

GAMES103: Intro to Physics-Based Animation

Rigid Body Dynamics

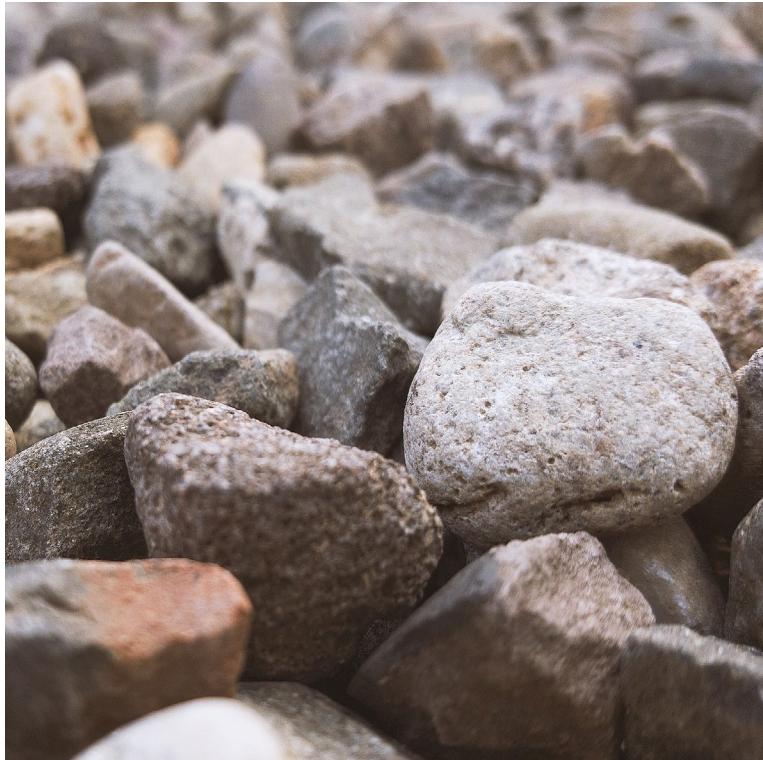
Huamin Wang

Nov 2021

What is rigid body dynamics?

Rigid Bodies

Our living environment is stuffed with rigid objects.

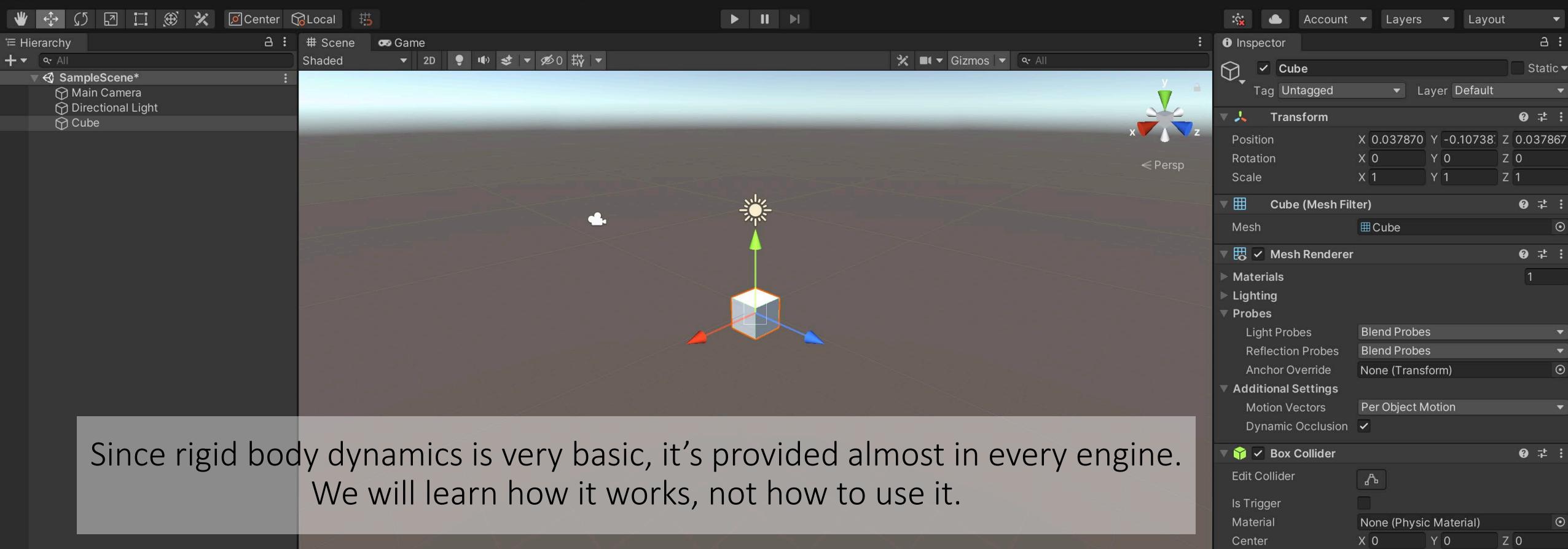


Rigid Bodies

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



SampleScene - My project - PC, Mac & Linux Standalone - Unity 2020.3.21f1 Personal (Personal) <Metal>



Project Console

Favorites

- All Materials
- All Models
- All Prefabs

Assets

Scenes

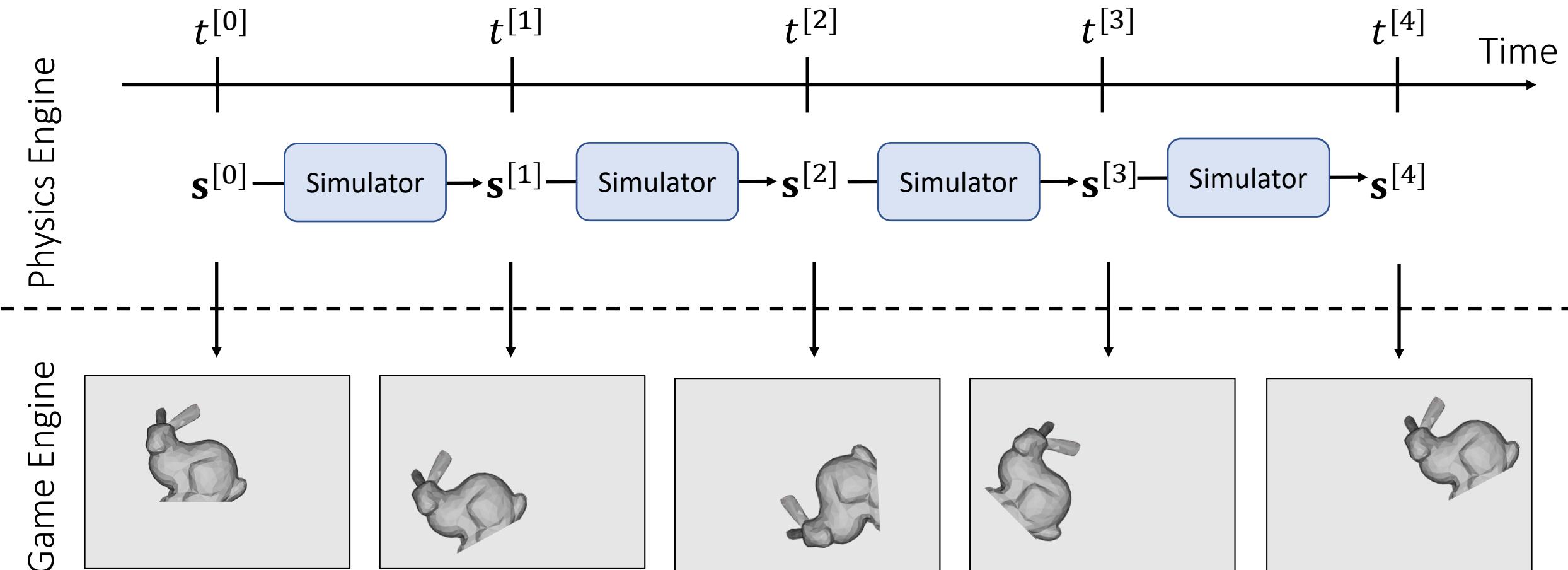
Default-Material (Material)

Shader Standard

Edit...

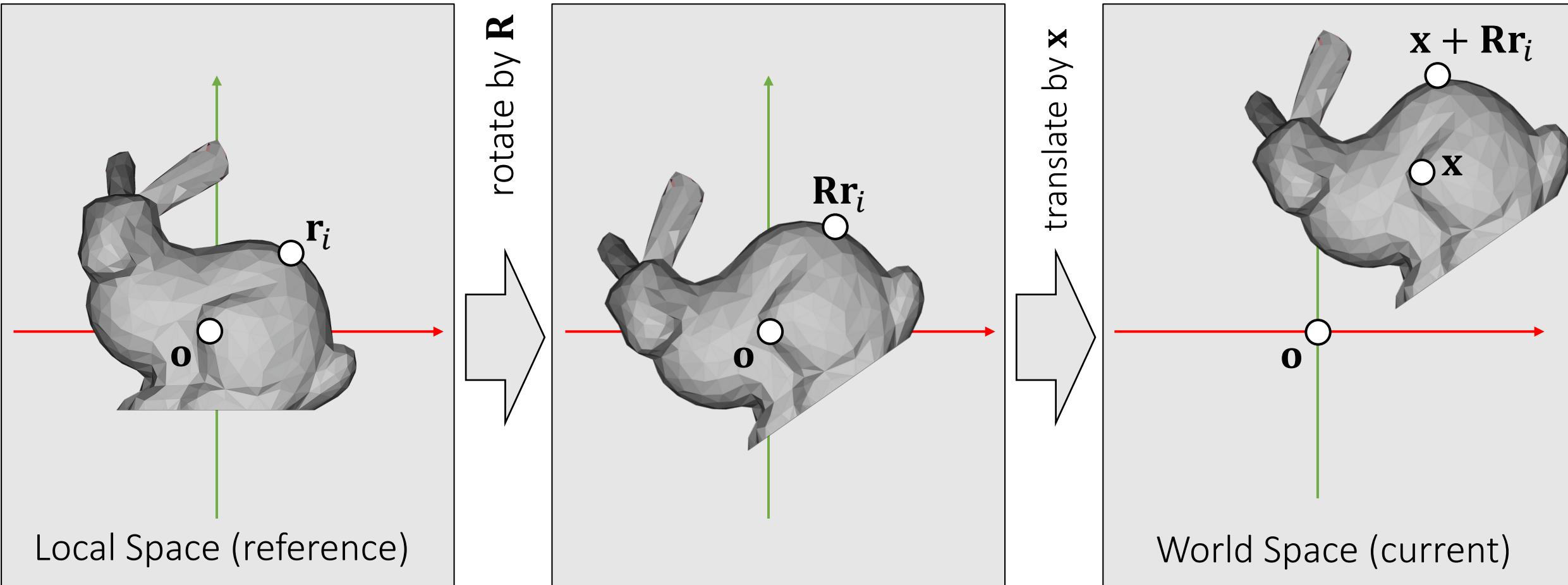
Rigid Body Simulation

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.



SampleScene - My project - PC, Mac & Linux Standalone - Unity 2020.3.21f1 Personal (Personal) <Metal>

The Unity Editor interface is shown, featuring the Hierarchy, Scene, and Inspector panels. In the Scene view, a cube is positioned at approximately (0.037870, -0.10738, 0.037867). The Inspector panel displays the Transform component for the cube, which includes Position (X: 0.037870, Y: -0.10738, Z: 0.037867), Rotation (X: 0, Y: 0, Z: 0), and Scale (X: 1, Y: 1, Z: 1). A yellow box highlights the Transform section of the Inspector panel.

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

Hierarchy:

- SampleScene*
- Main Camera
- Directional Light
- Cube

Scene View:

Inspector (Highlighted):

Transform	X	Y	Z
Position	0.037870	-0.10738	0.037867
Rotation	0	0	0
Scale	1	1	1

Mesh Filter:

- Mesh: Cube

Mesh Renderer:

- Materials: 1

Probes:

- Light Probes: Blend Probes
- Reflection Probes: Blend Probes
- Anchor Override: None (Transform)

Additional Settings:

- Motion Vectors: Per Object Motion
- Dynamic Occlusion: checked

Box Collider:

- Edit Collider:
- Is Trigger:
- Material: None (Physic Material)
- Center: X: 0, Y: 0, Z: 0
- Size: X: 1, Y: 1, Z: 1

Rigidbody:

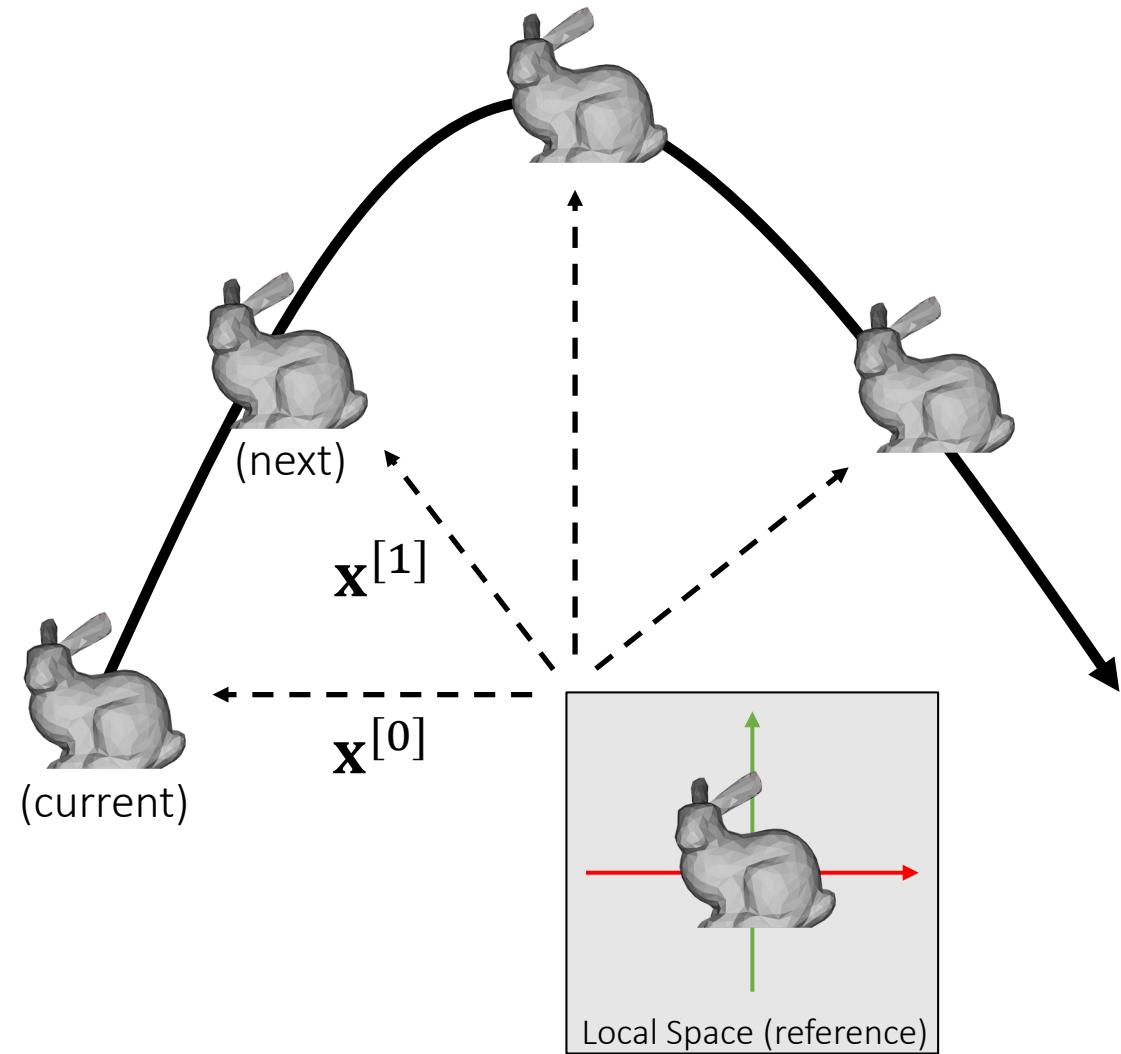
- Mass: 1
- Drag: 0
- Angular Drag: 0.05
- Use Gravity:
- Is Kinematic:
- Interpolate: None
- Collision Detection: Discrete
- Constraints:
- Info:

Assets:

- Scenes

Translational Motion

Translational Motion



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

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

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.

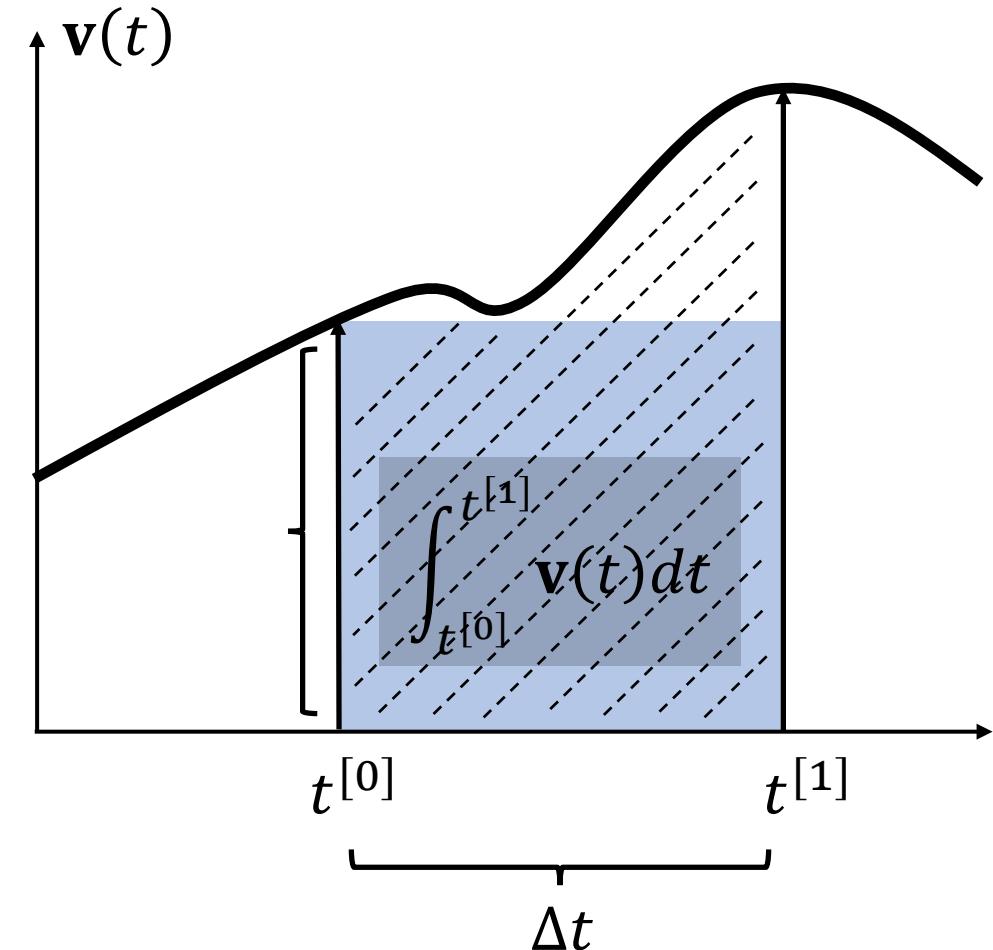
Explicit Euler (1st-order accurate) sets the height at $t^{[0]}$.

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

width height

$$\begin{aligned} \int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt &= \Delta t \mathbf{v}(t^{[0]}) + \frac{\Delta t^2}{2} \mathbf{v}'(t^{[0]}) + \dots \\ &= \Delta t \mathbf{v}(t^{[0]}) + O(\Delta t^2) \end{aligned}$$

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.

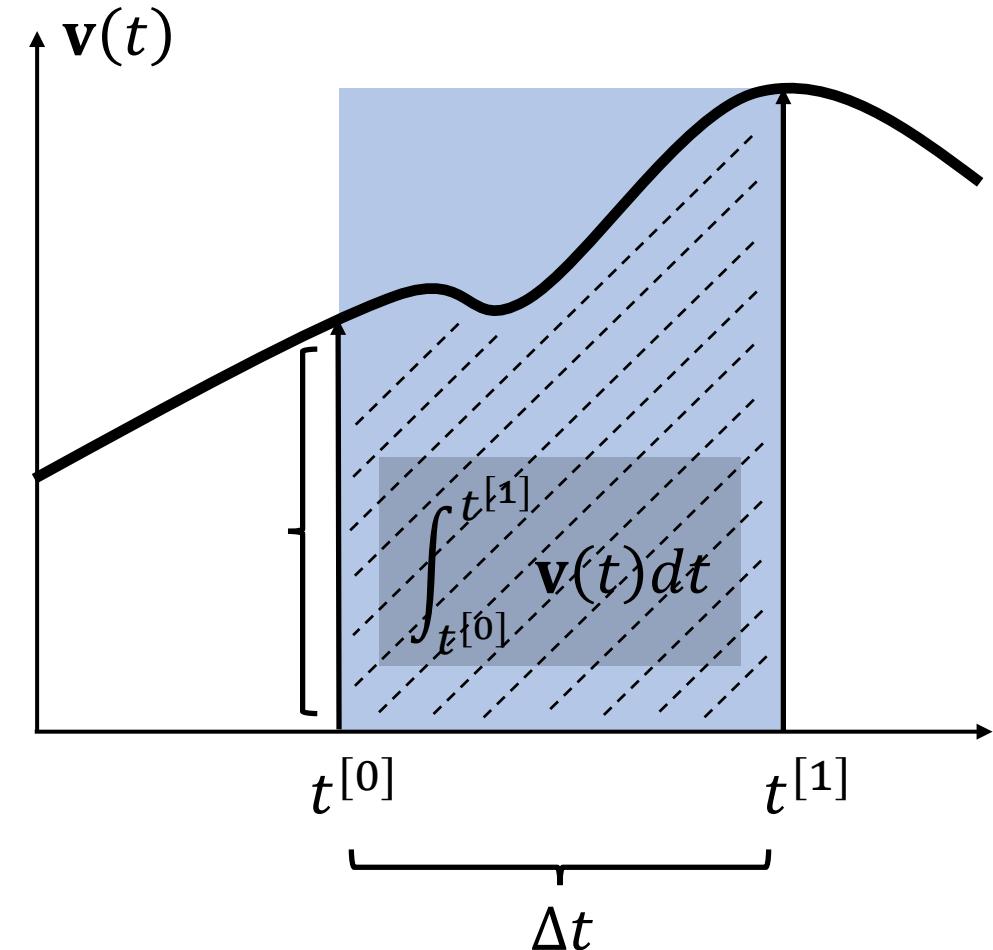
Implicit Euler (1st-order accurate) sets the height at $t^{[1]}$.

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

width height

$$\begin{aligned} \int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt &= \Delta t \mathbf{v}(t^{[1]}) - \frac{\Delta t^2}{2} \mathbf{v}'(t^{[1]}) + \dots \\ &= \Delta t \mathbf{v}(t^{[1]}) + O(\Delta t^2) \end{aligned}$$

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.

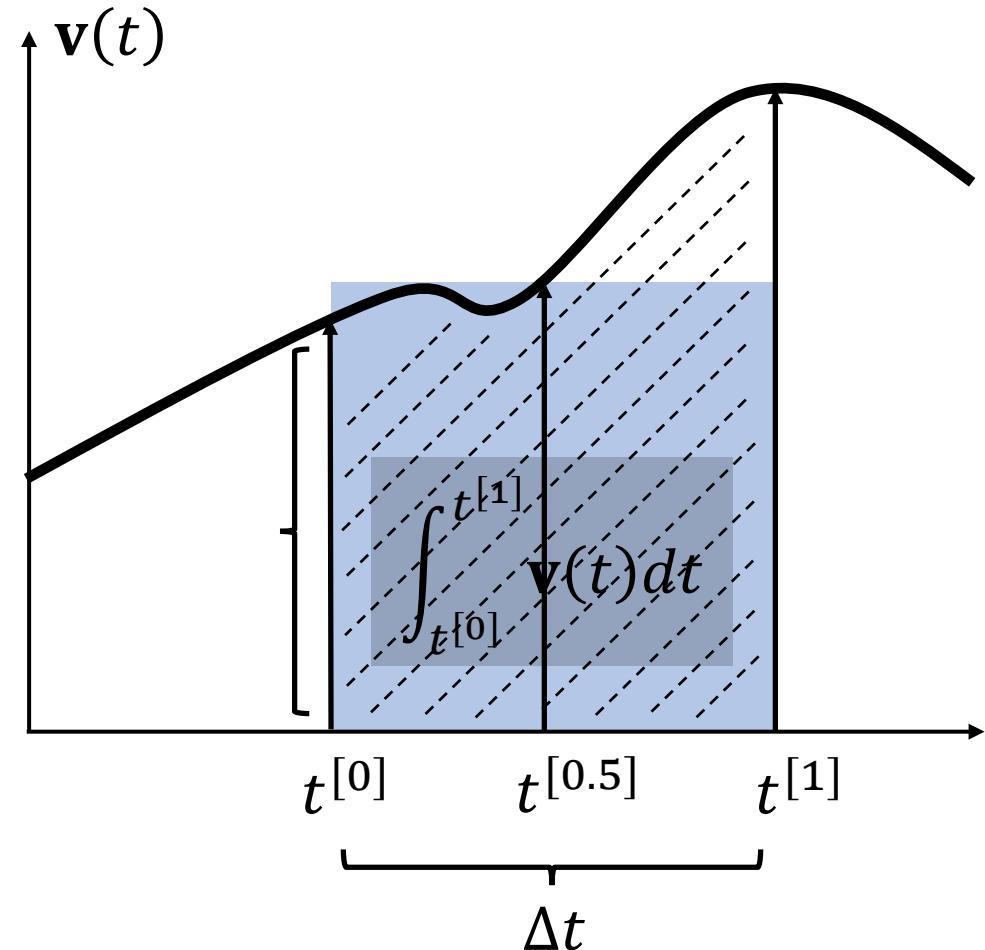
Mid-point (2nd-order accurate) sets the height at $t^{[0.5]}$.

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

width height

$$\begin{aligned} \int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt &= \int_{t^{[0]}}^{t^{[0.5]}} \mathbf{v}(t)dt + \int_{t^{[0.5]}}^{t^{[1]}} \mathbf{v}(t)dt \\ &= \frac{1}{2}\Delta t \mathbf{v}(t^{[0.5]}) - \frac{\Delta t^2}{2}\mathbf{v}'(t^{[0.5]}) + O(\Delta t^3) + \\ &\quad \frac{1}{2}\Delta t \mathbf{v}(t^{[0.5]}) + \frac{\Delta t^2}{2}\mathbf{v}'(t^{[0.5]}) + O(\Delta t^3) \\ &= \Delta t \mathbf{v}(t^{[0.5]}) + O(\Delta t^3) \end{aligned}$$

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 $t^{[0]}$.

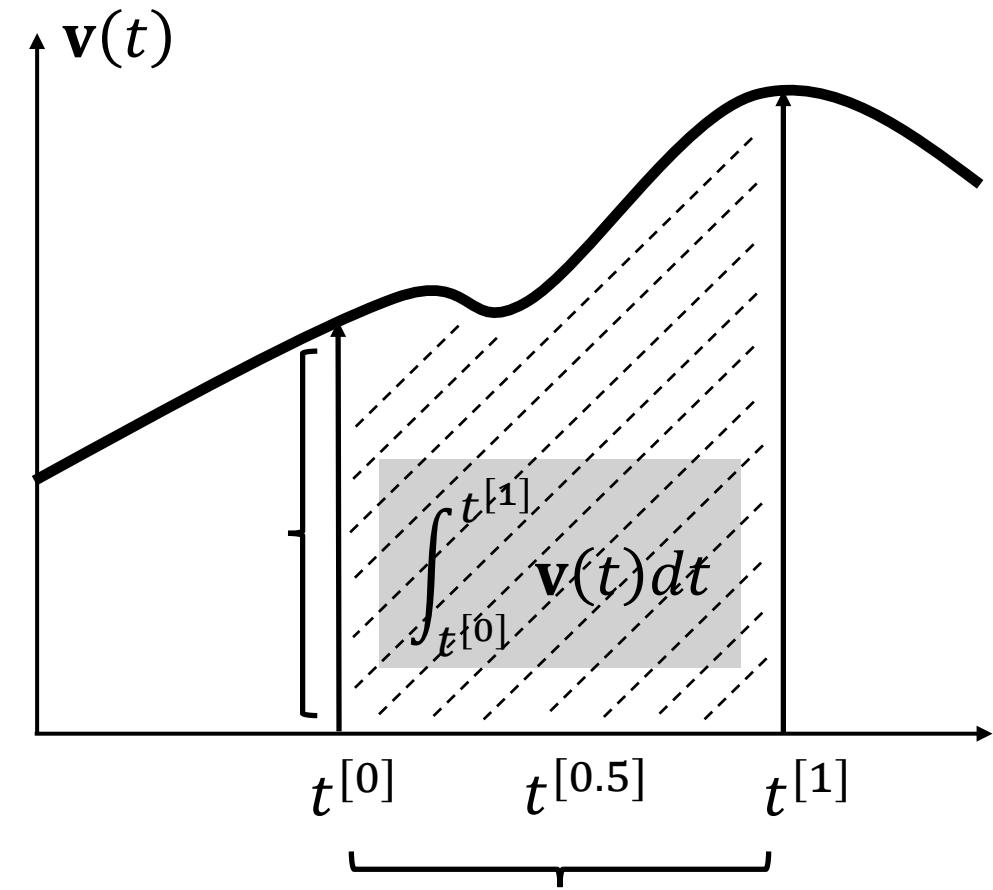
$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \Delta t \mathbf{v}(t^{[0]})$$

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

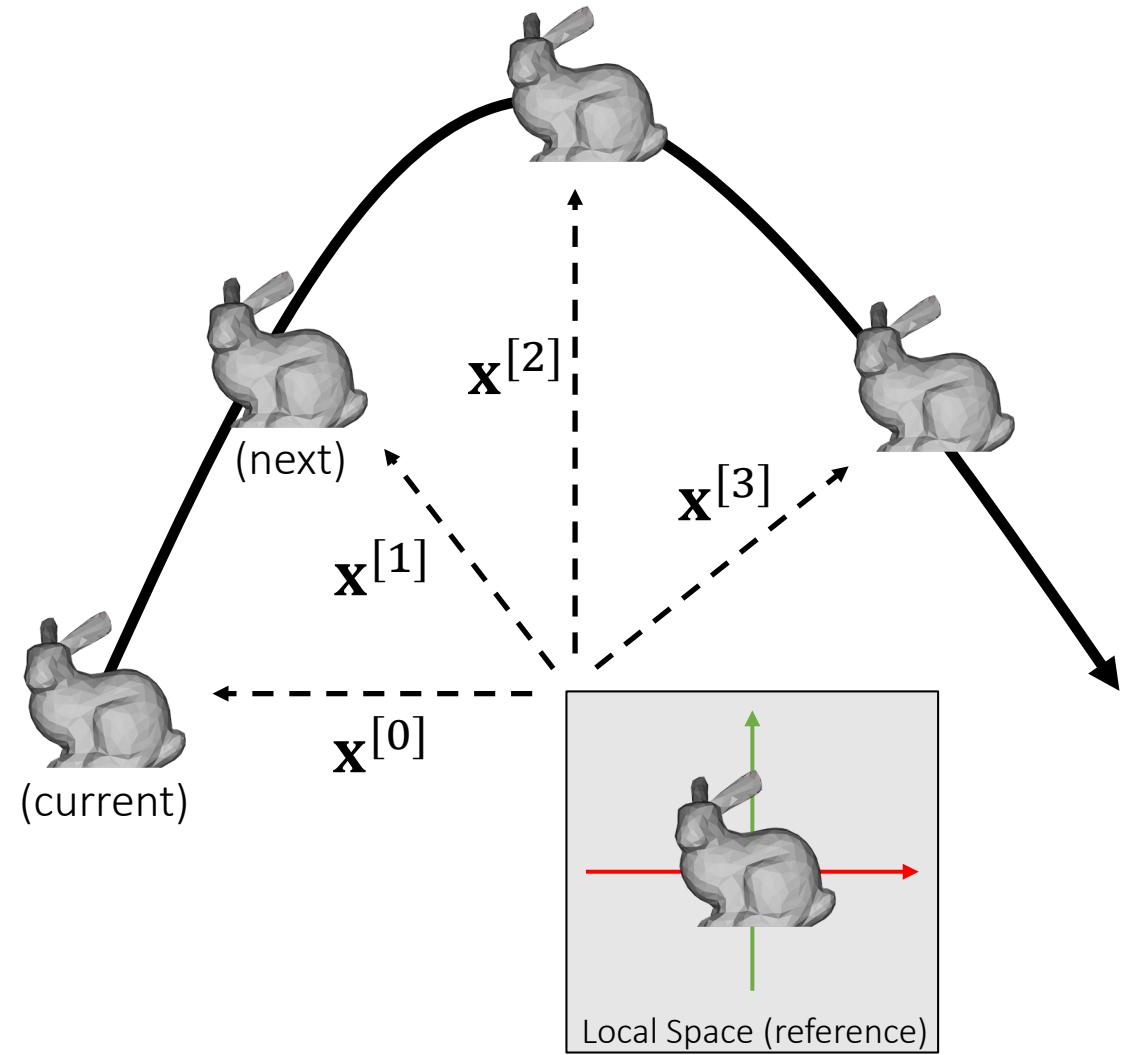
$$\int_{t^{[0]}}^{t^{[1]}} \mathbf{v}(t)dt \approx \Delta t \mathbf{v}(t^{[1]})$$

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} .

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

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*.

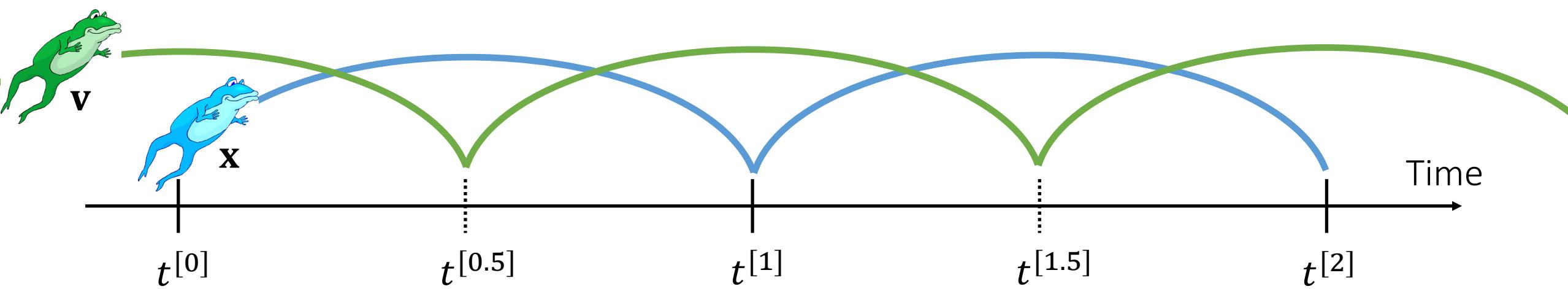
$$\begin{cases} \mathbf{v}^{[1]} = \mathbf{v}^{[0]} + \Delta t M^{-1} \mathbf{f}^{[0]} \\ \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]} \end{cases}$$

← Explicit
← Implicit

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

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

← Mid-point
← Mid-point



Types of Forces

Gravity Force:

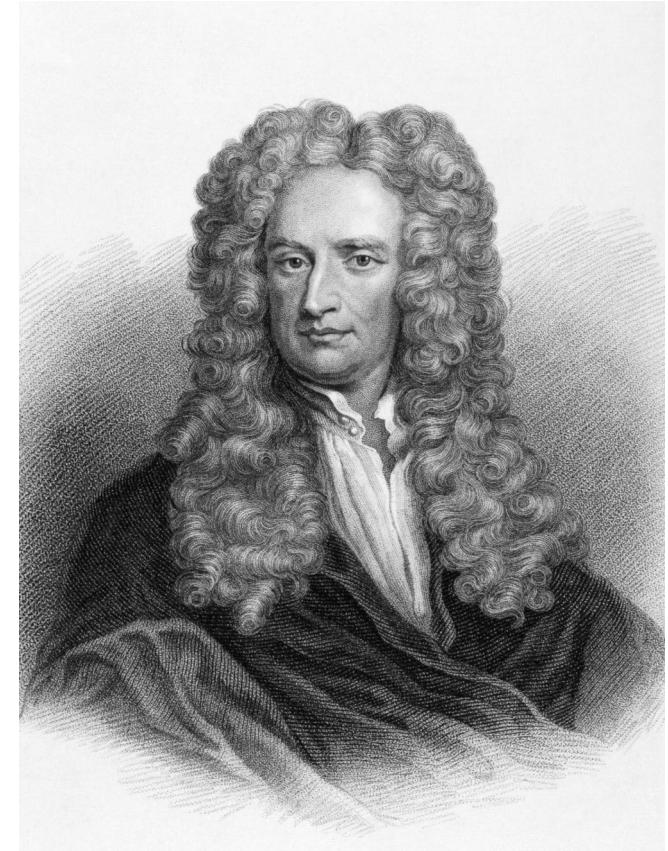
$$\mathbf{f}_{\text{gravity}}^{[0]} = M \mathbf{g}$$

gravity
mass

Drag Force:

$$\mathbf{f}_{\text{drag}}^{[0]} = -\sigma \mathbf{v}^{[0]}$$

velocity
drag coefficient

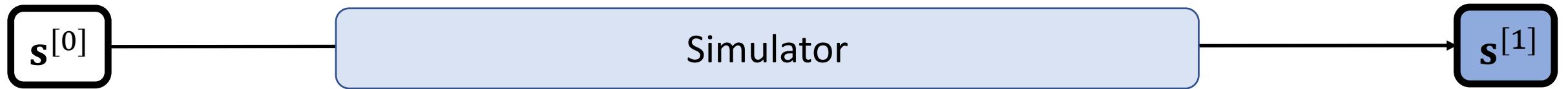


But since the drag reduces the velocity, a more popular way is to decay the velocity.

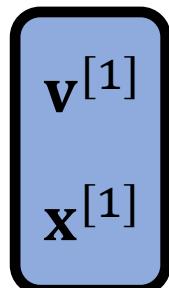
$$\mathbf{v}^{[1]} = \alpha \mathbf{v}^{[0]}$$

decay coefficient

Rigid Body Simulation (Translation Only)



$\mathbf{f}_i^{[0]} \leftarrow \text{Force}(\mathbf{x}_i^{[0]}, \mathbf{v}_i^{[0]})$
 $\mathbf{f}^{[0]} \leftarrow \sum \mathbf{f}_i^{[0]}$
 $\mathbf{v}^{[1]} \leftarrow \mathbf{v}^{[0]} + \Delta t M^{-1} \mathbf{f}^{[0]}$
 $\mathbf{x}^{[1]} \leftarrow \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]}$



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

Rotational Motion

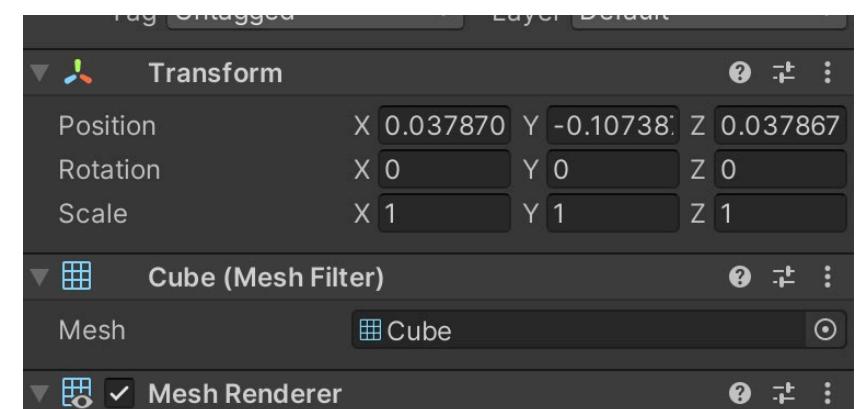
Rotation Represented by Matrix

- 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.

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

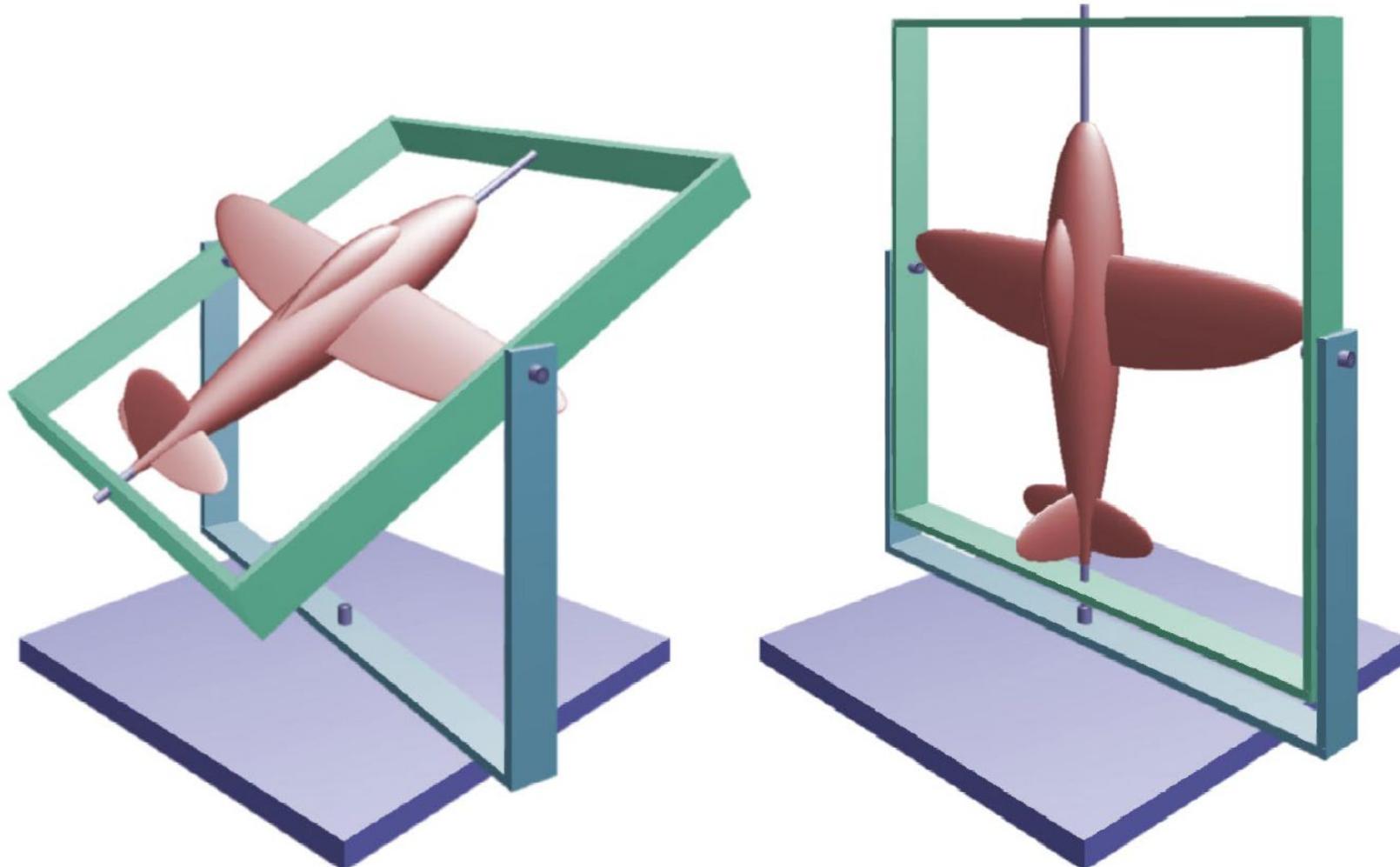
Rotation Represented by Euler Angles

- 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

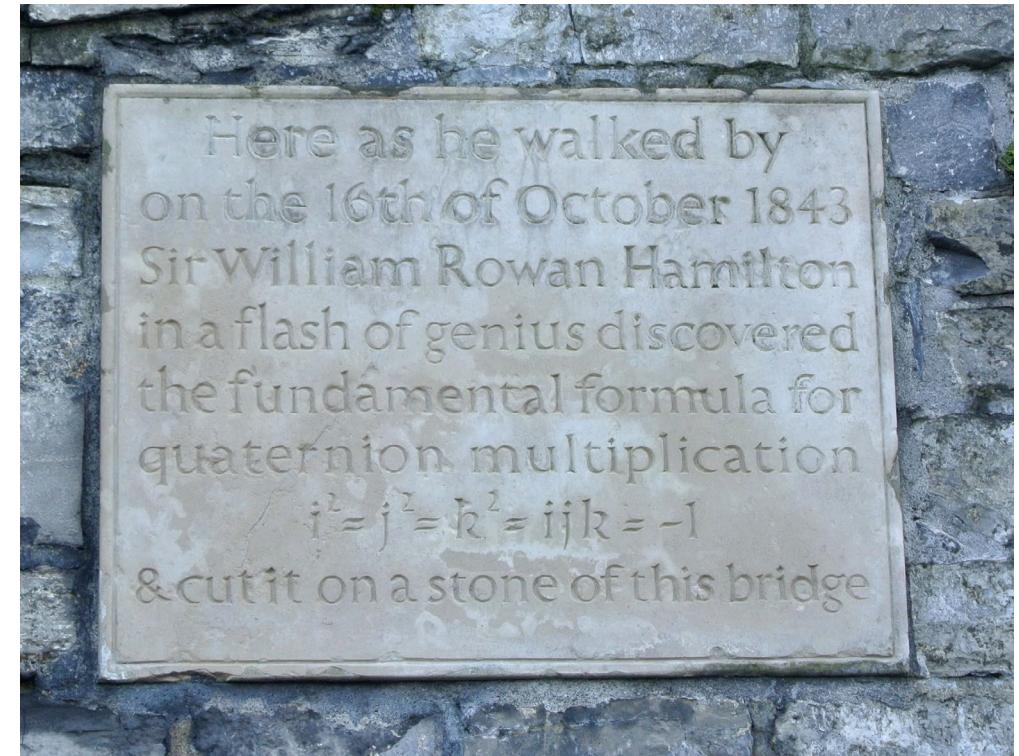
	1	i
1	1	i
i	i	-1

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 Arithmetic

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 $\mathbf{i}\mathbf{j}\mathbf{k}$.

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

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

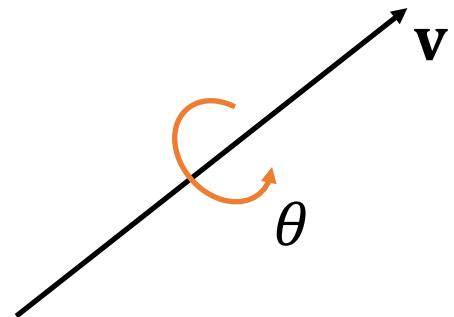
$$\mathbf{q}_1 \times \mathbf{q}_2 \quad \text{Multiplication}$$

$$= [s_1s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 \quad s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$$

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

Rotation Represented by Quaternion

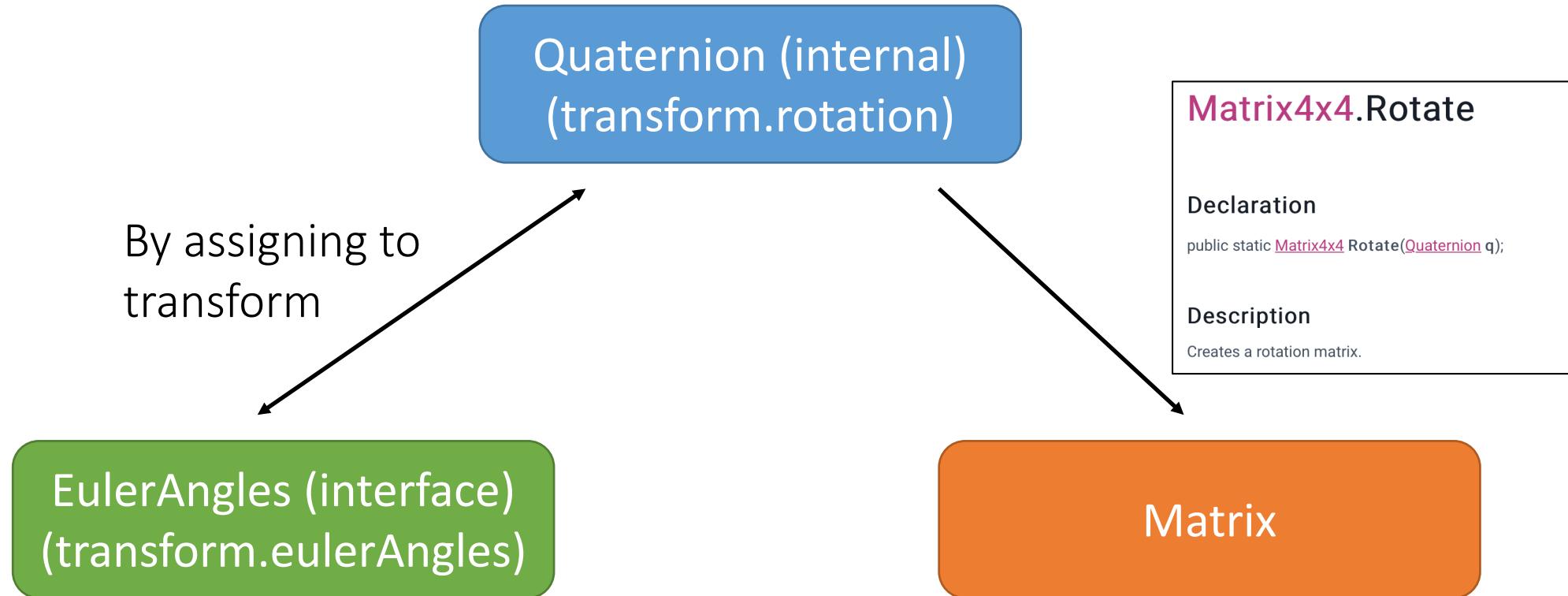
- To represent a rotation around \mathbf{v} by angle θ , we set the quaternion as:


$$\left\{ \begin{array}{l} \mathbf{q} = \left[\cos \frac{\theta}{2} \quad \mathbf{v} \right] \\ \|\mathbf{q}\| = 1 \end{array} \right. \rightarrow \left\{ \begin{array}{l} \mathbf{q} = \left[\cos \frac{\theta}{2} \quad \mathbf{v} \right] \\ \|\mathbf{v}\|^2 = \sin^2 \frac{\theta}{2} \end{array} \right.$$

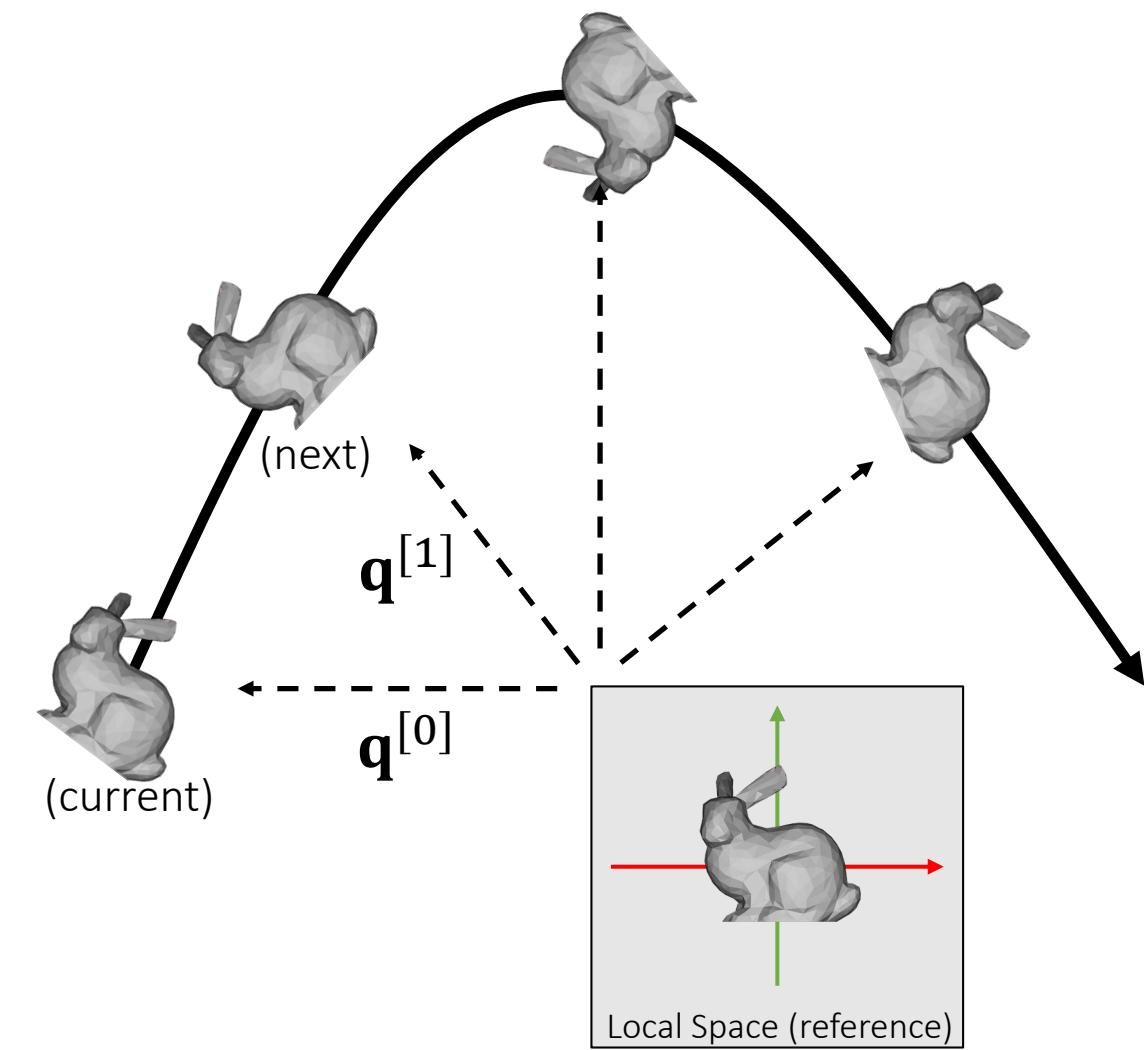
- It's very intuitive. It's the built-in representation in Unity.
- Convertible to the matrix:

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

Rotation Representations in Unity



Rotational Motion

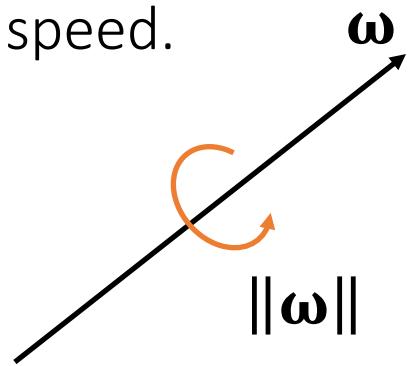


Now we choose quaternion \mathbf{q} to represent the orientation, i.e., the rotation from the *reference* to the *current*.

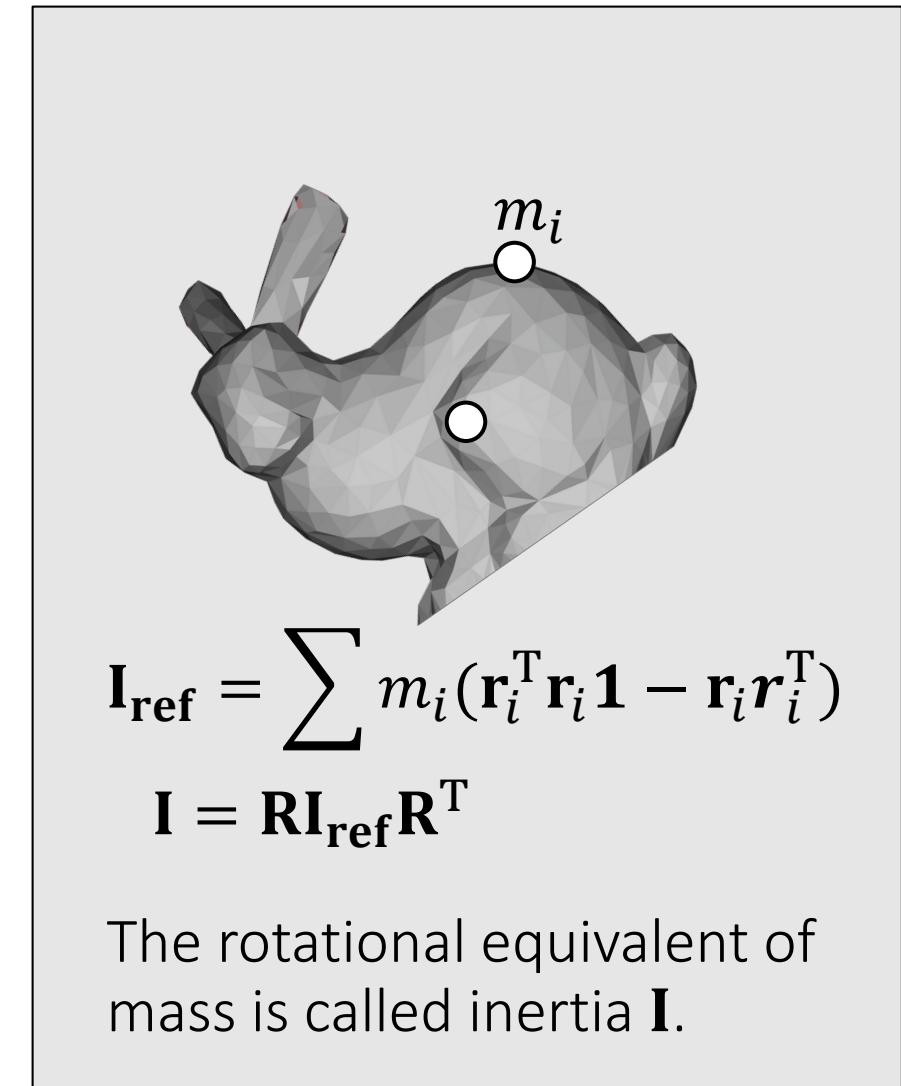
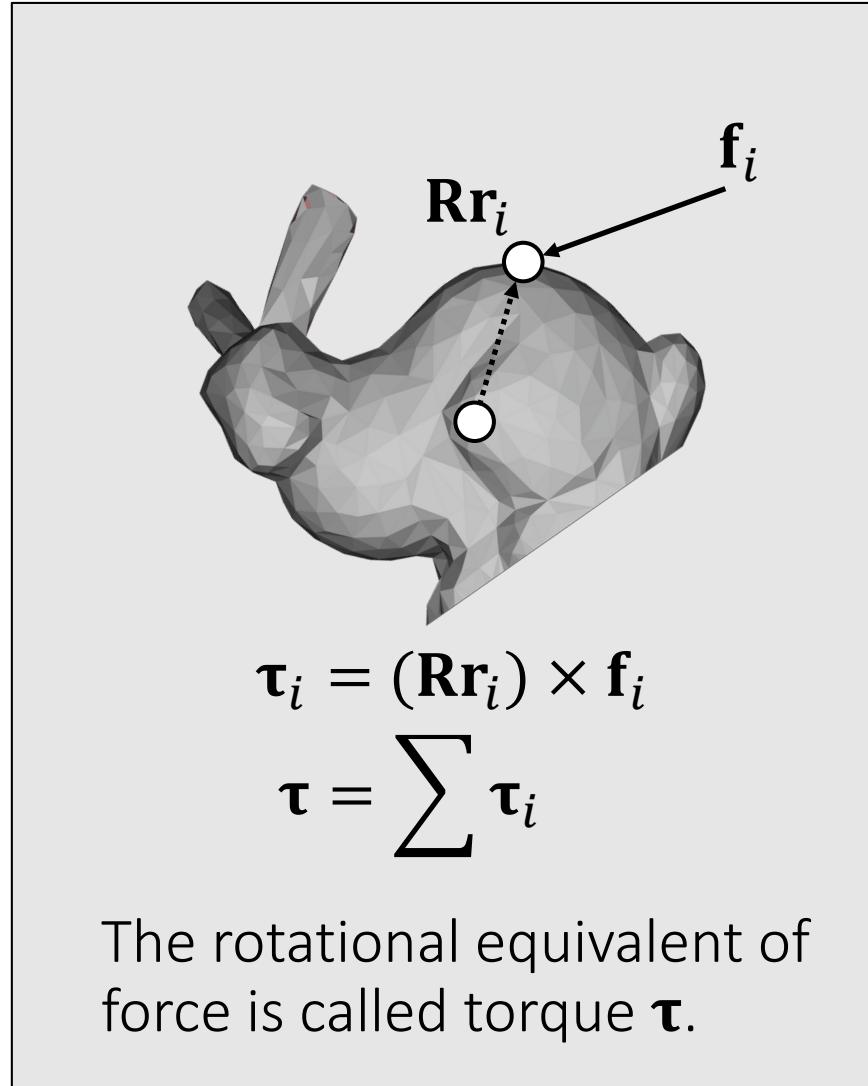
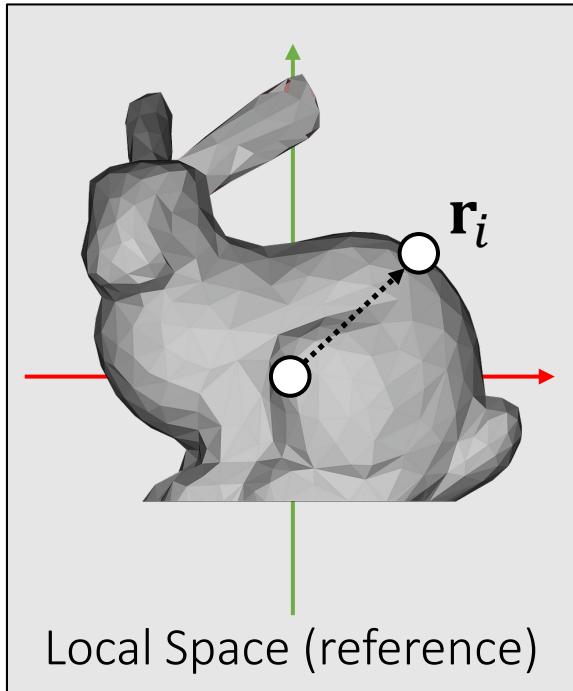
We use a 3D vector $\boldsymbol{\omega}$ to denote angular velocity.

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

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



Torque and Inertia



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)

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

States

Velocity \mathbf{v}

Position \mathbf{x}

(transform.position in Unity)

Angular velocity $\boldsymbol{\omega}$

Quaternion \mathbf{q}

(transform.rotation in Unity)

Physical Quantities

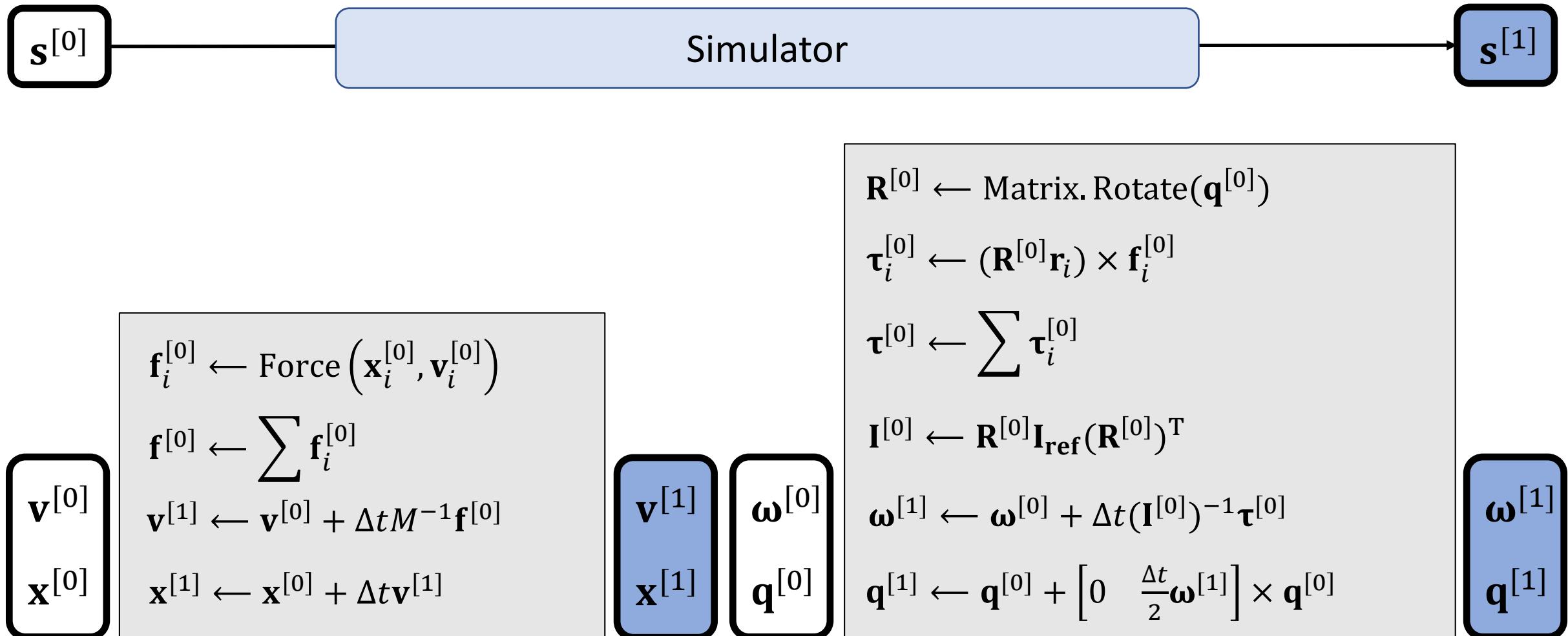
Mass M

Force \mathbf{f}

Inertia \mathbf{I}

Torque $\boldsymbol{\tau}$

Rigid Body Simulation



Implementation

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

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

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

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

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

v

x

v

x

ω

q

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

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

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

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

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

$$\mathbf{q} \leftarrow \mathbf{q} + \begin{bmatrix} 0 & \frac{\Delta t}{2} \boldsymbol{\omega} \end{bmatrix} \times \mathbf{q}$$

ω

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)



Physically Based Modeling

ONLINE SIGGRAPH 2001 COURSE NOTES

Please note: the lecture notes served from this page are copyright ©2001 by the authors Andrew Witkin and David Baraff. Chapters may be freely duplicated and distributed so long as no consideration is received in return, and this copyright notice remains intact. The slide sets are copyright ©2001 and may be neither distributed nor duplicated without permission of the authors.

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](#).

<https://graphics.pixar.com/pbm2001>

- [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](#)