

SPACE AND BODIES

SIMULATION, PHYSICS IN UNITY

TOMÁŠ POLÁŠEK IPOLASEK@FIT.VUT.CZ

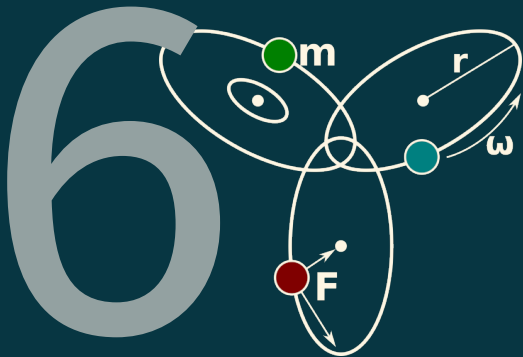
BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

DCGM, CPhoto@FIT

FACULTY OF FINE ARTS

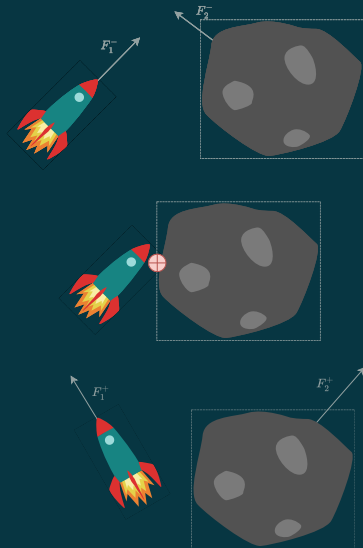
GAME MEDIA STUDIO



PHYSICAL SIMULATION

WHICH PHYSICS?

- Game Physics = Motion, Collision, Solve
- Goal: “Act as Expected” [2]
- Real-Time → Fake Everything [4]
- Focus on Special Cases:
 - Picking
 - Rigid Body Mechanics
 - Ragdoll



PHYSICS PRIMER

NEWTONIAN DYNAMICS

- Simple Approximation
- 3D space $\rightarrow \vec{\text{vec}}$ & **MAT**
- Variables: $\vec{F}$, m , $\vec{p}$, $\vec{s}$, $\vec{v}$, $\vec{a}$
- Laws of Motion [7]:

- ▶ Law of Inertia

$$\vec{F} = \vec{0} \Leftrightarrow \vec{v} = \vec{c}$$

- ▶ Force $\rightarrow$ Momentum

$$\vec{F} = m\vec{a}$$

- ▶ Action & Reaction

$$\vec{F}_1 = -\vec{F}_2$$

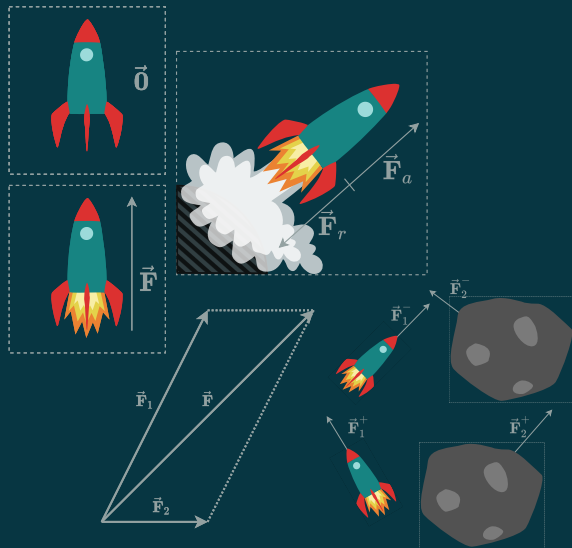
- Resolving Forces

$$\vec{F} = \vec{F}_1 + \vec{F}_2$$

- Conservation of Momentum

$$\vec{p} = m\vec{v}$$

$$m_1\vec{v}_1^- + m_2\vec{v}_2^- = m_1\vec{v}_1^+ + m_2\vec{v}_2^+$$



MOTION IN SPACE

■ Linear Motion: $\vec{a} \rightarrow \vec{v} \rightarrow \vec{s}$

- ▶ Force $\vec{F}$
- ▶ Mass m (vs Weight)
- ▶ Acceleration $\vec{a}$

■ Angular Motion: $\vec{\alpha} \rightarrow \vec{\omega} \rightarrow \vec{\theta}$

- ▶ Torque $\vec{\tau} \approx \text{Force}$

$$\vec{\tau} = \vec{d} \times \vec{F}$$
- ▶ Inertia $I \approx \text{Mass}$

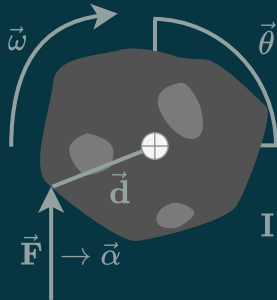
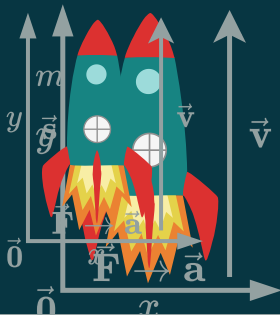
$$\vec{\tau} = I\vec{\alpha}$$
- ▶ Acceleration $\vec{\alpha}$

$$\vec{\alpha} = I^{-1}\vec{\tau}$$

$$\vec{\mathbf{F}} = m\vec{\mathbf{a}}$$

$$\vec{F} = m\vec{a} = m\vec{v}$$

$$\begin{aligned} \vec{p} &= m\vec{v} \\ \vec{s} &= \int \vec{v} dt \end{aligned}$$

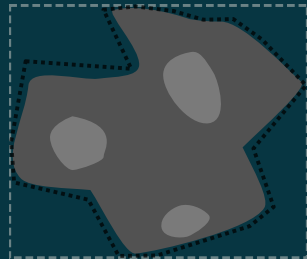
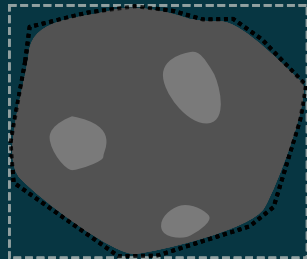
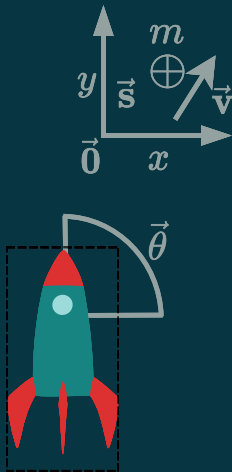


$$\vec{\tau} = \mathbf{I} \vec{\alpha}$$

$$\vec{\tau} = \vec{\mathbf{d}} \times \vec{\mathbf{F}}$$

ABSTRACT BODIES

- Level of Abstraction
- Shape Approximation
- Body Types:
 - Point Particle: $\vec{s}, \vec{m}, \vec{v}$
 - Rigid Body: + $\vec{\theta}$, shape
 - Soft Body: + deformation
- Universal Force → Gravity
- Inertia, Friction → Damping



SYSTEM CONSTRAINTS

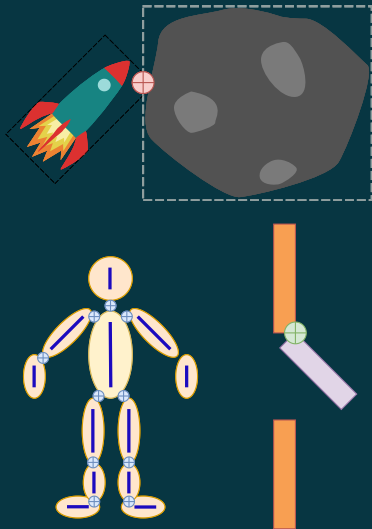
- Constraints → Interactions
- Explicit Limits: Strict × Loose
- Implicit Modification (Conservation)
- The Velocity Constraint:

$$C = f(\text{System}) \rightarrow 0$$

$$\dot{C} = J \cdot \vec{v}$$

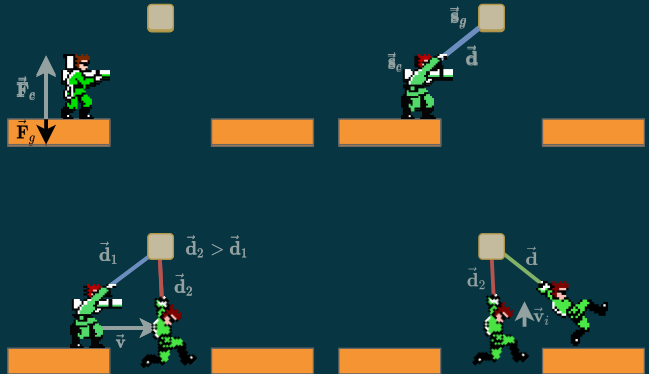
$$\vec{F} = J^T \lambda$$

- Collision & Interface
- Distance Constraint
- Hinge Constraint



SOLVING THE PHYSICS

- Simulated System
- Bodies & Constraints
- Linear / Angular **Motion**
- **Integrator**
- Systems of **Constraints**
- **Solver**: Global × Iterative



■ Integration → Iteration

■ Numerical Integration

■ Time Step Δt

■ Granularity $\times$ Precision

■ Common Techniques:

- ▶ Explicit Euler's
- ▶ Semi-Implicit Euler's
- ▶ Verlet
- ▶ Midpoint
- ▶ Runge-Kutta

$$\vec{v} = \int \vec{a} dt$$

$$\vec{s} = \int \vec{v} dt$$

↓

$$\vec{v}_{n+1} = \vec{v}_n + \vec{a}_n \Delta t$$

$$\vec{s}_{n+1} = \vec{s}_n + \vec{v}_n \Delta t$$

EULER'S INTEGRATION

■ Explicit Approach:

- ▶ Direct Calculation
- ▶ Instability → Frequency
- ▶ Simple Iteration

■ Implicit Approach:

- ▶ Future Time
- ▶ Expensive Approximation
- ▶ Domain Knowledge

■ Semi-Implicit Approach:

- ▶ Hybrid Time
- ▶ Improved Stability
- ▶ Most Common

Explicit

$$\vec{v}_{n+1} = \vec{v}_n + \underline{\vec{a}_n} \Delta t$$

$$\vec{s}_{n+1} = \vec{s}_n + \underline{\vec{v}_n} \Delta t$$

Implicit

$$\vec{v}_{n+1} = \vec{v}_n + \underline{\vec{a}_{n+1}} \Delta t$$

$$\vec{s}_{n+1} = \vec{s}_n + \underline{\vec{v}_{n+1}} \Delta t$$

Semi-Implicit

$$\vec{v}_{n+1} = \vec{v}_n + \underline{\vec{a}_n} \Delta t$$

$$\vec{s}_{n+1} = \vec{s}_n + \underline{\vec{v}_{n+1}} \Delta t$$

HIGHER ORDER APPROACHES

■ Verlet Integration:

- ▶ *Sine* Velocity
- ▶ Based on SI Euler (Init!)
- ▶ Reversible + Positions

■ Midpoint Method (RK2):

- ▶ Continuous Change → Look-Ahead
- ▶ Semi-Implicit Technique
- ▶ Enhanced Precision

■ Fourh Order Runge-Kutta (RK4):

- ▶ Multi-Look-Ahead
- ▶ Greater Precision
- ▶ Computation Complexity

Verlet

$$\vec{s}_{n+1} = \vec{s}_n + \vec{v}_n \Delta t + \vec{a}_n \Delta t^2$$

$$\vec{v}_n = \frac{\vec{s}_n - \vec{s}_{n-1}}{\Delta t}$$

Midpoint

$$\vec{v}_{n+0.5} = v(\vec{s}_n + \frac{\Delta t}{2} \vec{v}_n, t_n + \frac{\Delta t}{2})$$

$$\vec{s}_{n+1} = \vec{s}_n + \vec{v}_{n+0.5} \Delta t$$

RK4

$$\vec{k}_1 = v(\vec{s}_n, t_n)$$

$$\vec{k}_2 = v(\vec{s}_n + \frac{\Delta t}{2} \vec{k}_1, t_n + \frac{\Delta t}{2})$$

$$\vec{k}_3 = v(\vec{s}_n + \frac{\Delta t}{2} \vec{k}_2, t_n + \frac{\Delta t}{2})$$

$$\vec{k}_4 = v(\vec{s}_n + \frac{\Delta t}{2} \vec{k}_3, t_n + \Delta t)$$

$$\vec{s}_{n+1} = \vec{s}_n + (\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \frac{\Delta t}{6}$$

CONSTRAINT SOLVER

- **Solver:** Systems of Equations
- Degrees of Freedom
- Global × Iterative
- Slow Convergence → Bias

$$\dot{\mathbf{C}} = \mathbf{J} \cdot \vec{\mathbf{v}}$$

⇓

$$\dot{\mathbf{C}} = \mathbf{J} \cdot \vec{\mathbf{v}} + \vec{\mathbf{b}}; \vec{\mathbf{b}} = \frac{\beta}{\Delta t} \mathbf{C}$$

```
Solution SolveGlobal(Constraints constraints)
{
    return Solver.solve(constraints);
}

Solution SolveIterative(Constraints constraints)
{
    var solution = initializeSolution(constraints);
    for (var step = 0; step < STEPS; ++step)
    {
        foreach (var constraint in constraints)
        { solution = Solver.solve(constraint, solution); }
    }
    return solution;
}
```

SOLVING CONSTRAINTS

■ Projection Method [6]

- ▶ Update Position
- ▶ Temporary Solution
- ▶ Consider Properties

■ Impulse Method [6]

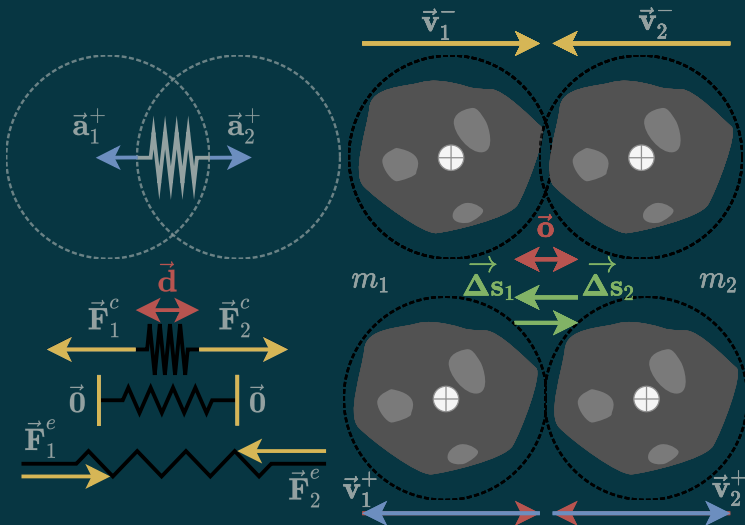
- ▶ Apply Impulse $\vec{v}$
- ▶ Momentum $\vec{p} = m\vec{v}$
- ▶ Restitution Coefficient

$$e = \frac{|\vec{v}_1^+ - \vec{v}_2^+|}{|\vec{v}_1^- - \vec{v}_2^-|}$$

■ Penalty Method [7]

- ▶ Higher Derivative
- ▶ Apply Acceleration $\vec{a}$
- ▶ Spring Tension

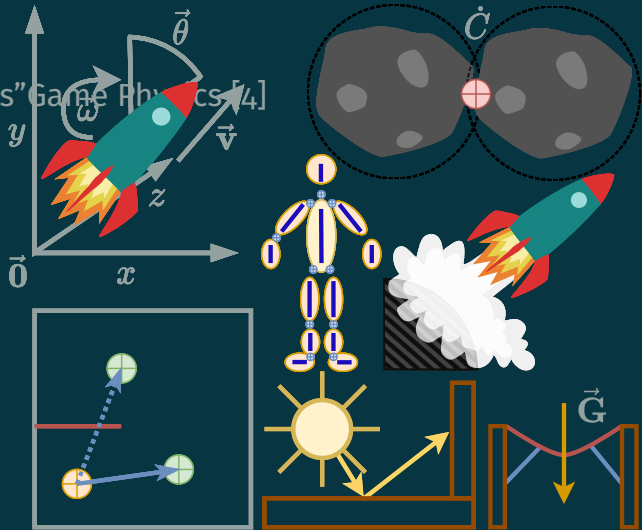
$$\vec{F} = -k\vec{d}$$



PHYSICS ENGINE

PHYSICS SUBSYSTEM

- **Goal:** Simulate Physics“Physics” Game Physics [4]
 - ▶ Linear & Angular Motion
 - ▶ Collisions, Constraints
 - ▶ Special Effects
- Physics Engine → Interaction
- Support Functions
 - ▶ Spatial Queries
 - ▶ Visibility, Raycasting
 - ▶ Gameplay



GAME PHYSICS OVERVIEW

■ Phases of Operation:

1. Body Motion
2. Detect Collisions
3. Resolve Constraints

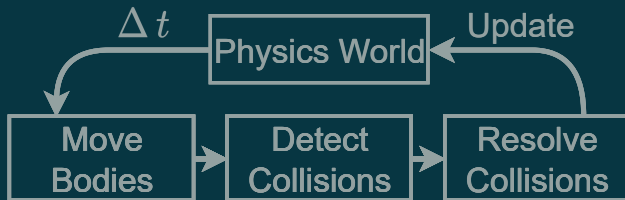
■ The Physics World

■ Update Loop

■ Time Step

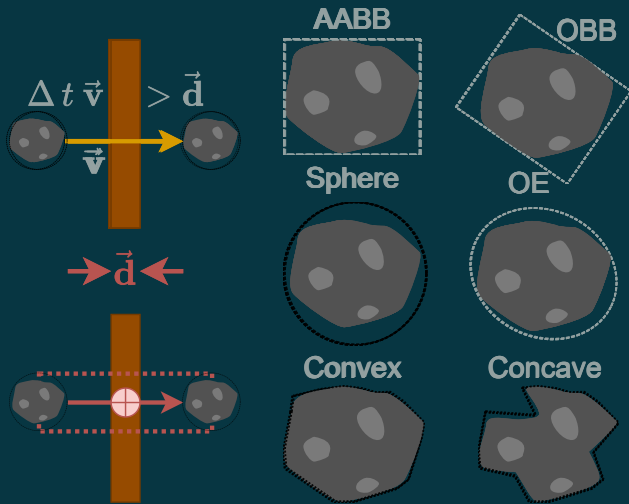
■ Separate System

■ → Physics Engine



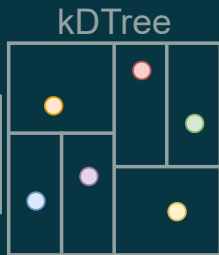
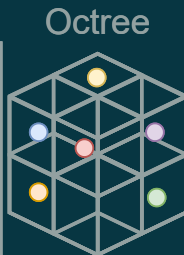
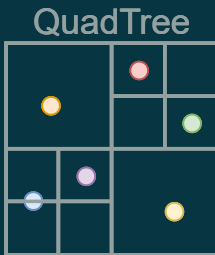
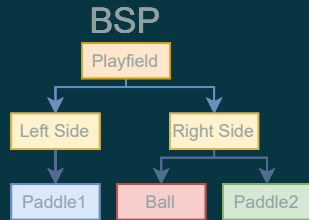
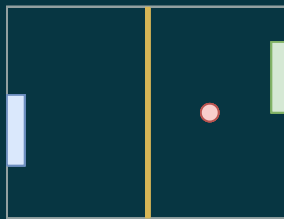
COLLISION DETECTION

- Primary Instigator
- Complex Game World
- Collision Volumes
 - ▶ Axis-Aligned BB
 - ▶ Oriented BB
 - ▶ Sphere & Ellipse
 - ▶ Polygon
- Broad & Narrow Phase
- Considerations
 - ▶ Discrete $\neq$ Continuous
 - ▶ Timestep Δt
 - ▶ Volume Shapes



GOING BROAD AND NARROW

- Collision Detection
- Complexity → Broad & Narrow
- Bounding Volume Hierarchy
 - BSP & BSP Tree
 - Quadtree
 - Octree
 - kD-Tree
- Choosing BVH
- The Narrow Phase
 - Contact Points
 - Collision Normals
 - Distances



RESOLVING CONSTRAINTS

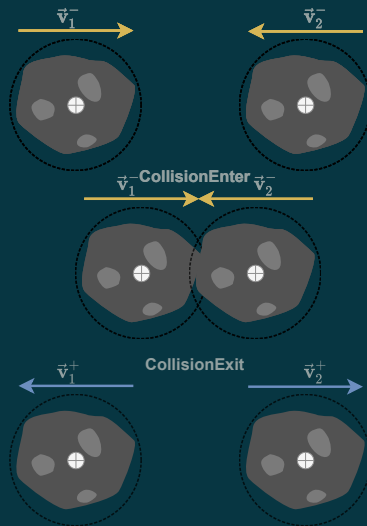
■ Collision Response

- ▶ Sequential Impulse [1]
- ▶ Projected Gauss-Seidel [5]
- ▶ Temporal Gauss-Seidel [3]

■ Physics Response

■ Gameplay Response

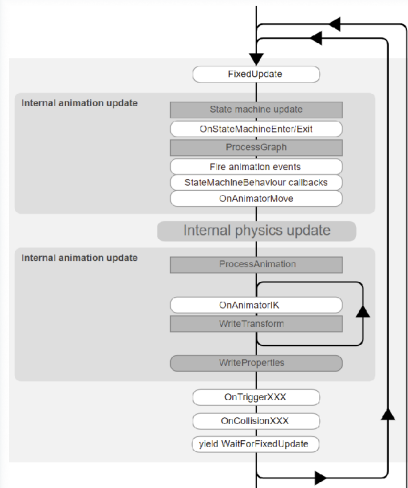
- ▶ Action Tags
- ▶ Simulation Events
- ▶ Enter/Exit Callbacks



PHYSICS IN UNITY

PHYSICS OVERVIEW

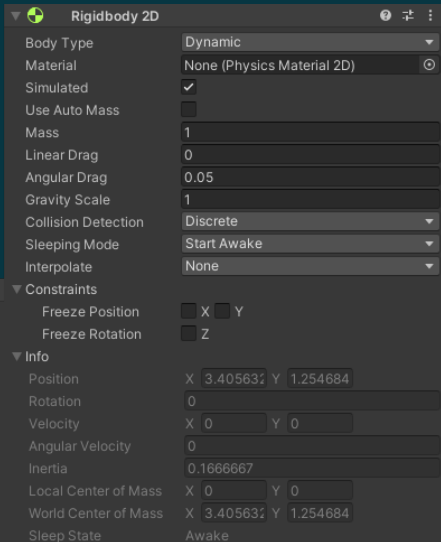
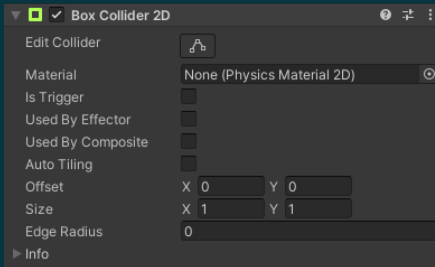
- Integrated Physics Engine
- Many Options
 - ▶ Unity Physics 2D
 - ▶ Unity Physics 3D
 - ▶ DOTS Physics
 - ▶ Havok Physics
- Future DOTS Integration



Source: Unity Documentation: Execution Order

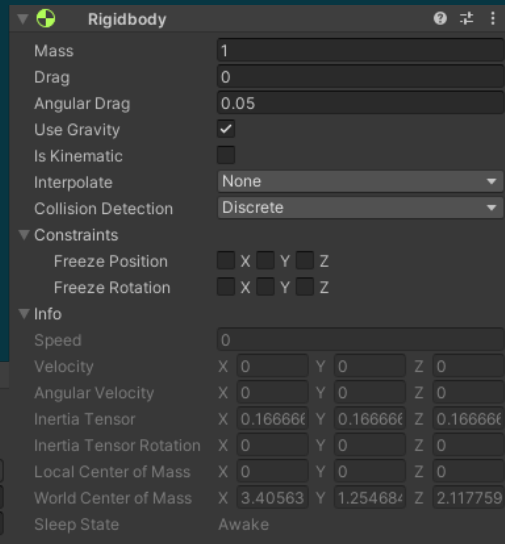
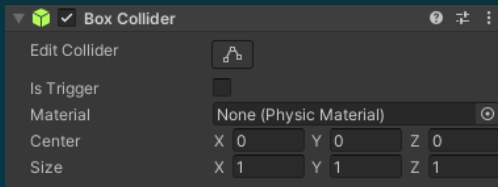
UNITY PHYSICS 2D

- Default 2D Solution
- Rigidbody2D
- Collider2D
 - ▶ BoxCollider2D
 - ▶ CircleCollider2D
 - ▶ PolygonCollider2D
 - ▶ EdgeCollider2D



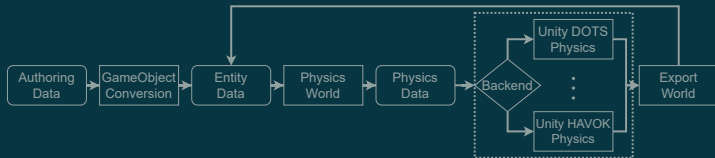
UNITY PHYSICS 3D

- Default 3D Solution
- Rigidbody
- Collider
 - ▶ BoxCollider
 - ▶ SphereCollider
 - ▶ CapsuleCollider
 - ▶ MeshCollider

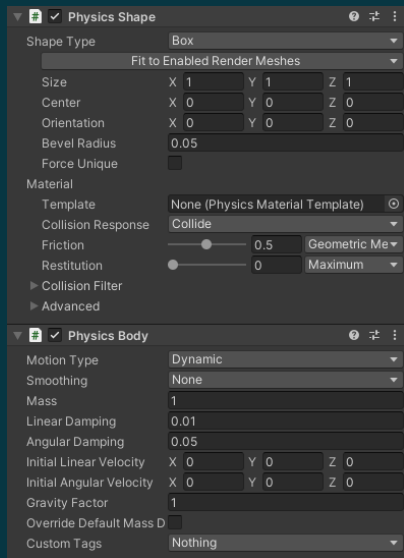


DOTS UNITY PHYSICS

- New Approach
- Using DOTS → ECS
- Physics Body Authoring
- Physics Shape Authoring
- DOTS Physics Overview
- Custom Simulators

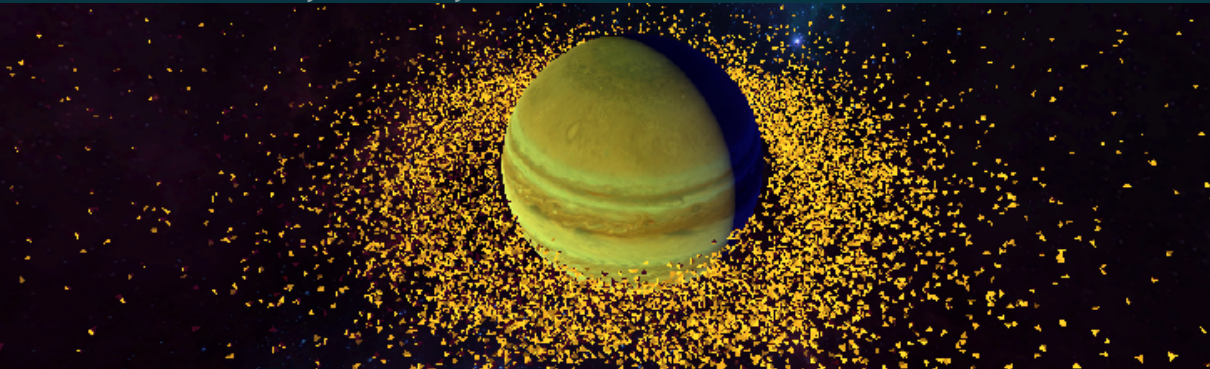


Source: Overview of Unity Physics



ADDITIONAL RESOURCES

- [Book] Kenny Erleben : Physics-Based Animation
- [YouTube] Adam Mechtley : Overview of physics in DOTS
- [YouTube] Steve Ewart : Overview of Havok Physics in Unity + Comparison
- [YouTube] Unity : Cloth Physics



Source: Unity Entity Component System Samples

Thanks For
Your Attention!



World of Goo

REFERENCES I

- [1] MING-LUN CHOU. **GAME PHYSICS SERIES**.
<http://allenchou.net/game-physics-series/>. 2013.
- [2] JASON GREGORY. **GAME ENGINE ARCHITECTURE, SECOND EDITION**. 3rd. USA: A. K. Peters, Ltd., CRC Press, 2018. ISBN: 1351974288.
- [3] MILES MACKLIN ET AL. **“SMALL STEPS IN PHYSICS SIMULATION”**. In: *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '19. Los Angeles, California: Association for Computing Machinery, 2019. ISBN: 9781450366779. DOI: 10.1145/3309486.3340247. URL: <https://doi.org/10.1145/3309486.3340247>.
- [4] IAN MILLINGTON. **GAME PHYSICS ENGINE DEVELOPMENT**. 2nd. USA: M. K. Publishers Inc., 2010. ISBN: 0123819768.
- [5] JOHAN SUNDBERG. **PARALLEL PROJECTED GAUSS-SEIDEL SOLVER FOR LARGE-SCALE GRANULAR MATTER**. 2014.

REFERENCES II

- [6] MARIJN TAMIS. ***COMPARISON BETWEEN PROJECTED GAUSS SEIDEL AND SEQUENTIAL IMPULSE SOLVERS FOR REAL-TIME PHYSICS SIMULATIONS.*** 2015.
- [7] NEWCASTLE UNIVERSITY. ***GAME TECHNOLOGIES COURSE.***
<https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/>.
2021.