy -o output.avi
```

followed by:

```
ffmpeg -i output.avi -vcodec libx264 -crf 25 output.mp4
```

in order to create an mp4 video you may upload for the Creative portion of the assignment.

Note, ffmpeg can be used to create a video from the png images in one step, although sometimes this will give an error depending on the count and dimensions of input files. Nevertheless, the following command can take pngs and convert them to an mp4 movie in one:

```
ffmpeg -r 24 -f image2 -i ./pngs/frame%05d.png -vcodec libx264 -crf 25 -pix_fmt yuv420p test.mp4
```

Lastly, you may preview your movie by running the following command and looking at the virtual desktop:

```
mplayer <movie_name>
```

Please submit the mp4 file to the Peer Review Assignment portion of this week. An explanation for arguments to both mencoder and ffmpeg can be found online.

6 Food For Thought

Answers to these questions are not required, we have included them only for your personal edification.

1. Did we have to choose the center of mass to encode the body's translation? What would happen if we selected a different point in body space?
2. Can you identify where in this derivation we used the assumption of rigidity?

7 Complications in moving to 3D

The complexity of a rigid body simulation jumps significantly from 2D to 3D, be warned! Complications include:

1. The axis of rotation is now a three vector.
2. The moment of inertia is dependent on orientation (it is a *tensor*).
3. The derivative of angular momentum has extra terms because the inertia tensor changes over time.
4. The representation of orientation is no longer a single angle. There are many options, all with tradeoffs:
   (a) *Euler angles*: Simple (only three scalars), but suffers from 'gimbal lock' (who has watched *Apollo 13*?). There are also 24 possible conventions for representing orientation with Euler angles, and this ambiguity can be confusing.
   (b) *Rotation matrix*: Simple, but suffers from numeric drift, and requires the storage of nine scalars.
   (c) *Quaternion*: Requires only four scalars, easy to 'renormalize' to correct for drift. Renormalization is not free, however. Conceptually more difficult to work with.
   (d) *Exponential map*: No renormalization required. Updating orientations fit 'naturally' into time stepping routines such as explicit Euler.