GAMES103: Intro to Physics-Based Animation

Rigid Body Dynamics

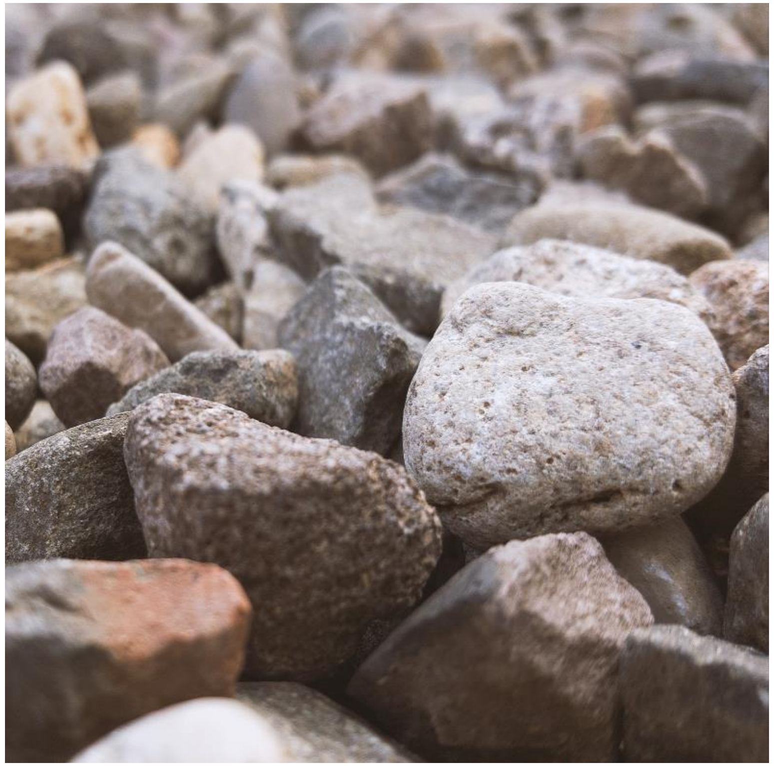
Huamin Wang

Nov 2021

What is rigid body dynamics?

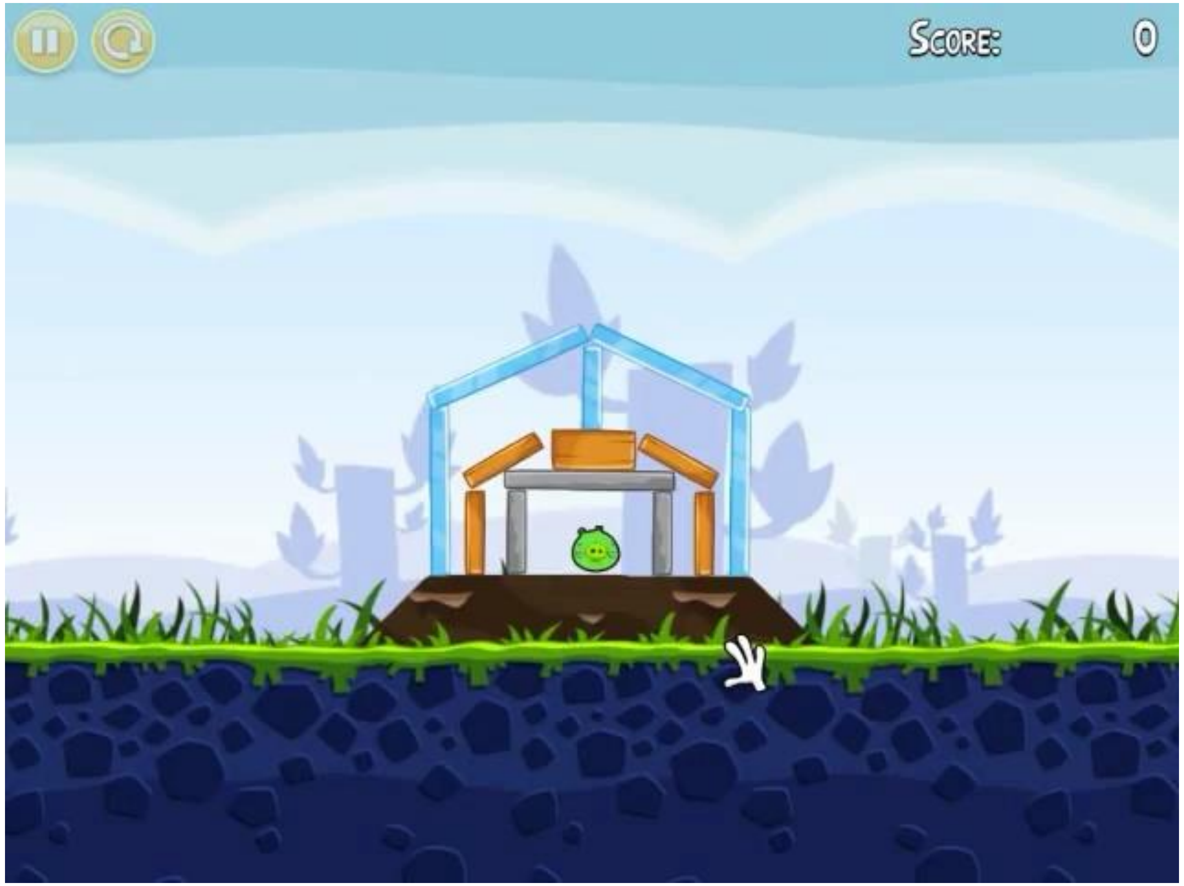
Rigid Bodies

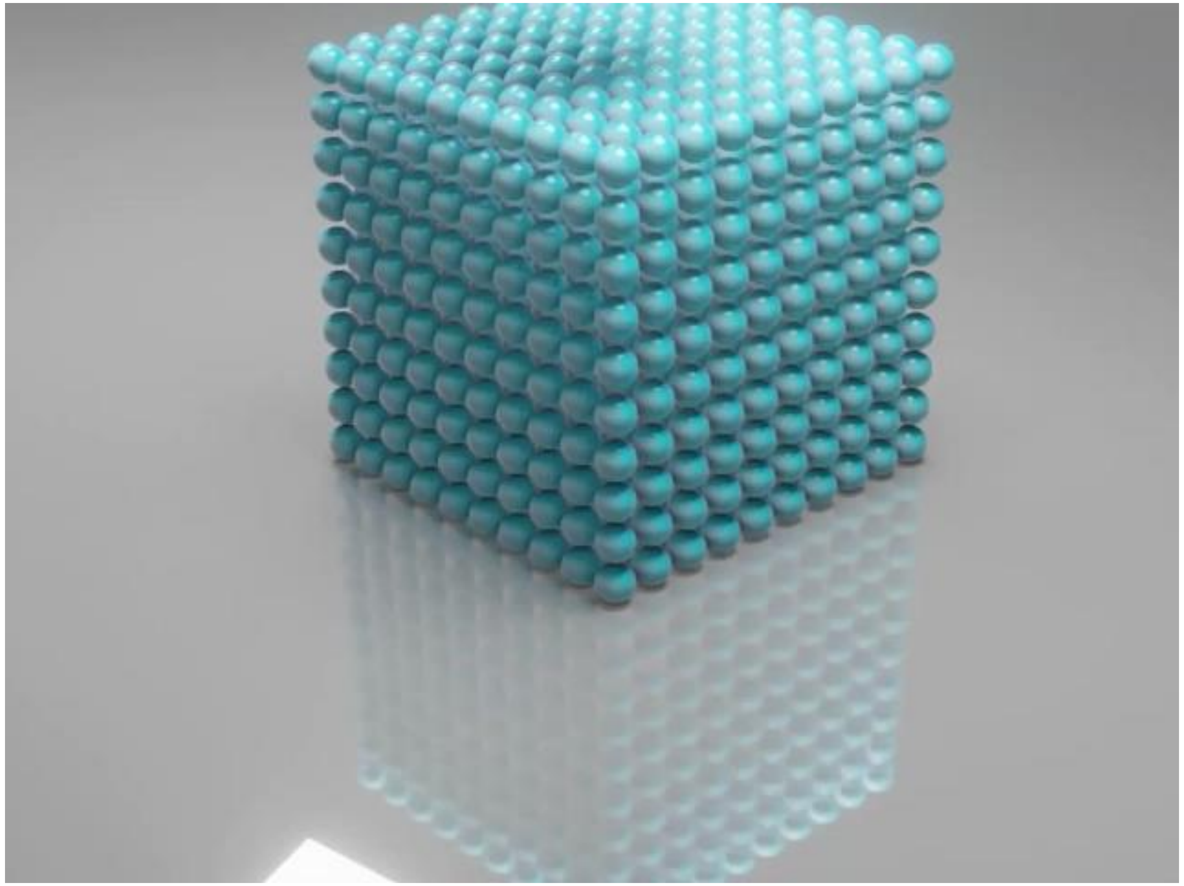
Our living environment is stuffed with rigid objects.

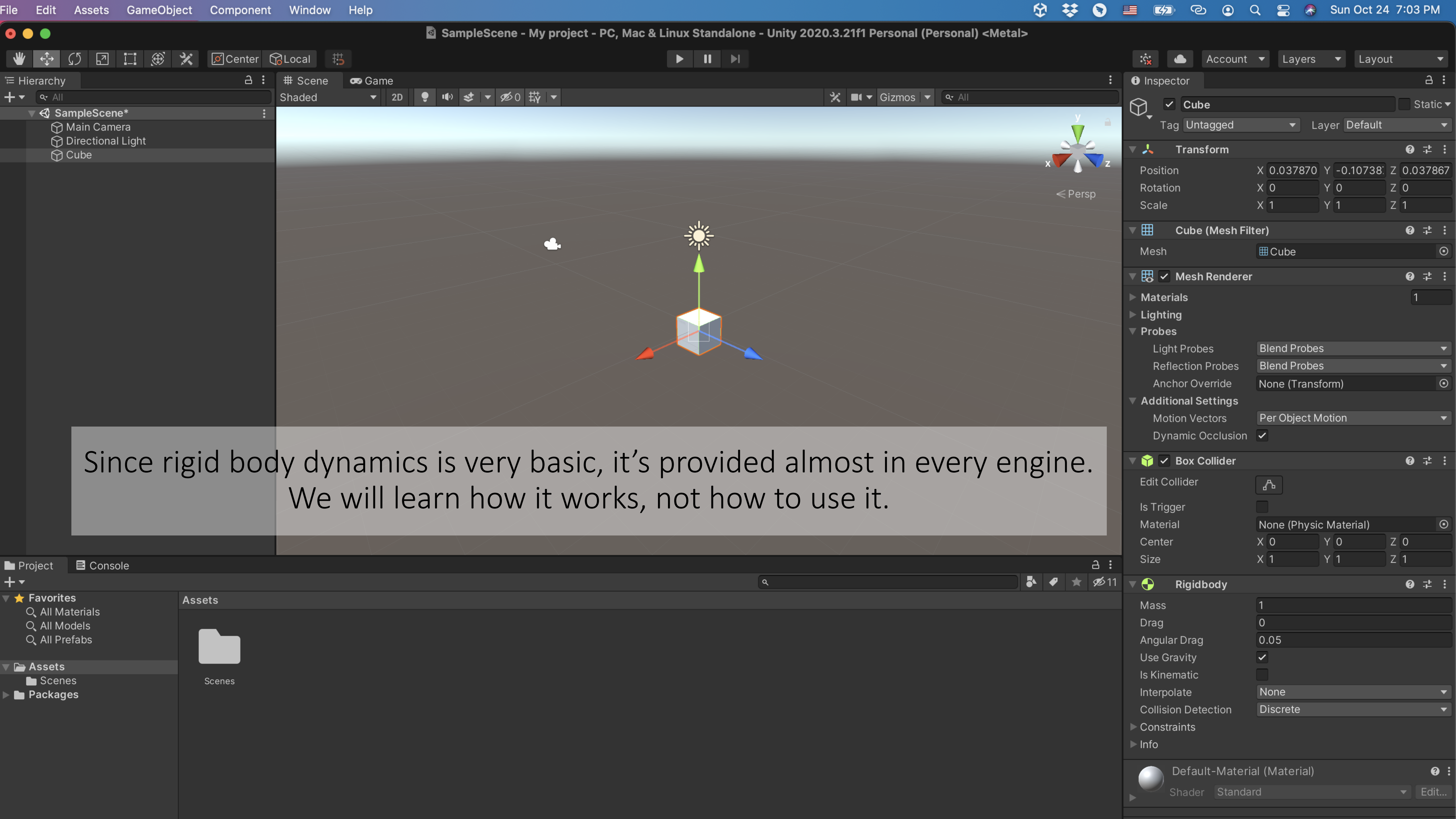


Rigid Bodies

In virtual worlds, we want to simulate rigid body motions as well.



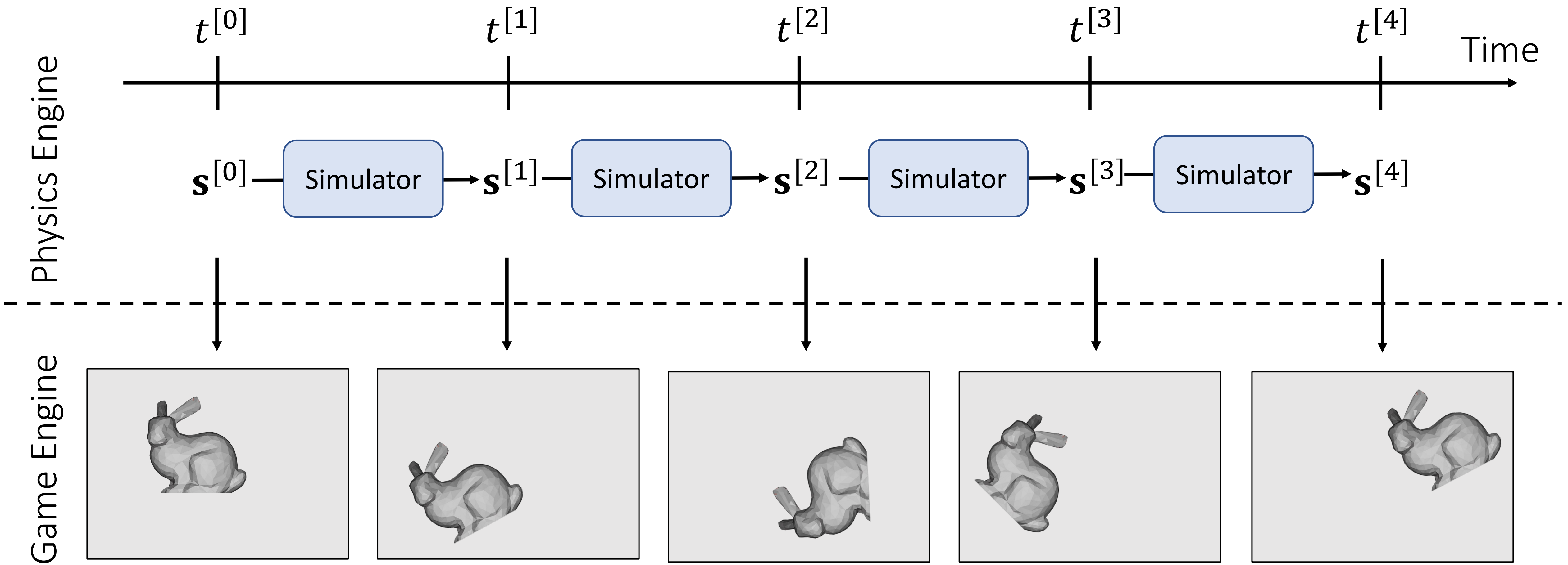




Since rigid body dynamics is very basic, it’s provided almost in every engine.
We will learn how it works, not how to use it.

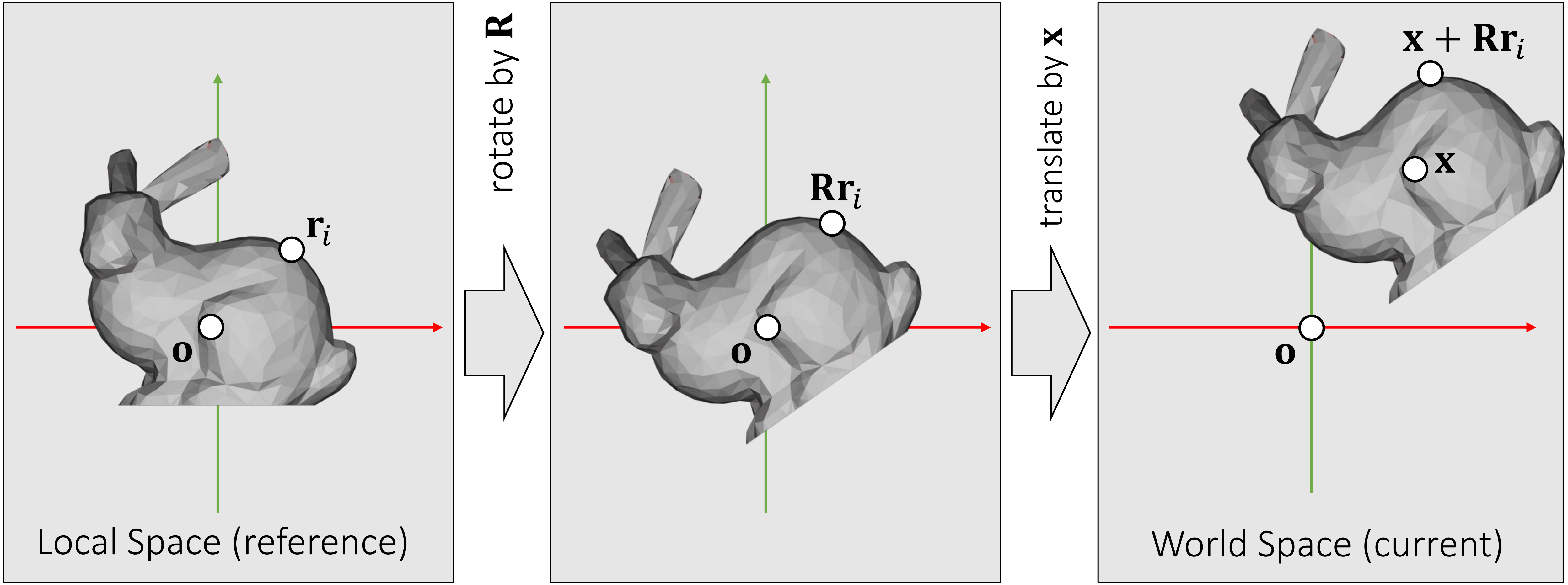
Rigid Body Simulation

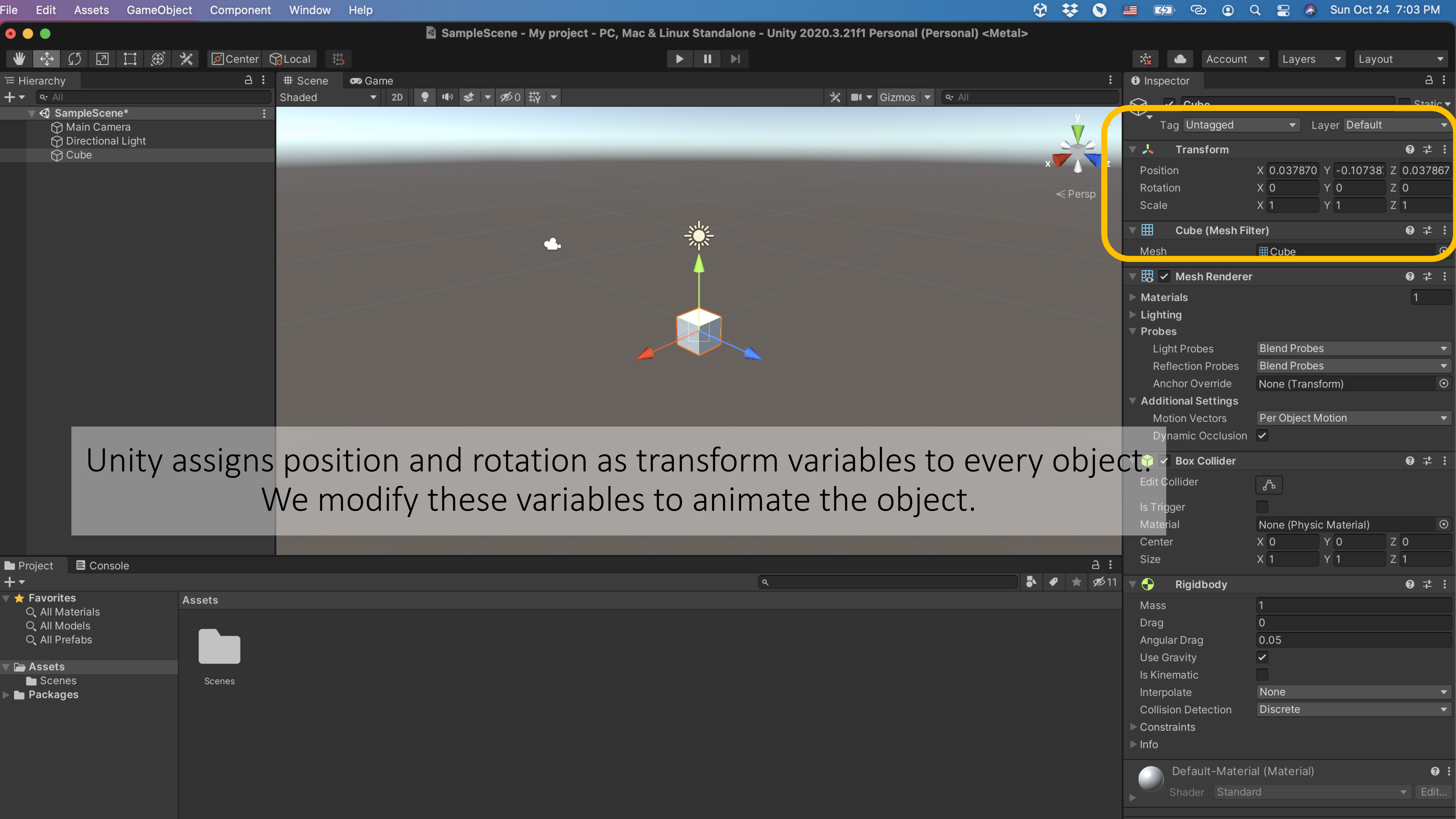
The goal of simulation is to update the state variable $\mathbf{s}^{[k]}$ over time.



Rigid Body Motion

If a rigid body cannot deform, its motion consists of two parts: translation and rotation.

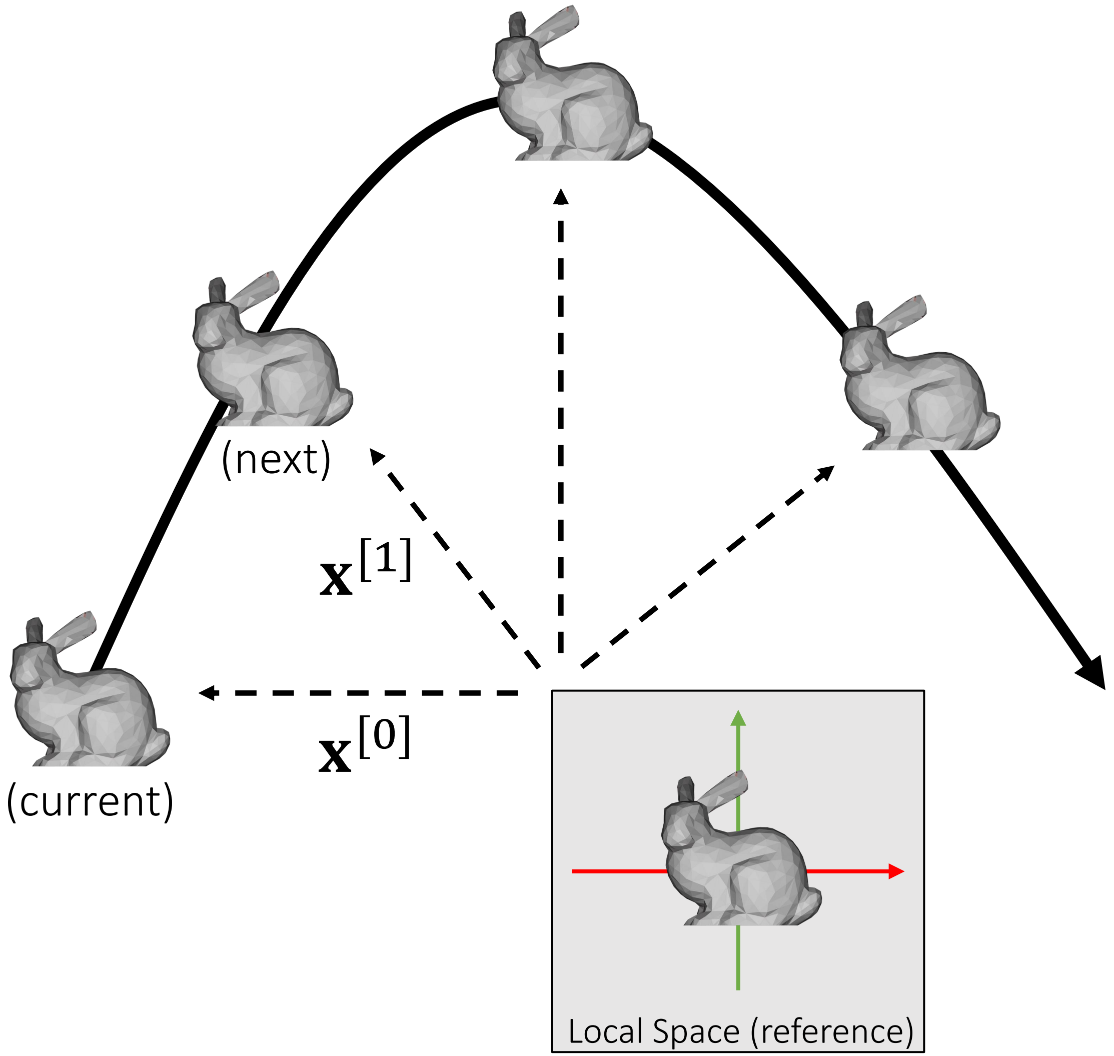




Unity assigns position and rotation as transform variables to every object.
We modify these variables to animate the object.

Translational Motion

Translational Motion



For translational motion, the state variable contains the position $\mathbf{x}$ and the velocity $\mathbf{v}$.

$$
\begin{cases}
\mathbf{v}\left(t^{[1]}\right) = \mathbf{v}\left(t^{[0]}\right) + M^{-1}\int_{t^{[0]}}^{t^{[1]}} \mathbf{f}(\mathbf{x}(t), \mathbf{v}(t), t)dt \\
\mathbf{x}\left(t^{[1]}\right) = \mathbf{x}\left(t^{[0]}\right) + \int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt
\end{cases}
$$

integration

Integration Methods Explained

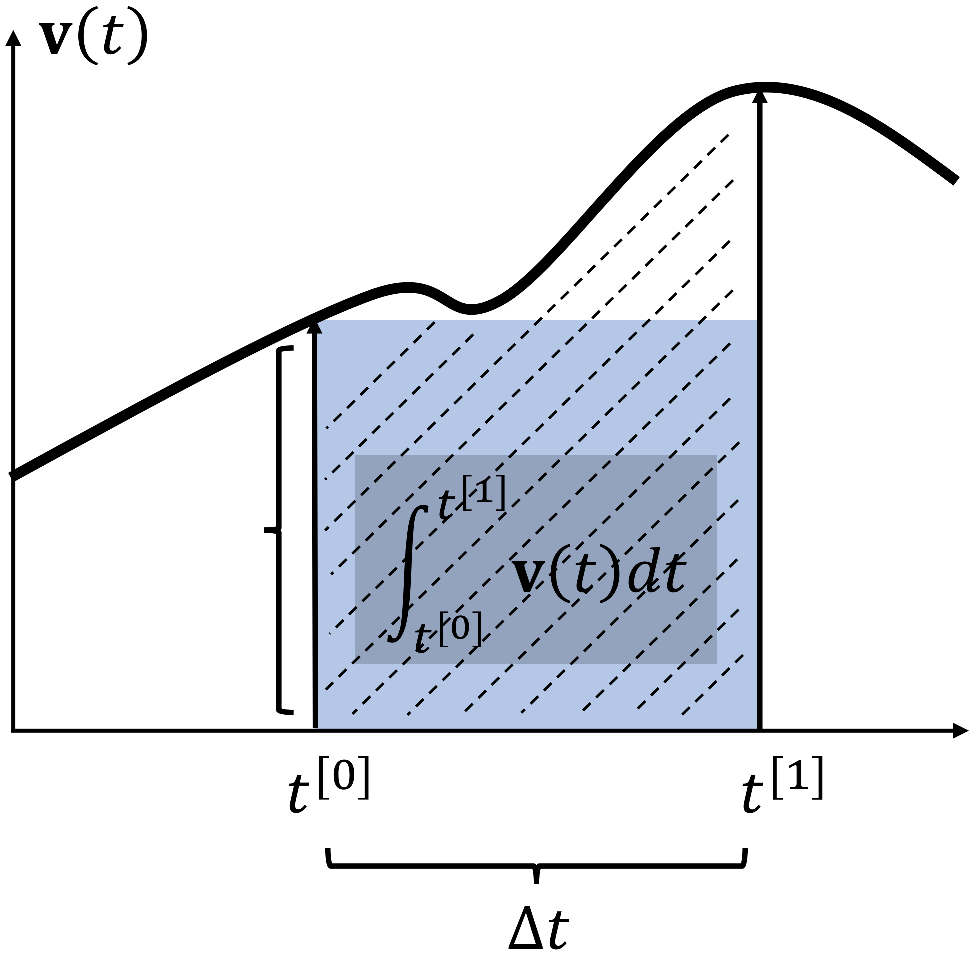
By definition, the integral $\mathbf{x}(t) = \int \mathbf{v}(t)dt$ is the area. Many methods estimate the area as a box.

Explicit Euler (1st-order accurate) sets the height at $t^{[0]}$.

$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \underbrace{\Delta t}_{\text{width}}\ \underbrace{\mathbf{v}\left(t^{[0]}\right)}_{\text{height}}$$

$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt = \Delta t\ \mathbf{v}\left(t^{[0]}\right) + \frac{\Delta t^2}{2}\mathbf{v}'\left(t^{[0]}\right) + \cdots$$

$$= \Delta t\ \mathbf{v}\left(t^{[0]}\right) + \underbrace{\boxed{O(\Delta t^2)}}_{\text{error}}$$



Integration Methods Explained

By definition, the integral $\mathbf{x}(t) = \int \mathbf{v}(t)dt$ is the area. Many methods estimate the area as a box.

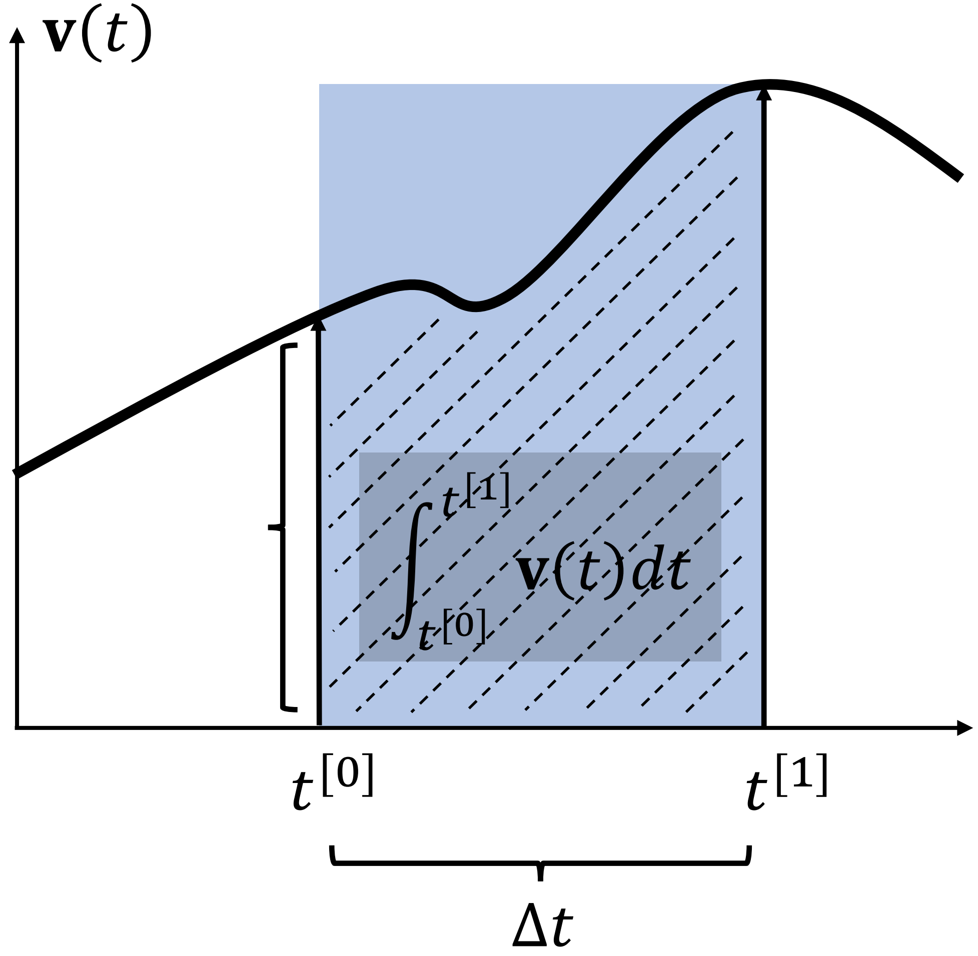
Implicit Euler (1st-order accurate) sets the height at $t^{[1]}$.

$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \underbrace{\Delta t}_{\text{width}} \underbrace{\mathbf{v}\left(t^{[1]}\right)}_{\text{height}}$$

$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt = \Delta t\, \mathbf{v}\left(t^{[1]}\right) - \frac{\Delta t^2}{2}\mathbf{v}'\left(t^{[1]}\right) + \cdots$$

$$= \Delta t\, \mathbf{v}\left(t^{[1]}\right) + \boxed{O(\Delta t^2)}$$

error



Integration Methods Explained

By definition, the integral $\mathbf{x}(t) = \int \mathbf{v}(t)dt$ is the area. Many methods estimate the area as a box.

Mid-point (2nd-order accurate) sets the height at $t^{[0.5]}$.

$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \underbrace{\Delta t}_{\text{width}} \underbrace{\mathbf{v}\left(t^{[0.5]}\right)}_{\text{height}}$$

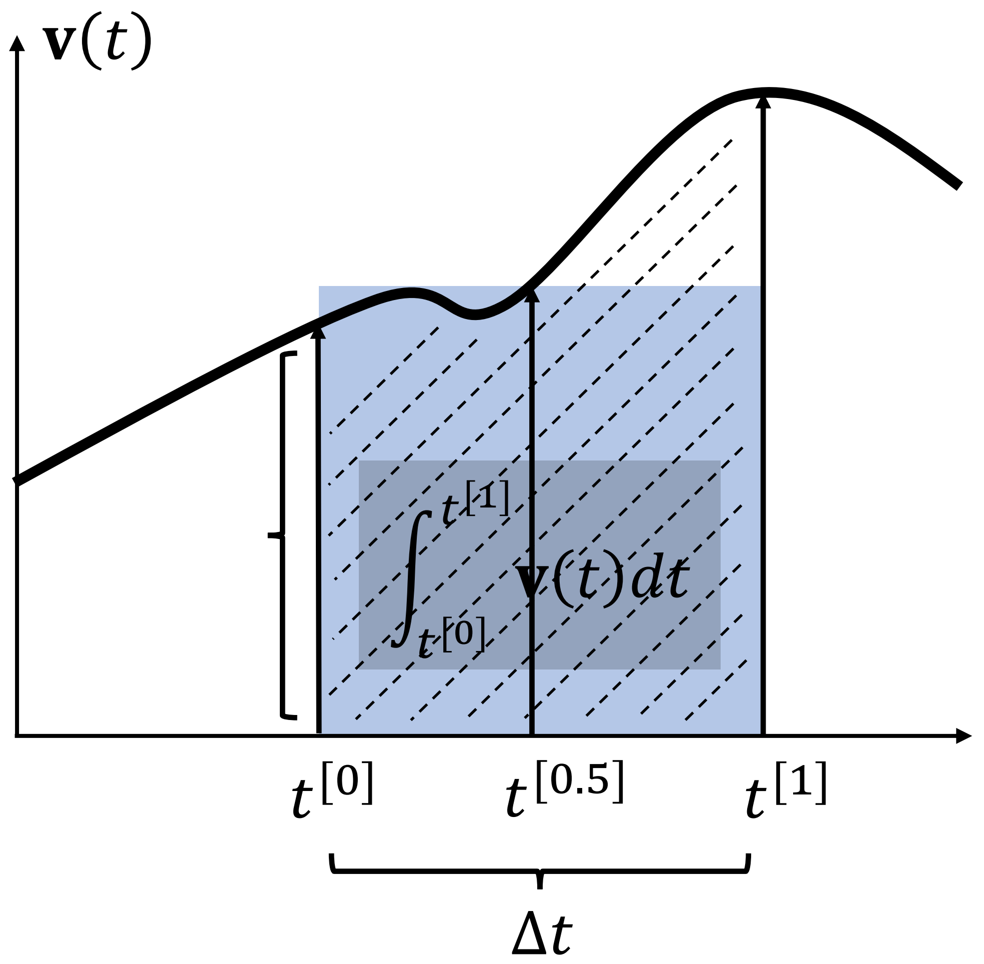
$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt = \int_{t^{[0]}}^{t^{[0.5]}} \mathbf{v}(t)dt + \int_{t^{[0.5]}}^{t^{[1]}} \mathbf{v}(t)dt$$

$$= \tfrac{1}{2}\Delta t\, \mathbf{v}\left(t^{[0.5]}\right) - \tfrac{\Delta t^2}{2}\mathbf{v}'\left(t^{[0.5]}\right) + O(\Delta t^3) +$$

$$\tfrac{1}{2}\Delta t\, \mathbf{v}\left(t^{[0.5]}\right) + \tfrac{\Delta t^2}{2}\mathbf{v}'\left(t^{[0.5]}\right) + O(\Delta t^3)$$

$$= \Delta t\, \mathbf{v}\left(t^{[0.5]}\right) + \boxed{O(\Delta t^3)}$$

error



Integration Methods Explained

By definition, the integral $\mathbf{x}(t) = \int \mathbf{v}(t)dt$ is the area. Many methods estimate the area as a box.

Explicit Euler (1st-order accurate) sets the height at $t^{[0]}$.

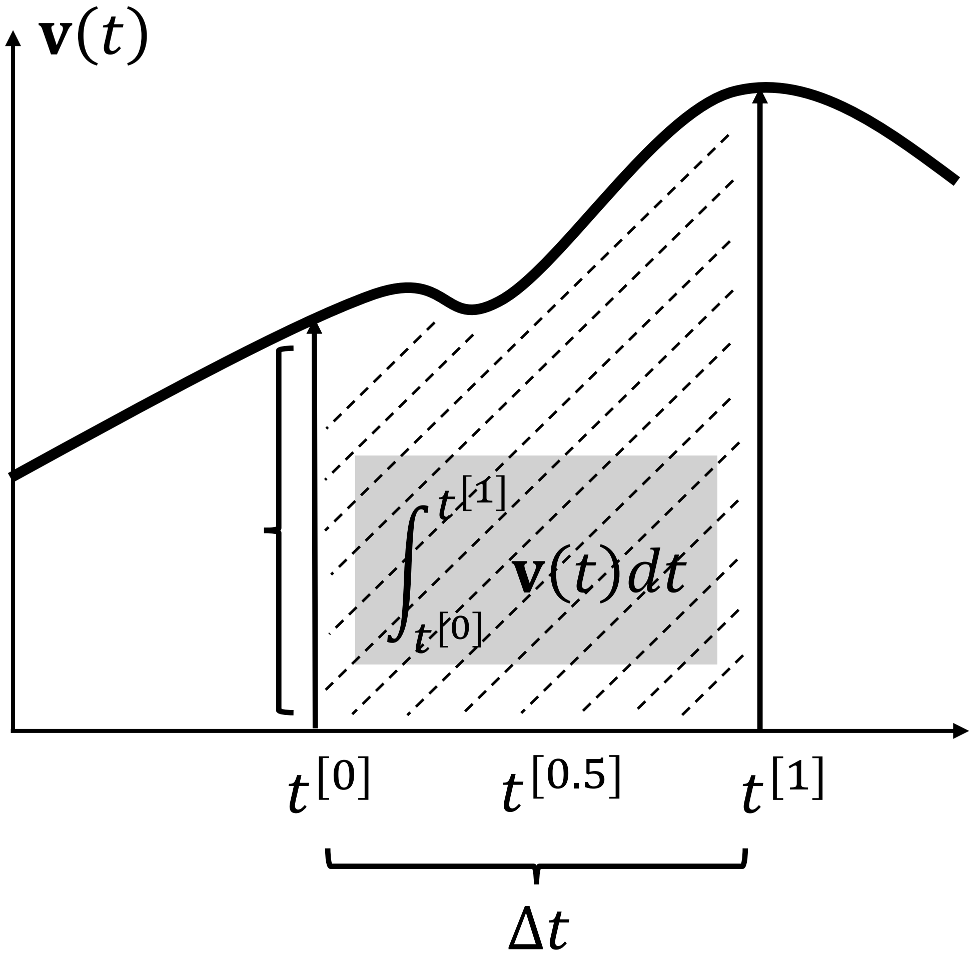
$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \Delta t\, \mathbf{v}\left(t^{[0]}\right)$$

Implicit Euler (1st-order accurate) sets the height at $t^{[1]}$.

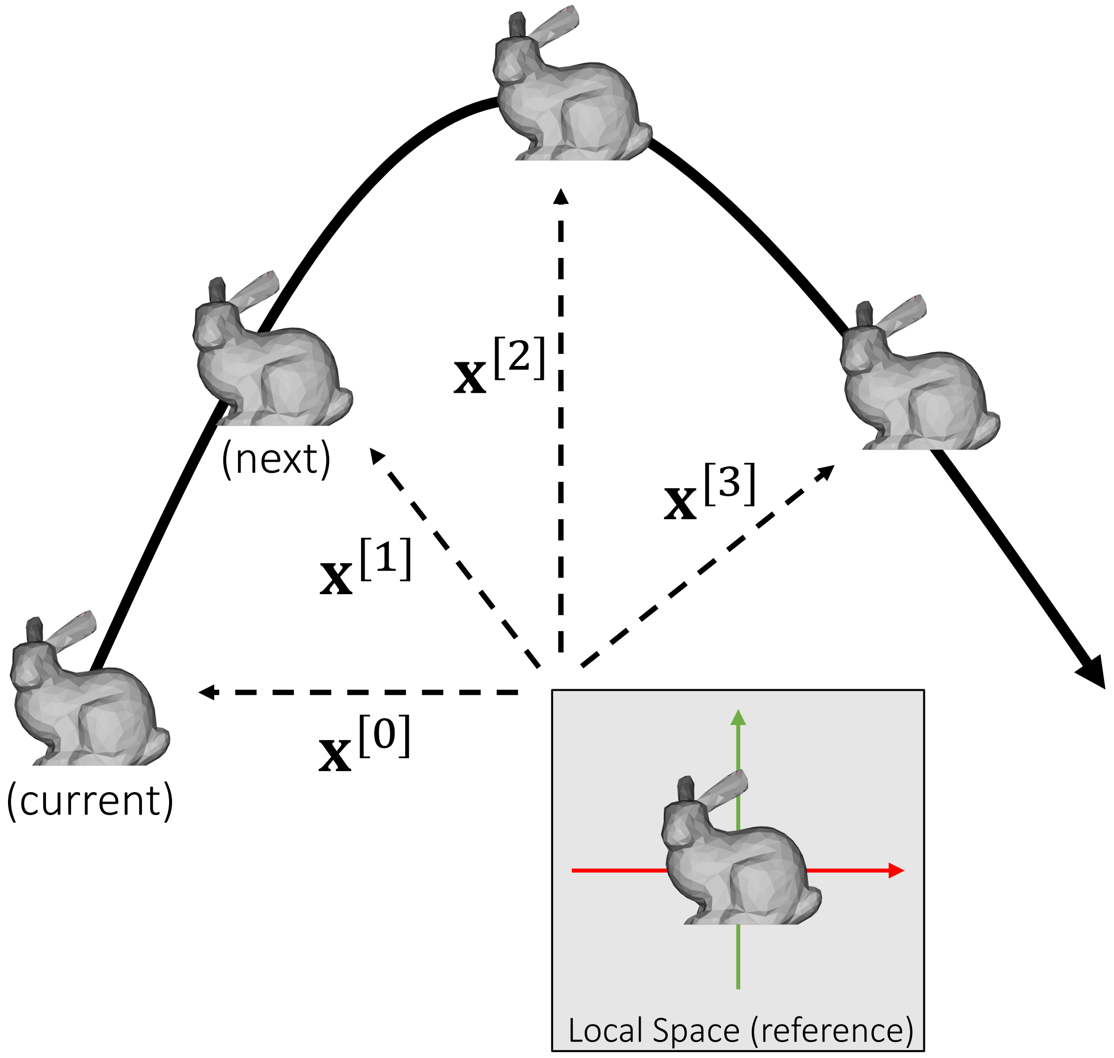
$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \Delta t\, \mathbf{v}\left(t^{[1]}\right)$$

Mid-point (2nd-order accurate) sets the height at $t^{[0.5]}$.

$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \Delta t\, \mathbf{v}(t^{[0.5]})$$



Translational Motion



For translational motion, the state variable contains the position $\mathbf{x}$ and the velocity $\mathbf{v}$.

$$\begin{cases} \mathbf{v}\left(t^{[1]}\right) = \mathbf{v}\left(t^{[0]}\right) + M^{-1}\int_{t^{[0]}}^{t^{[1]}} \mathbf{f}(\mathbf{x}(t), \mathbf{v}(t), t)dt \\ \mathbf{x}\left(t^{[1]}\right) = \mathbf{x}\left(t^{[0]}\right) + \int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \end{cases}$$

integration

$$\begin{cases} \mathbf{v}^{[1]} = \mathbf{v}^{[0]} + \Delta t M^{-1}\mathbf{f}^{[0]} \\ \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]} \end{cases}$$

← Explicit

← Implicit

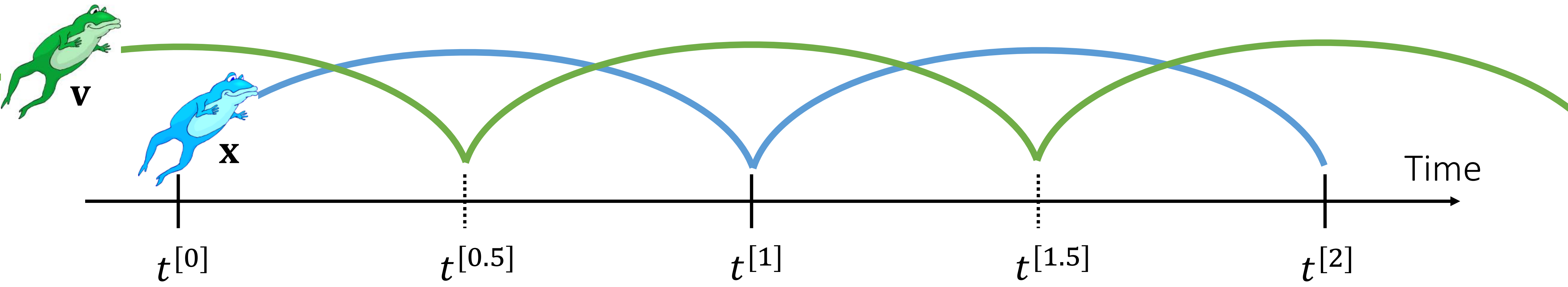
Leapfrog Integration

In some literature, such a approach is called *semi-implicit*.

$$\begin{cases} \mathbf{v}^{[1]} = \mathbf{v}^{[0]} + \Delta t M^{-1} \mathbf{f}^{[0]} & \longleftarrow \text{Explicit} \\ \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]} & \longleftarrow \text{Implicit} \end{cases}$$

It has a funnier name: the *leapfrog method*.

$$\begin{cases} \mathbf{v}^{[0.5]} = \mathbf{v}^{[-0.5]} + \Delta t M^{-1} \mathbf{f}^{[0]} & \longleftarrow \text{Mid-point} \\ \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[0.5]} & \longleftarrow \text{Mid-point} \end{cases}$$



Types of Forces

Gravity Force: $$\mathbf{f}^{[0]}_{\text{gravity}} = \underbrace{M}_{\text{mass}}\ \overbrace{\mathbf{g}}^{\text{gravity}}$$

~~Drag Force:~~ $$\cancel{\mathbf{f}^{[0]}_{\text{drag}} = -\underbrace{\sigma}_{\text{drag coefficient}}\ \overbrace{\mathbf{v}^{[0]}}^{\text{velocity}}}$$

But since the drag reduces the velocity, a more popular way is to decay the velocity.

$$\mathbf{v}^{[1]} = \underbrace{\alpha}_{\text{decay coefficient}}\ \mathbf{v}^{[0]}$$

Rigid Body Simulation (Translation Only)

$$\mathbf{s}^{[0]} \longrightarrow \text{Simulator} \longrightarrow \mathbf{s}^{[1]}$$

$$\begin{matrix} \mathbf{v}^{[0]} \\ \mathbf{x}^{[0]} \end{matrix}$$

$$\mathbf{f}_i^{[0]} \leftarrow \text{Force}\left(\mathbf{x}_i^{[0]}, \mathbf{v}_i^{[0]}\right)$$

$$\mathbf{f}^{[0]} \leftarrow \sum \mathbf{f}_i^{[0]}$$

$$\mathbf{v}^{[1]} \leftarrow \mathbf{v}^{[0]} + \Delta t M^{-1} \mathbf{f}^{[0]}$$

$$\mathbf{x}^{[1]} \leftarrow \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]}$$

$$\begin{matrix} \mathbf{v}^{[1]} \\ \mathbf{x}^{[1]} \end{matrix}$$

The mass M and the time step Δt are user-specified variables.

Rotational Motion

Rotation Represented by Matrix

- The matrix representation is widely used for rotational motion.
- It's friendly for applying rotation to each vertex (by matrix-vector multiplication).

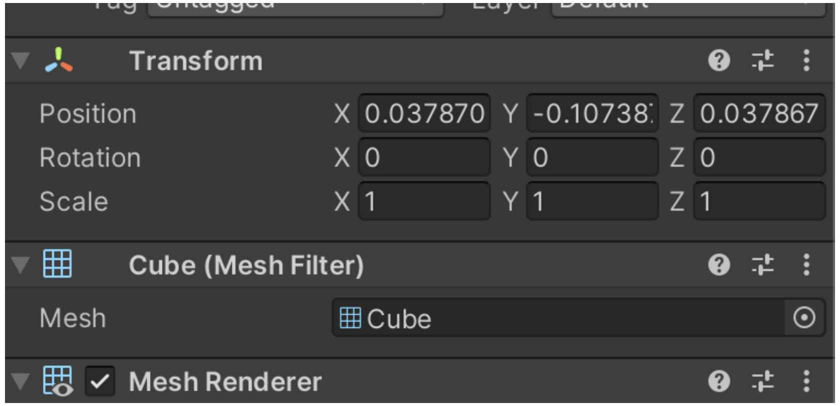
- But it is not suitable for dynamics:
  - It has too much redundancy: 9 elements but only 3 DoFs.
  - It is non-intuitive.
  - Defining its time derivative (*rotational velocity*) is also difficult.

$$\mathbf{R} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}$$

Rotation Represented by Euler Angles

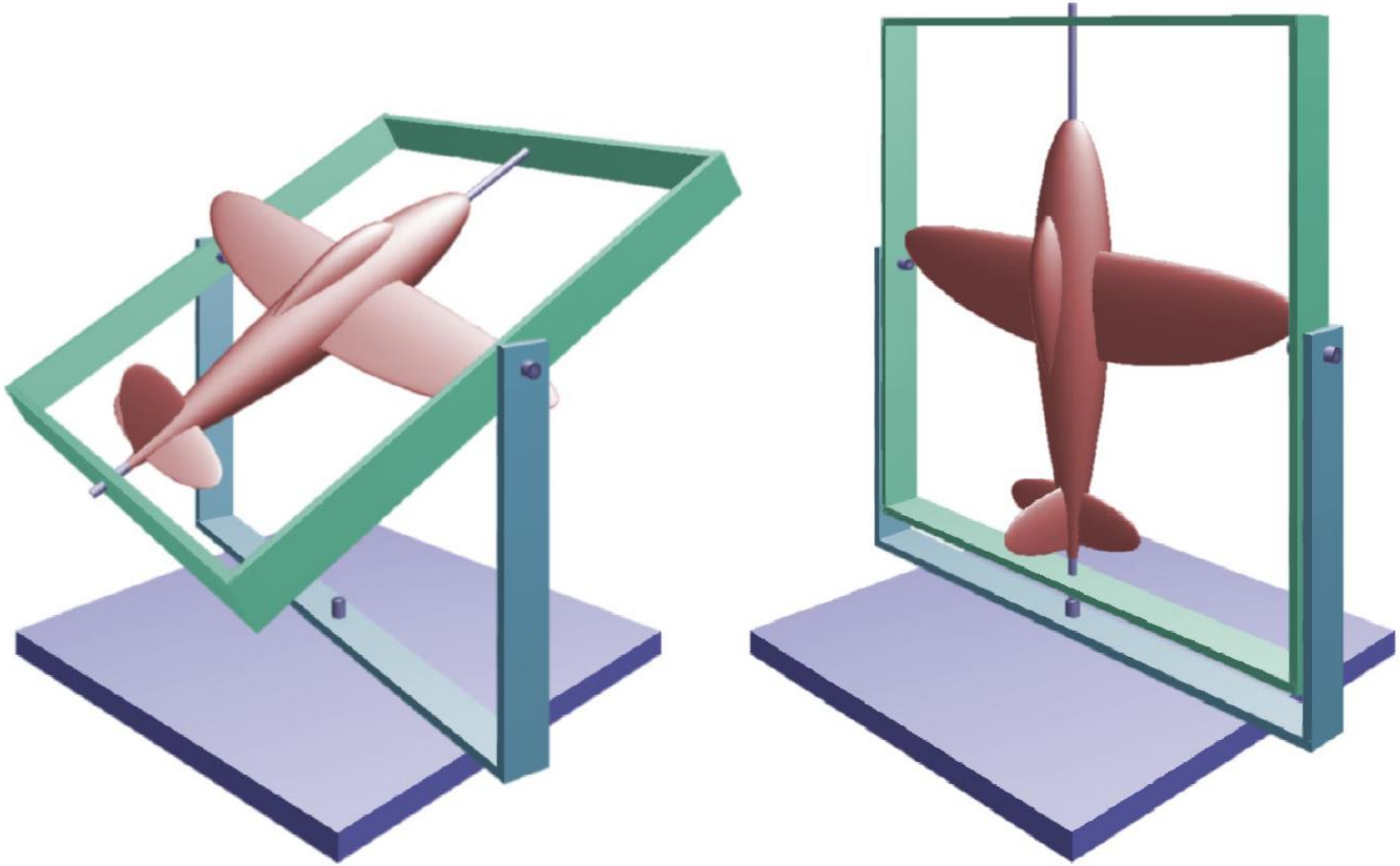
- The Euler Angles representation is also popular, often in design and control.
- It is intuitive. It uses three axial rotations to represent one general rotation. Each axial rotation uses an angle.
- In Unity, the order is rotation-by-Z, rotation-by-X, then rotation-by-Y.

- But it is not suitable for dynamics either:
  - It can lose DoFs in certain statuses: *gimbal lock*.
  - Defining its time derivative (*rotational velocity*) is difficult.



Gimbal Lock

The alignment of two or more axes results in a loss of rotational DoFs.



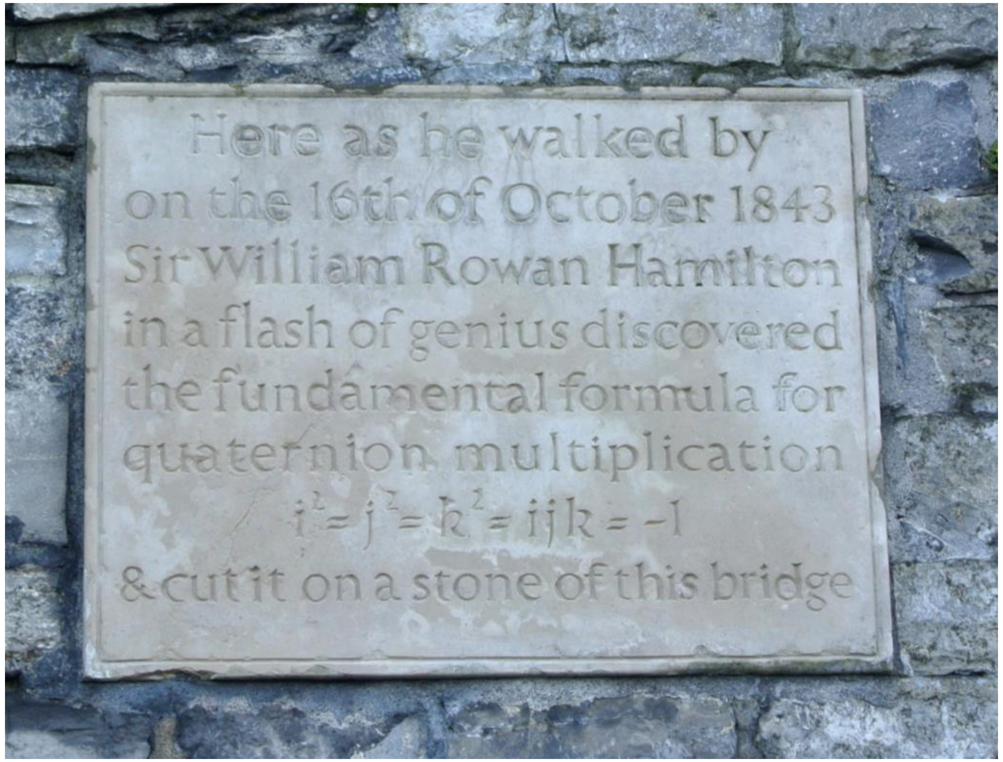
Rotation Represented by Quaternion

Complex multiplications

	1	**i**
1	1	**i**
i	**i**	−1

Quaternion multiplications

	1	**i**	**j**	**k**
1	1	**i**	**j**	**k**
i	**i**	−1	**k**	−**j**
j	**j**	−**k**	−1	**i**
k	**k**	**j**	−**i**	−1



In the complex system, two numbers represent a 2D point.

What about a "complex" system for 3D point? Quaternion!
Four numbers represent a 3D point (with multiplication and division).

Quaternion Arithematic

Let $\mathbf{q} = [s \quad \mathbf{v}]$ be a quaternion made of two parts: a scalar part s and a 3D vector part $\mathbf{v}$, accounting for $\mathbf{ijk}$.

$a\mathbf{q} = [as \quad a\mathbf{v}]$ — Scalar-quaternion Multiplication

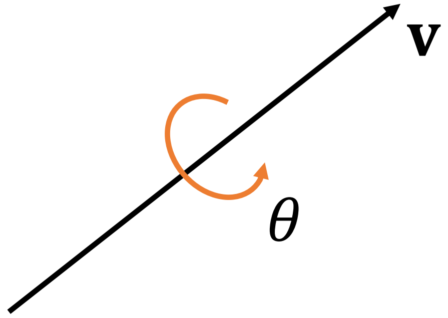
$\mathbf{q}_1 \pm \mathbf{q}_2 = [s_1 \pm s_2 \quad \mathbf{v}_1 \pm \mathbf{v}_2]$ — Addition/Subtraction

$\mathbf{q}_1 \times \mathbf{q}_2 = [s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 \quad s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$ — Multiplication

$\|\mathbf{q}\| = \sqrt{s^2 + \mathbf{v} \cdot \mathbf{v}}$ — Magnitude

Rotation Represented by Quaternion

- To represent a rotation around $\mathbf{v}$ by angle θ, we set the quaternion as:



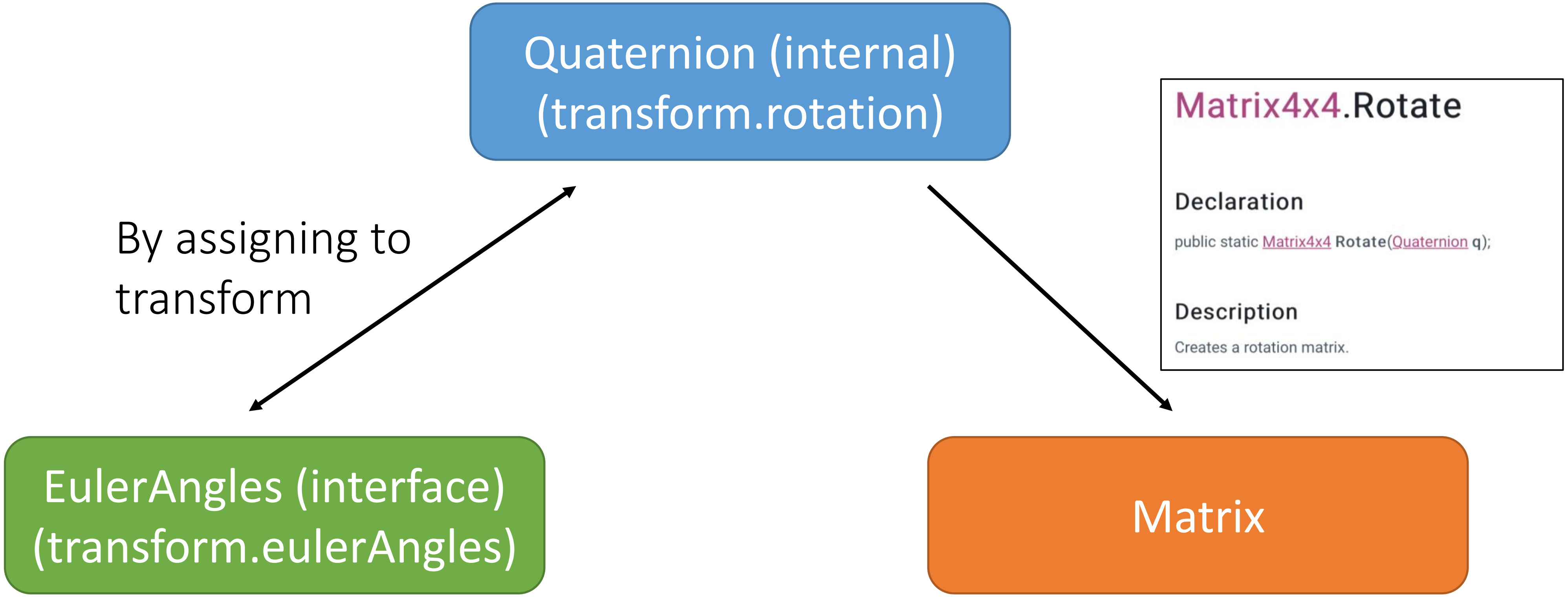
$$\begin{cases} \mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} & \mathbf{v} \end{bmatrix} \\ \|\mathbf{q}\| = 1 \end{cases} \Rightarrow \begin{cases} \mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} & \mathbf{v} \end{bmatrix} \\ \|\mathbf{v}\|^2 = \sin^2\frac{\theta}{2} \end{cases}$$

- It's very intuitive. It's the built-in representation in Unity.

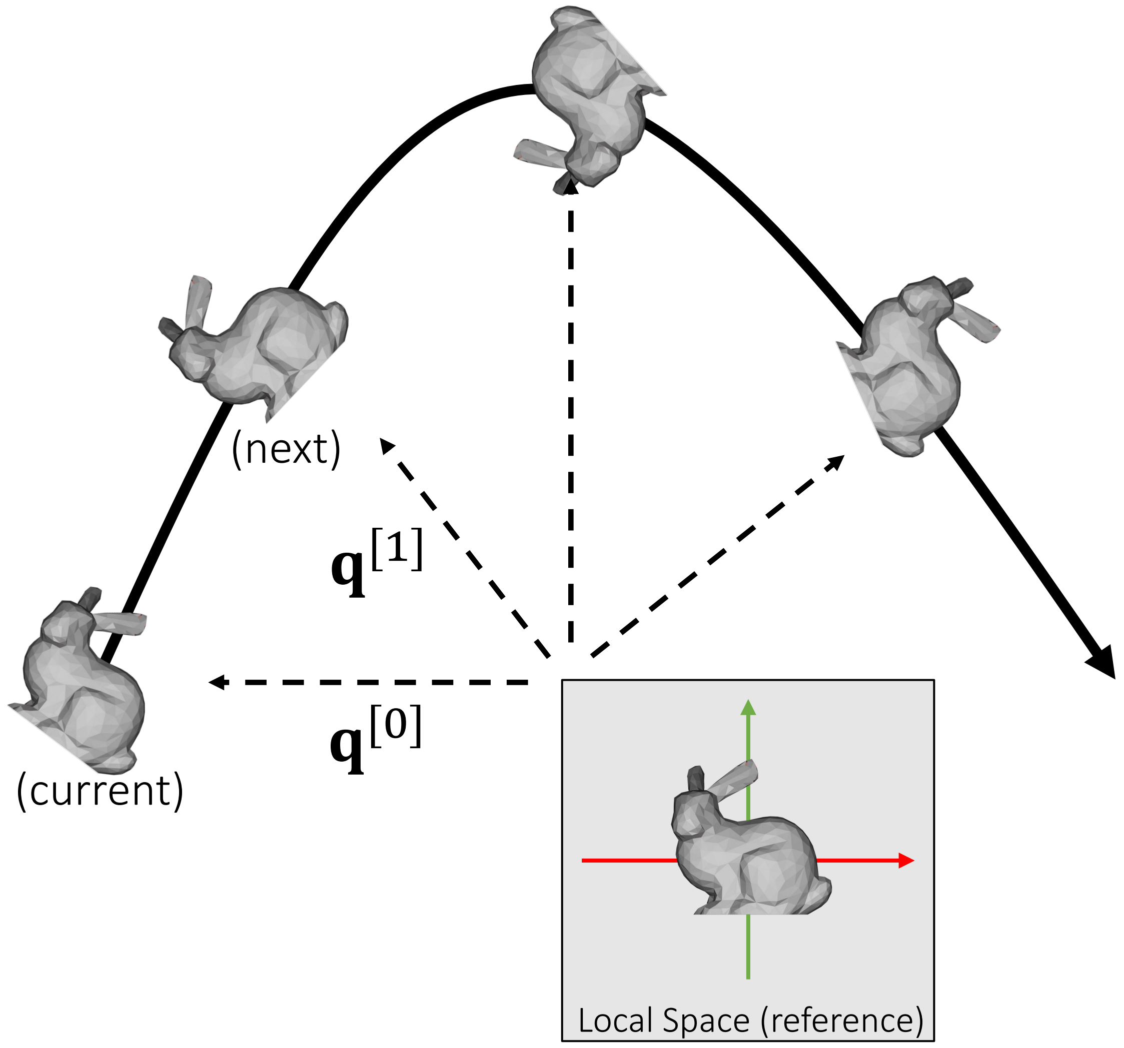
- Convertible to the matrix:

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

Rotation Representations in Unity



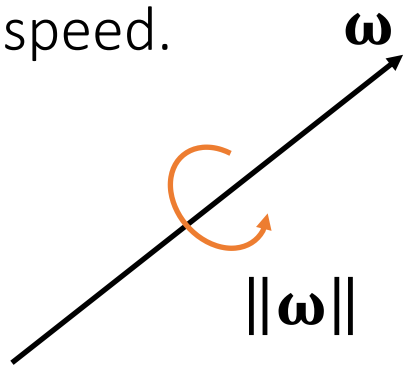
Rotational Motion



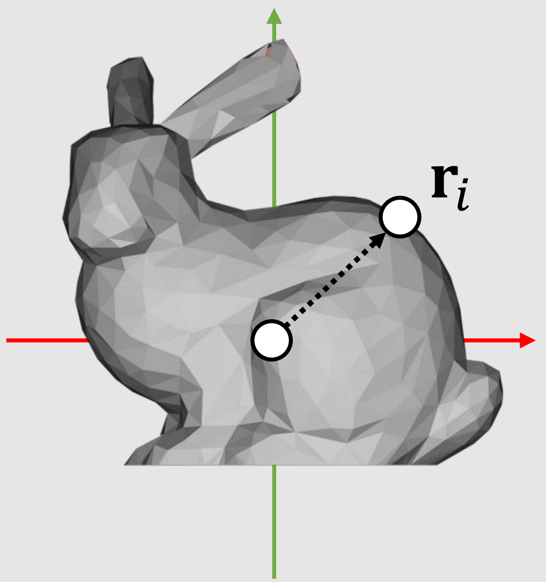
Now we choose quaternion $\mathbf{q}$ to represent the orientation, i.e., the rotation from the *reference* to the *current*.

We use a 3D vector $\boldsymbol{\omega}$ to denote angular velocity.

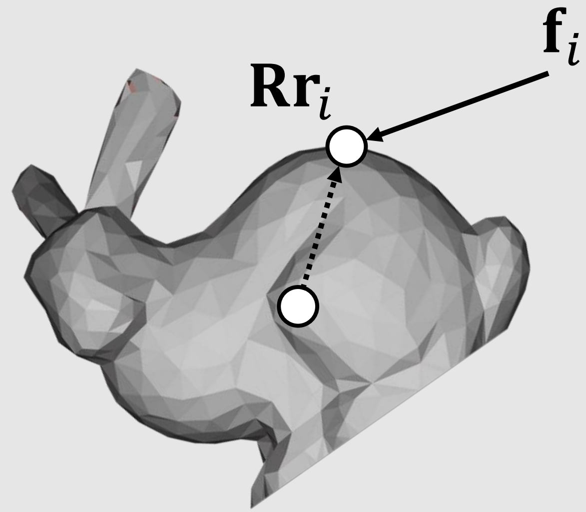
- The direction of $\boldsymbol{\omega}$ is the axis.
- The magnitude of $\boldsymbol{\omega}$ is the speed.



Torque and Inertia



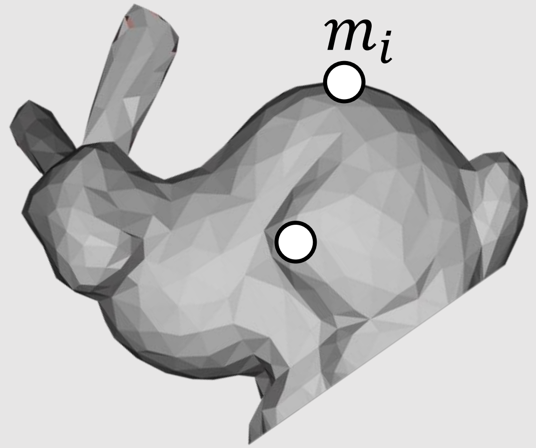
Local Space (reference)



$$\boldsymbol{\tau}_i = (\mathbf{R}\mathbf{r}_i) \times \mathbf{f}_i$$

$$\boldsymbol{\tau} = \sum \boldsymbol{\tau}_i$$

The rotational equivalent of force is called torque $\boldsymbol{\tau}$.



$$\mathbf{I}_{\mathbf{ref}} = \sum m_i(\mathbf{r}_i^{\mathrm{T}}\mathbf{r}_i\mathbf{1} - \mathbf{r}_i\boldsymbol{r}_i^{\mathrm{T}})$$

$$\mathbf{I} = \mathbf{R}\mathbf{I}_{\mathbf{ref}}\mathbf{R}^{\mathrm{T}}$$

The rotational equivalent of mass is called inertia $\mathbf{I}$.

Translational and Rotational Motion

Translational (linear)

$$\begin{cases} \mathbf{v}^{[1]} = \mathbf{v}^{[0]} + \Delta t M^{-1} \mathbf{f}^{[0]} \\ \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]} \end{cases}$$

Rotational (Angular)

$$\begin{cases} \boldsymbol{\omega}^{[1]} = \boldsymbol{\omega}^{[0]} + \Delta t (\mathbf{I}^{[0]})^{-1} \boldsymbol{\tau}^{[0]} \\ \mathbf{q}^{[1]} = \mathbf{q}^{[0]} + \begin{bmatrix} 0 & \frac{\Delta t}{2} \boldsymbol{\omega}^{[1]} \end{bmatrix} \times \mathbf{q}^{[0]} \end{cases}$$

States

Velocity $\mathbf{v}$

Position $\mathbf{x}$
(transform.position in Unity)

Angular velocity $\boldsymbol{\omega}$

Quaternion $\mathbf{q}$
(transform.rotation in Unity)

Physical Quantities

Mass M

Force $\mathbf{f}$

Inertia $\mathbf{I}$

Torque $\boldsymbol{\tau}$

Rigid Body Simulation

$\mathbf{s}^{[0]}$ → Simulator → $\mathbf{s}^{[1]}$

$\mathbf{v}^{[0]}$

$\mathbf{x}^{[0]}$

$$\mathbf{f}_i^{[0]} \leftarrow \mathrm{Force}\left(\mathbf{x}_i^{[0]}, \mathbf{v}_i^{[0]}\right)$$

$$\mathbf{f}^{[0]} \leftarrow \sum \mathbf{f}_i^{[0]}$$

$$\mathbf{v}^{[1]} \leftarrow \mathbf{v}^{[0]} + \Delta t M^{-1} \mathbf{f}^{[0]}$$

$$\mathbf{x}^{[1]} \leftarrow \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]}$$

$\mathbf{v}^{[1]}$

$\mathbf{x}^{[1]}$

$\boldsymbol{\omega}^{[0]}$

$\mathbf{q}^{[0]}$

$$\mathbf{R}^{[0]} \leftarrow \mathrm{Matrix.Rotate}(\mathbf{q}^{[0]})$$

$$\boldsymbol{\tau}_i^{[0]} \leftarrow (\mathbf{R}^{[0]} \mathbf{r}_i) \times \mathbf{f}_i^{[0]}$$

$$\boldsymbol{\tau}^{[0]} \leftarrow \sum \boldsymbol{\tau}_i^{[0]}$$

$$\mathbf{I}^{[0]} \leftarrow \mathbf{R}^{[0]} \mathbf{I}_{\mathbf{ref}} (\mathbf{R}^{[0]})^{\mathrm{T}}$$

$$\boldsymbol{\omega}^{[1]} \leftarrow \boldsymbol{\omega}^{[0]} + \Delta t (\mathbf{I}^{[0]})^{-1} \boldsymbol{\tau}^{[0]}$$

$$\mathbf{q}^{[1]} \leftarrow \mathbf{q}^{[0]} + \left[0 \quad \frac{\Delta t}{2} \boldsymbol{\omega}^{[1]}\right] \times \mathbf{q}^{[0]}$$

$\boldsymbol{\omega}^{[1]}$

$\mathbf{q}^{[1]}$

Implementation

In practice, we update the same state variable $\mathbf{s} = \{\mathbf{v}, \mathbf{x}, \boldsymbol{\omega}, \mathbf{q}\}$ over time.

$\mathbf{v}$

$\mathbf{x}$

$$\mathbf{f}_i \leftarrow \mathrm{Force}(\mathbf{x}_i, \mathbf{v}_i)$$

$$\mathbf{f} \leftarrow \sum \mathbf{f}_i$$

$$\mathbf{v} \leftarrow \mathbf{v} + \Delta t M^{-1}\mathbf{f}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta t \mathbf{v}$$

$\mathbf{v}$

$\mathbf{x}$

$\boldsymbol{\omega}$

$\mathbf{q}$

$$\mathbf{R} \leftarrow \mathrm{Matrix.Rotate}(\mathbf{q})$$

$$\boldsymbol{\tau}_i \leftarrow (\mathbf{R}\mathbf{r}_i) \times \mathbf{f}_i$$

$$\boldsymbol{\tau} \leftarrow \sum \boldsymbol{\tau}_i$$

$$\mathbf{I} \leftarrow \mathbf{R}\mathbf{I}_{\mathbf{ref}}\mathbf{R}^{\mathrm{T}}$$

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \Delta t(\mathbf{I})^{-1}\boldsymbol{\tau}$$

$$\mathbf{q} \leftarrow \mathbf{q} + \left[0 \quad \frac{\Delta t}{2}\boldsymbol{\omega}\right] \times \mathbf{q}$$

$\boldsymbol{\omega}$

$\mathbf{q}$

Some More Implementation Issues

- Translational motion is much easier to implement than rotational motion.

- You can implement the update of $\mathbf{q}$ first using a constant $\boldsymbol{\omega}$. In that case, the object should spin constantly.

- Gravity doesn't cause any torque! If your simulator does not contain any other force, there is no need to update $\boldsymbol{\omega}$.

- Do the lab assignment to learn more details.

After-Class Reading (Before Collision)

https://graphics.pixar.com/pbm2001

Physically Based Modeling

ONLINE SIGGRAPH 2001 COURSE NOTES

All documents on this page are in Adobe Acrobat format. If you need to obtain an Acrobat reader, please visit the Adobe Acrobat Reader page.

- **Introduction**
- **Differential Equation Basics**
  - Lecture Notes
  - Slides
- **Particle Dynamics**
  - Lecture Notes
  - Slides
- **Implicit Methods**
  - Lecture Notes
  - Slides
- **Cloth and Fur Energy Functions**
  - Slides
- **Rigid Body Dynamics**
  - Lecture Notes
  - Slides
- **Constrained Dynamics**
  - Lecture Notes
  - Slides
- **Collision and Contact**
  - Slides