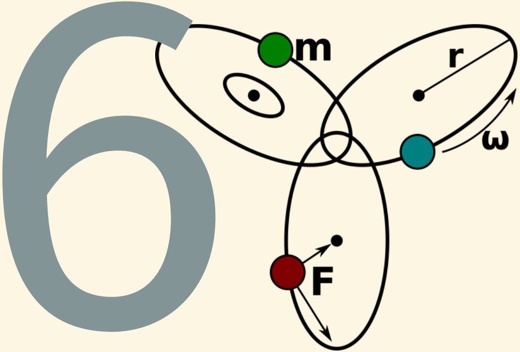
Space and Bodies

Simulation, Physics in Unity

Tomáš Polášek ipolasek@fit.vutbr.cz

Brno University of Technology

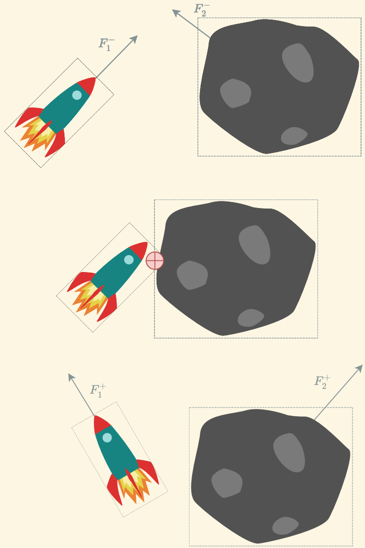
Faculty of Information Technology
DCGM, CPhoto@FIT
Faculty of Fine Arts
Game Media Studio



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

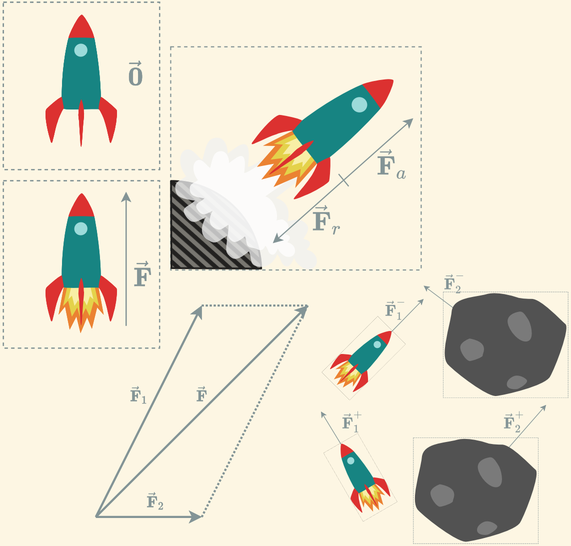


Physics Primer

Newtonian Dynamics

- Simple Approximation
- 3D space → $\overrightarrow{\textbf{vec}}$ & **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 → 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
  $$\vec{p} = m\vec{v}$$

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

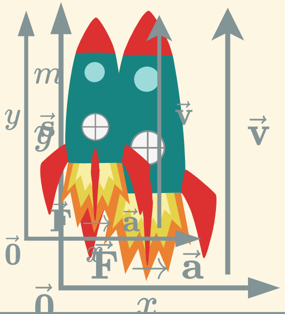


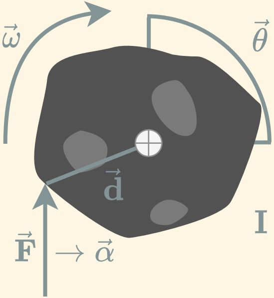
- Linear Motion: $\vec{a} \rightarrow \vec{v} \rightarrow \vec{s}$
  - Force $\vec{F}$
  - Mass m (vs Weight)
  - Acceleration $\vec{a}$
- Angular Motion: $\vec{\alpha} \rightarrow \vec{\omega} \rightarrow \vec{\theta}$
  - Torque $\vec{\tau} \approx$ Force
    $$\vec{\tau} = \vec{d} \times \vec{F}$$
  - Inertia $I \approx$ Mass
    $$\vec{\tau} = I\vec{\alpha}$$
  - Acceleration $\vec{\alpha}$
    $$\vec{\alpha} = I^{-1}\vec{\tau}$$

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

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

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



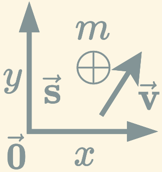


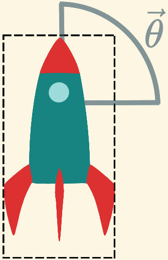
$$\vec{\tau} = \mathbf{I}\,\vec{\alpha}$$

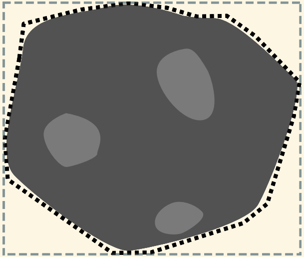
$$\vec{\tau} = \vec{\mathbf{d}} \times \vec{\mathbf{F}}$$

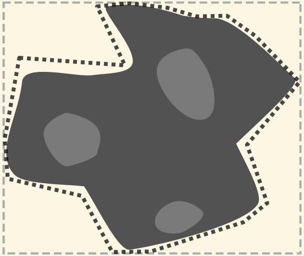
Abstract Bodies

- 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









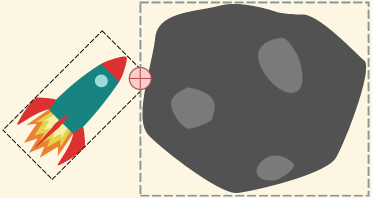
- Constraints → Interactions
- Explicit Limits: Strict × Loose
- Implicit Modification (Conservation)
- The Velocity Constraint:

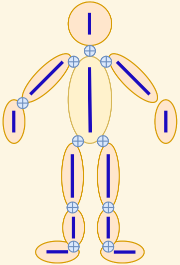
$$C = f(\text{System}) \to 0$$

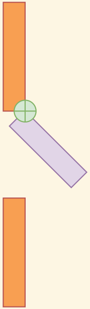
$$\dot{C} = \boldsymbol{J} \cdot \vec{\boldsymbol{v}}$$

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

- Collision & Interface
- Distance Constraint
- Hinge Constraint





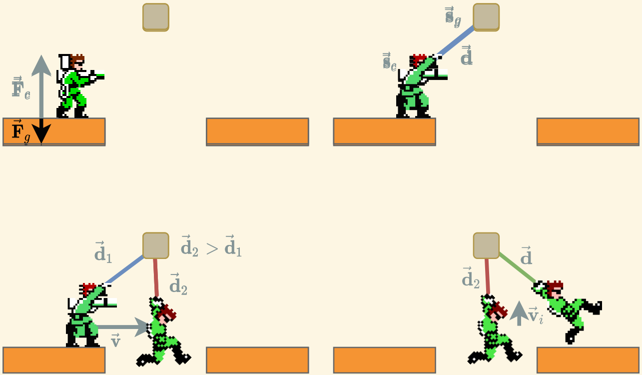


Solving the Physics

- Simulated System
- Bodies & Constraints

- Linear / Angular **Motion**
- **Integrator**

- Systems of **Constraints**
- **Solver**: Global × Iterative



- **Integration $\rightarrow$ Iteration**
- Numerical Integration
- **Time Step Δt**
- Granularity $\times$ Precision
- Common Techniques:
  - Explicit Euler's
  - Semi-Implicit Euler's
  - Verlet
  - Midpoint
  - Runge-Kutta

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

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

$$\Downarrow$$

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

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

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

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

- 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

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

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

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

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

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

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

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

$$\Downarrow$$

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

```
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;
}
```

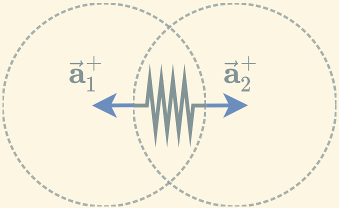
Solving Constraints

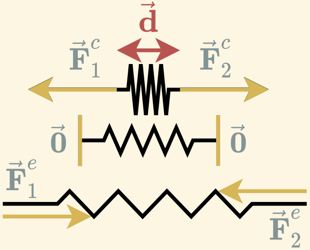
- **Projection Method [6]**
  - Update Position
  - Temporary Solution
  - Consider Properties
- **Impulse Method [6]**
  - Apply Impulse $\vec{v}$
  - Momentum $\vec{p} = m\vec{v}$
  - Restitution Coefficient

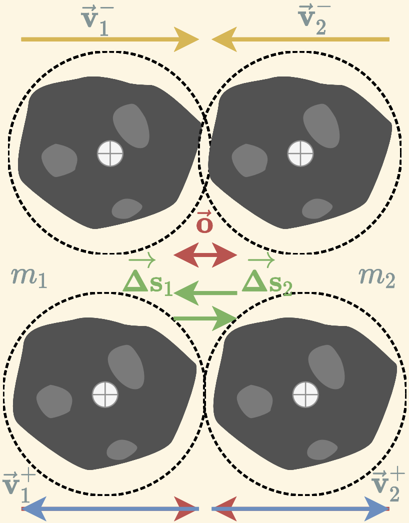
$$e = \frac{|\vec{v}_1^{+} - \vec{v}_2^{+}|}{|\vec{v}_1^{-} - \vec{v}_2^{-}|}$$

- **Penalty Method [7]**
  - Higher Derivative
  - Apply Acceleration $\vec{a}$
  - Spring Tension

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

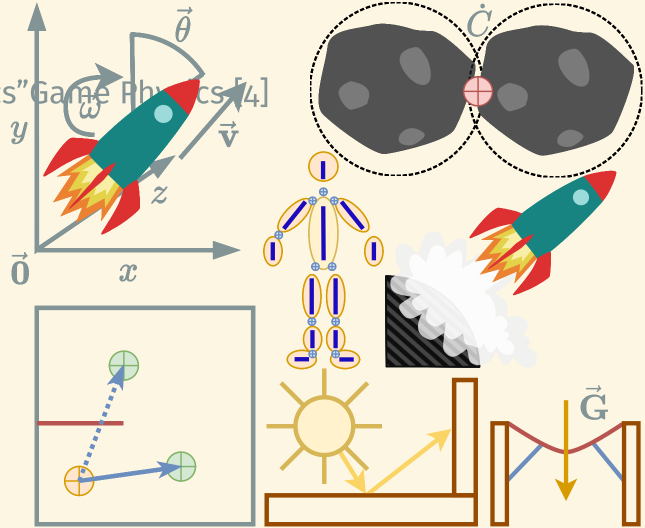




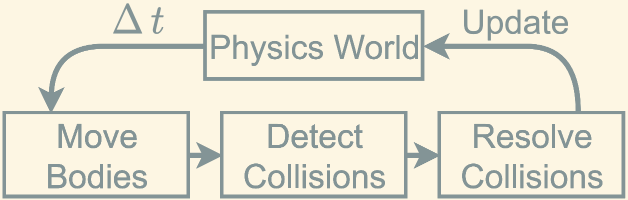


Physics Engine

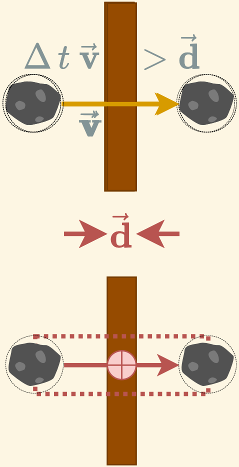
- **Goal**: Simulate Physics“Physics”Game Physics [4]
  - Linear & Angular Motion
  - Collisions, Constraints
  - Special Effects
- Physics Engine → Interaction
- Support Functions
  - Spatial Queries
  - Visibility, Raycasting
  - Gameplay

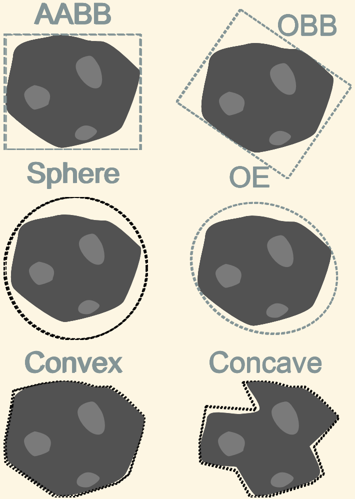


- Phases of Operation:
  1. Body Motion
  2. Detect Collisions
  3. Resolve Constraints
- The Physics World
- Update Loop
- **Time Step**
- Separate System
- → Physics Engine

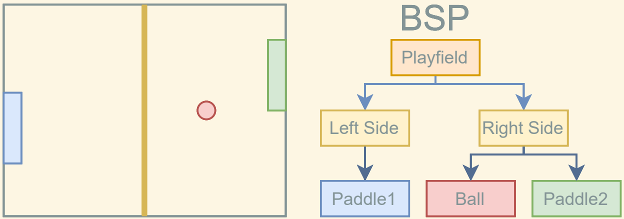


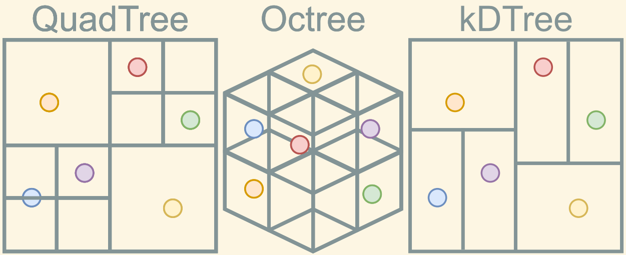
- Primary Instigator
- Complex Game World
- Collision Volumes
  - Axis-Aligned BB
  - Oriented BB
  - Sphere & Ellipse
  - Polygon
- Broad & Narrow Phase
- Considerations
  - Discrete $\neq$ Continuous
  - Timestep Δt
  - Volume Shapes



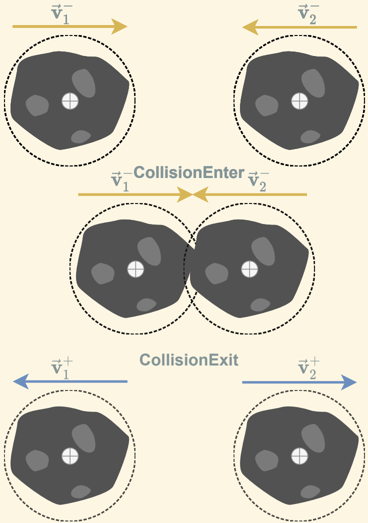


- Collision Detection
- Complexity → Broad & Narrow
- Bounding Volume Hierarchy
  - BSP & BSP Tree
  - Quadtree
  - Octree
  - kD-Tree
- Choosing BVH
- The Narrow Phase
  - Contact Points
  - Collision Normals
  - Distances



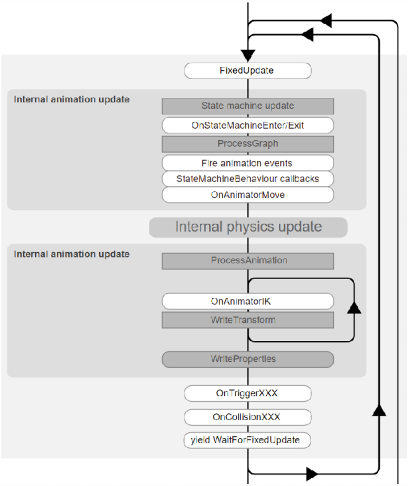


- Collision Response
  - Sequential Impulse [1]
  - Projected Gauss-Seidel [5]
  - Temporal Gauss-Seidel [3]
- Physics Response
- Gameplay Response
  - Action Tags
  - Simulation Events
  - Enter/Exit Callbacks



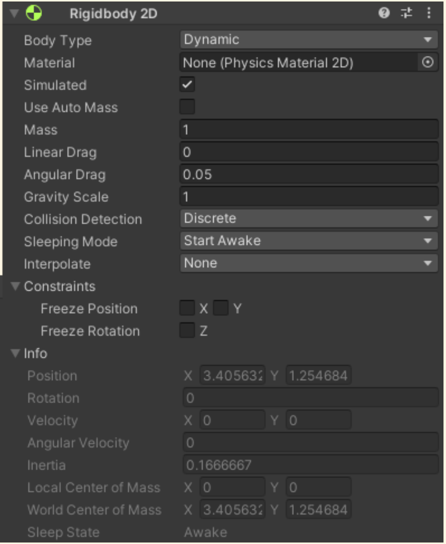
Physics in Unity

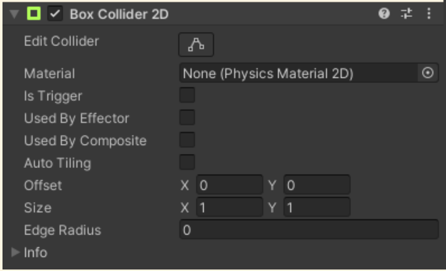
- Integrated Physics Engine
- Many Options
  - Unity Physics 2D
  - Unity Physics 3D
  - DOTS Physics
  - Havok Physics
- Future DOTS Integration



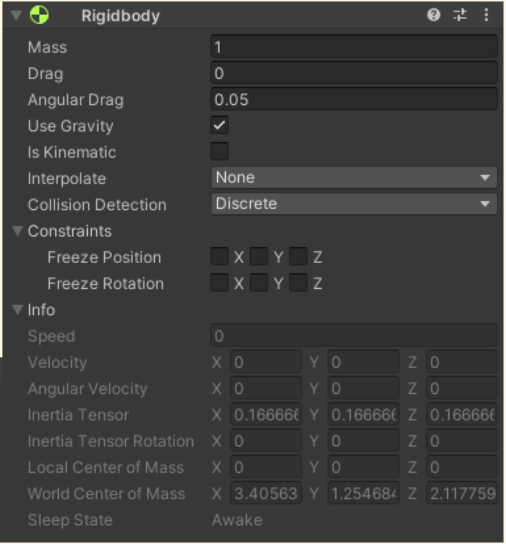
Source: Unity Documentation: Execution Order

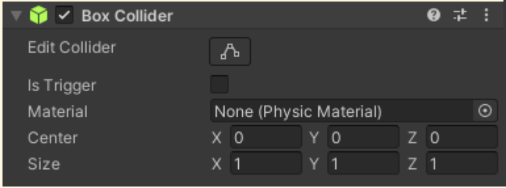
- Default 2D Solution
- RigidBody2D
- Collider2D
  - BoxCollider2D
  - CircleCollider2D
  - PolygonCollider2D
  - EdgeCollider2D



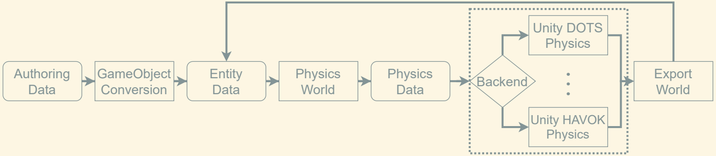


- Default 3D Solution
- RigidBody
- Collider
  - BoxCollider
  - SphereCollider
  - CapsuleCollider
  - MeshCollider

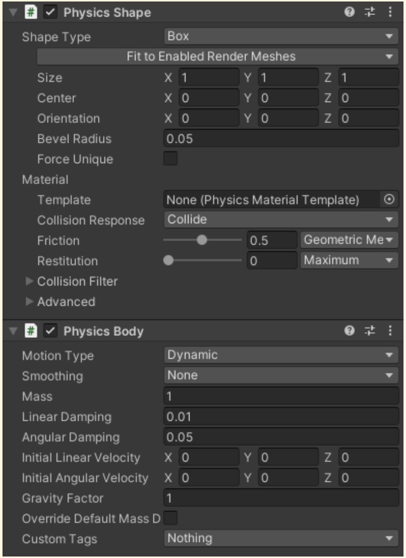




- New Approach
- Using DOTS → ECS
- Physics Body Authoring
- Physics Shape Authoring
- DOTS Physics Overview
- Custom Simulators

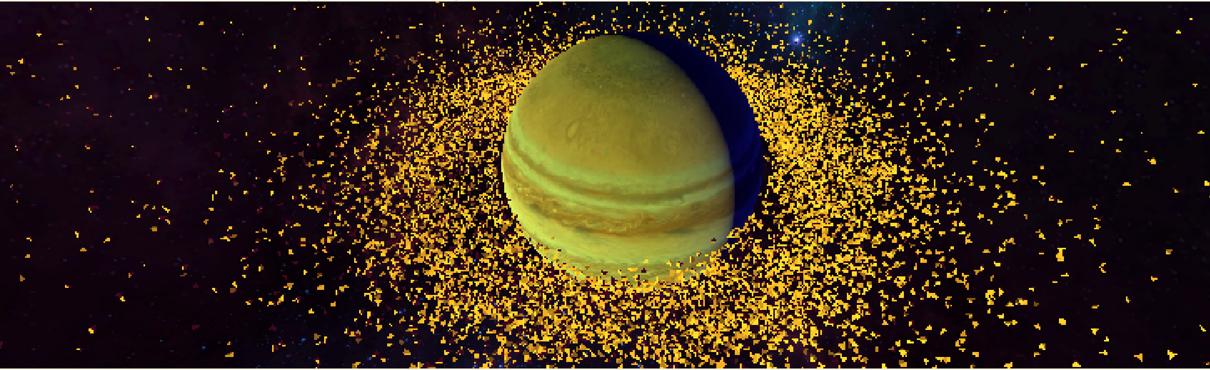


Source: Overview of Unity Physics



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-seidelsolver 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.