Voice from Communities

- Will MetaParser be open-source?
- Will we keep updating Wiki?
- Could Wang Xi make a video from professional's perspective to explain the bugs in the hottest games?
- We will have a voting campaign for the naming of Mini Engine later this week. The name of the Mini Engine will be decided by our community!

Pilot Engine V0.0.5 Released - 24 May

PILOT
Game engine

New Feature

- FXAA

Jiang Dunchun
jiangdunchun

Refactoring

- Framework
  - replaced singleton by global context
  - component system architecture
- Rendering
  - swap data context
  - RHI, RenderScene, RenderResource, RenderPipeline
  - Separated Vulkan-related logic
  - Decoupled editor UI and render logic
- Editor
  - Separated UI and Input layer
  - Mouse events (selecting, selection axis, camera speed adjusting)
  - Keyboard events (camera moving, deleting)
  - Switching between Editor Mode and Game Mode

Optimizations

- Added compile database to optimize development environment

Contributors

hyv1001, boooooommmmmm, and 9 other contributors

Lecture 10

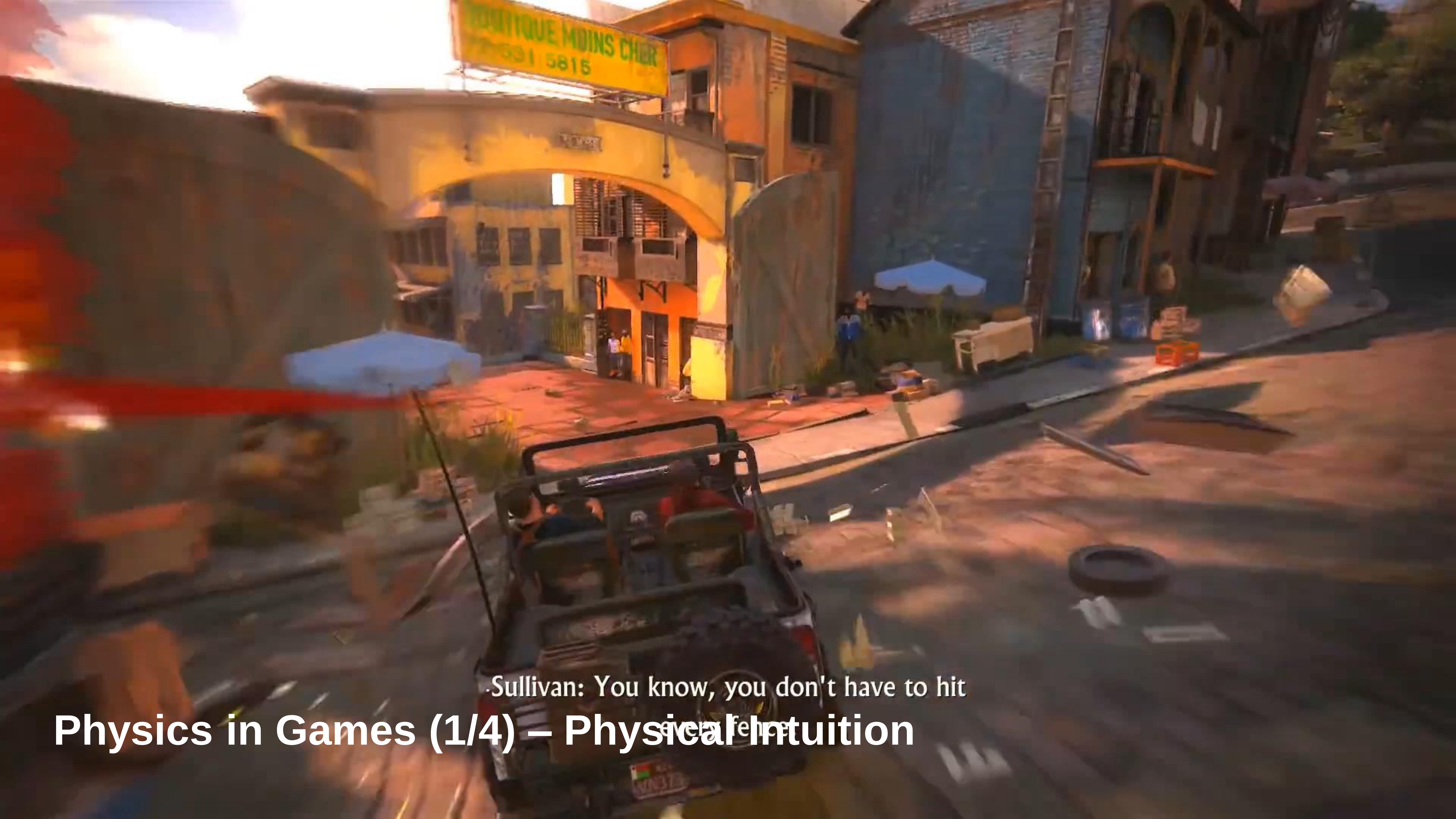
Physics System

Basic Concepts

WANG XI

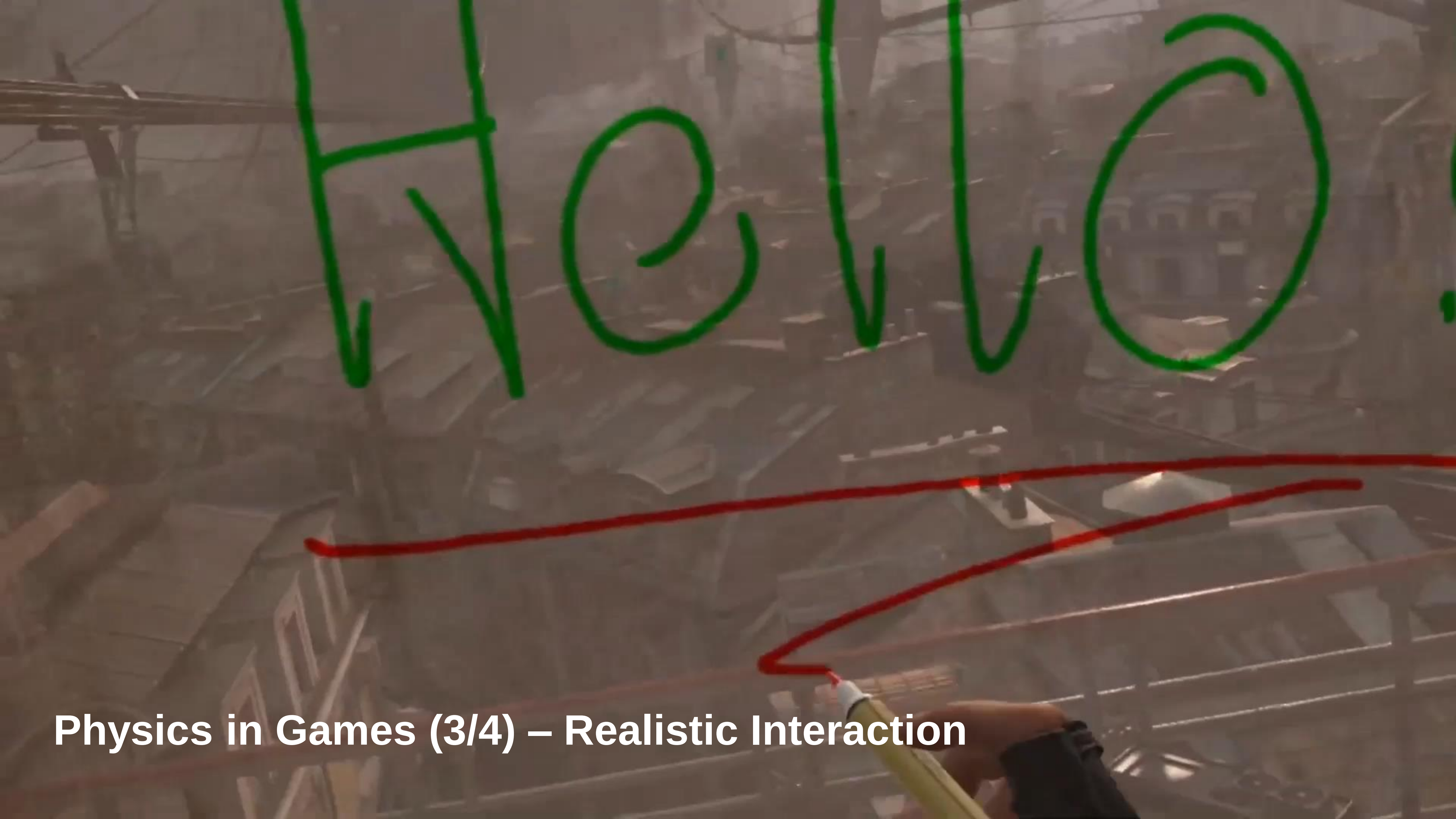
GAMES 104

2022



Physics in Games (1/4) – Physical Intuition

Physics in Games (2/4) – Dynamic Environment



Physics in Games (3/4) – Realistic Interaction

Physics in Games (4/4) – Artistic

Outline of Physics System

01.

Basic Concepts

- Physics Actors and Shapes
- Forces
- Movements
- Rigid Body Dynamics
- Collision Detection
- Collision Resolution
- Scene Query
- Efficiency, Accuracy, and Determinism

02.

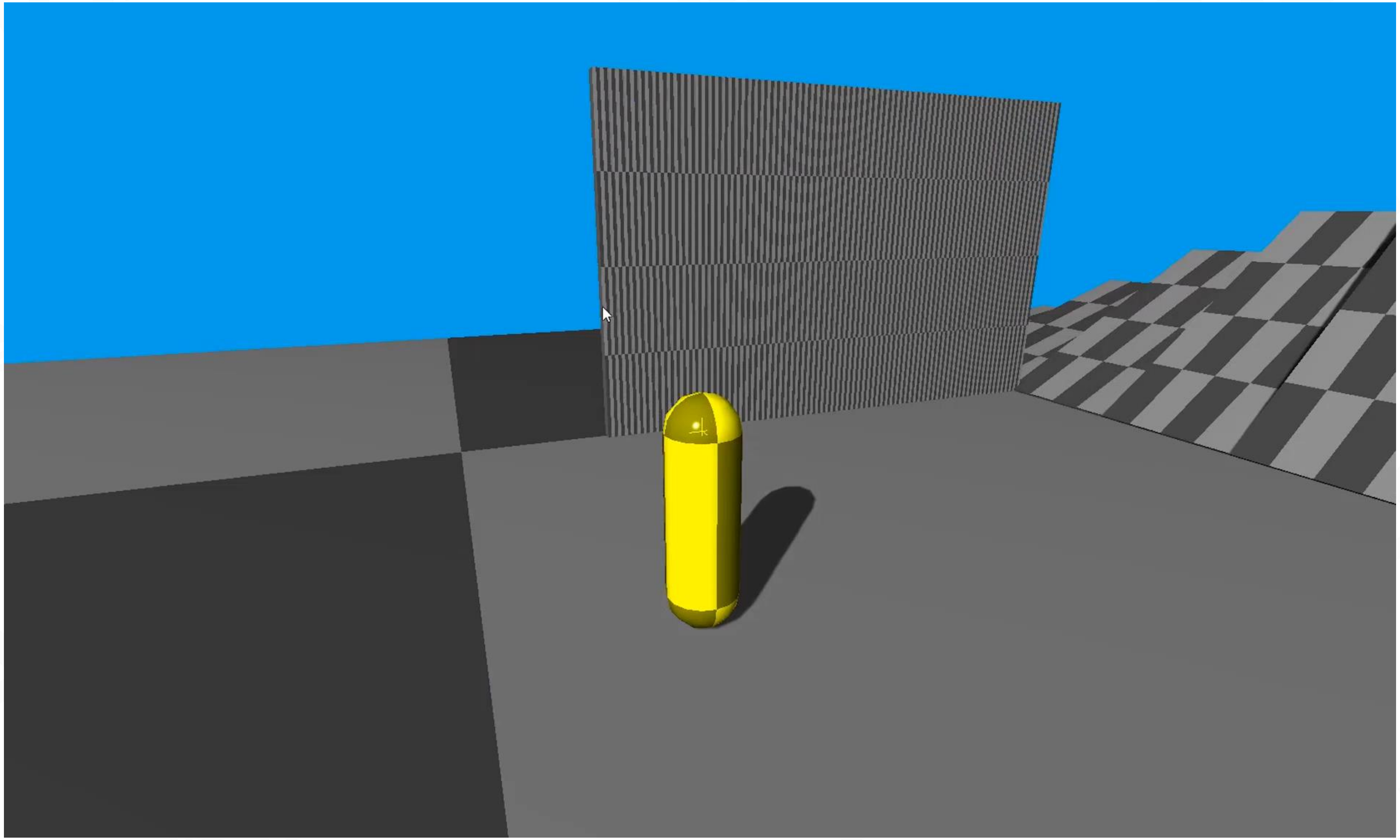
Applications

- Character Controller
- Ragdoll
- Destruction
- Cloth
- Vehicle
- Advanced Physics : PBD

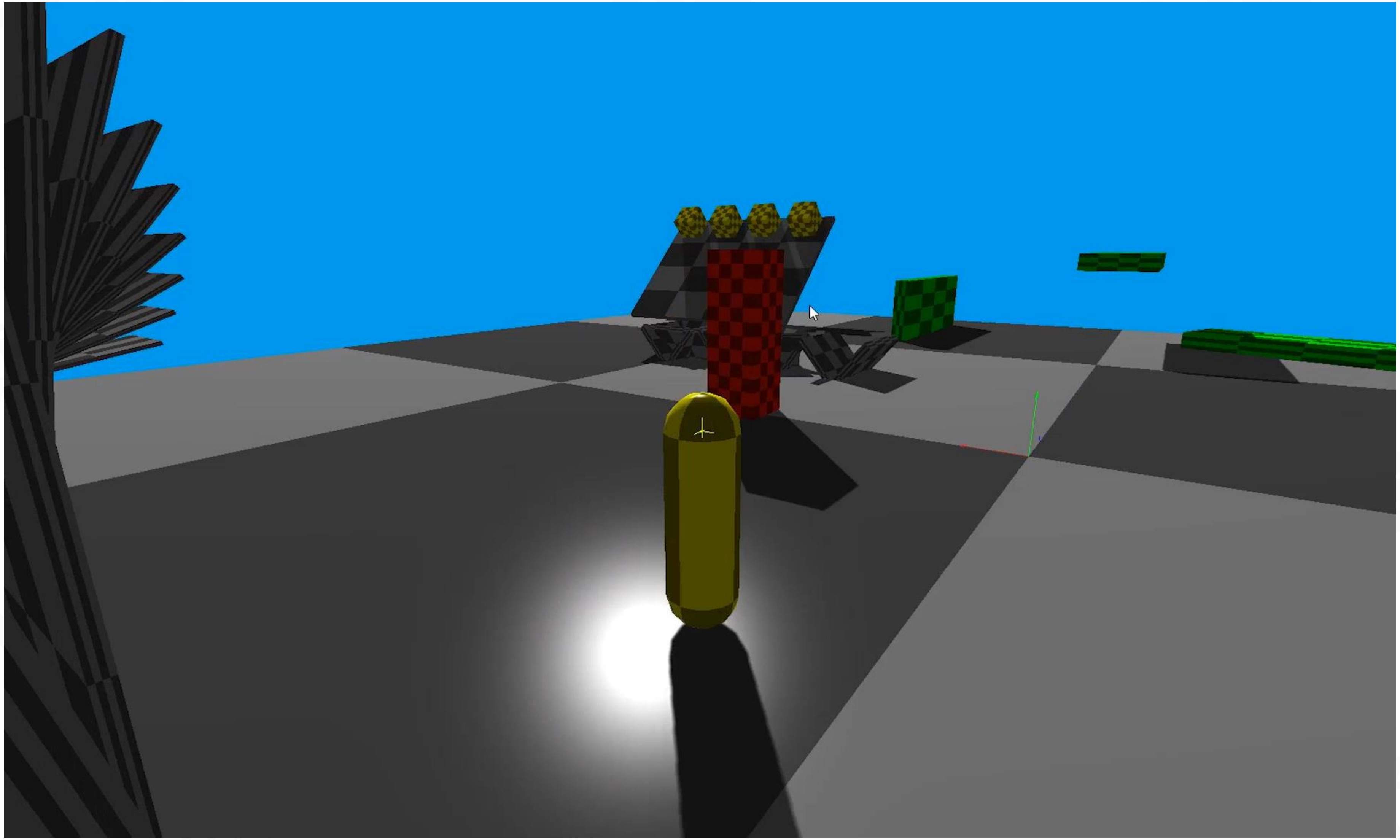
Physics Actors and Shapes

Visual World vs. Physics World

Actor – Static



Actor – Dynamic



Trigger

- Like static actor, not moving
- But not blocking
- Notifies when actors enter or exit

Physics Law is Unbreakable, But in Game…

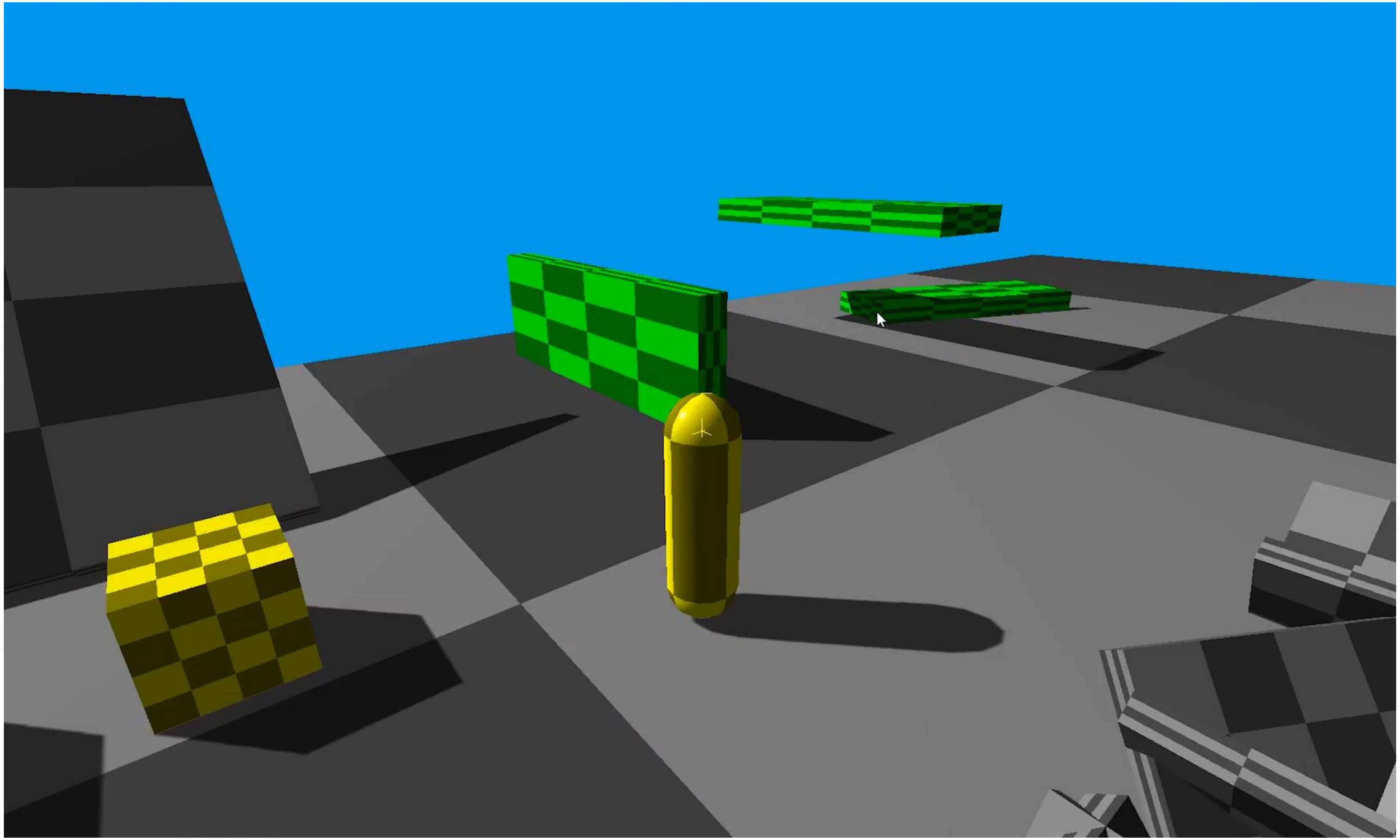
Elon Musk @elonmusk · 27 Dec 2021

Replying to @PPathole

People are able to break any laws made by humans, but none made by physics

1,012 1,330 11.3K

Actor – Kinematic (No Physics Law)



Kinematic Actors are Troublemakers

Actor – Summary

Static Actor

- Not moving

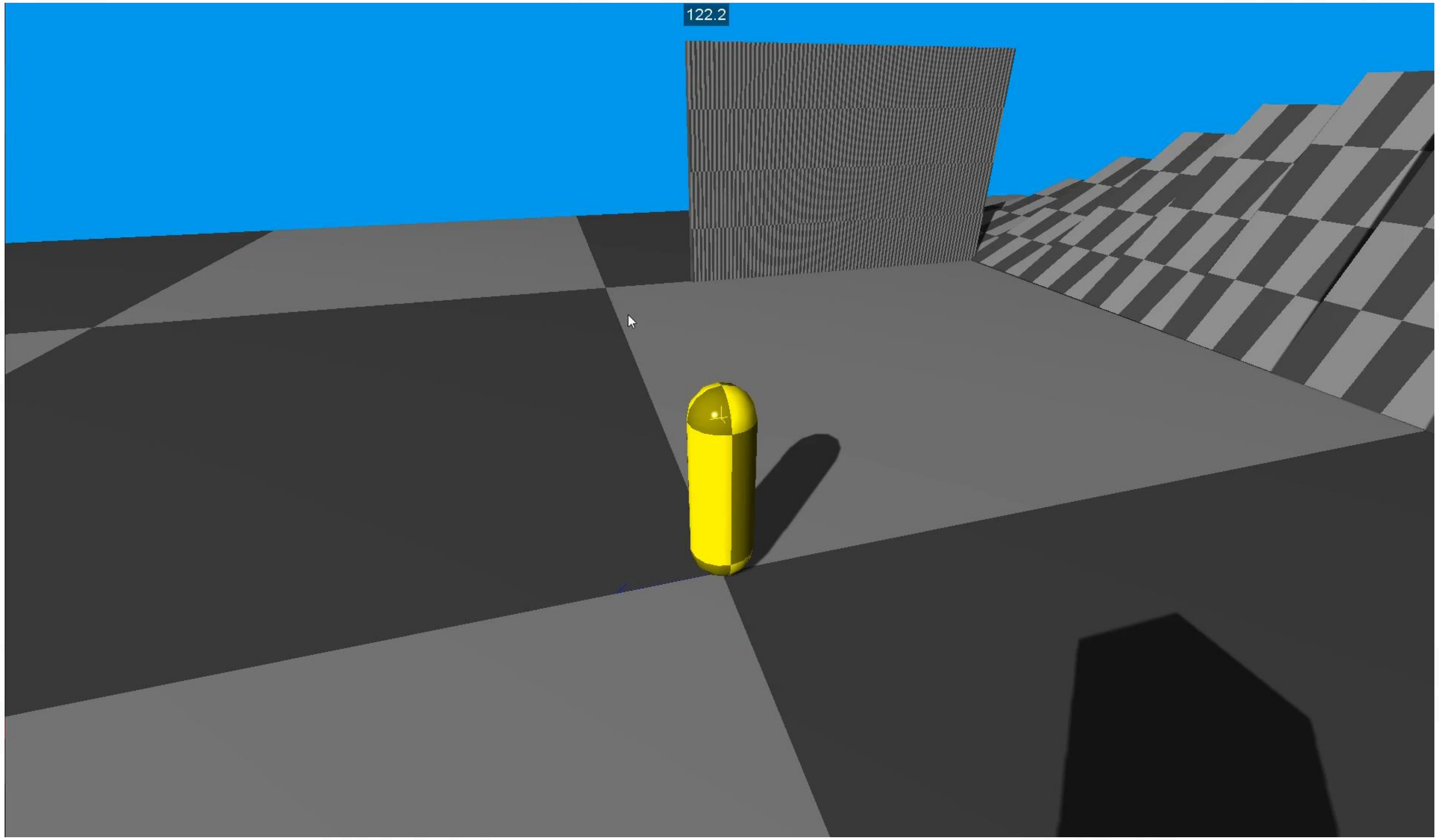
Dynamic Actor

- Can be affected by forces/torques/impulses

Trigger

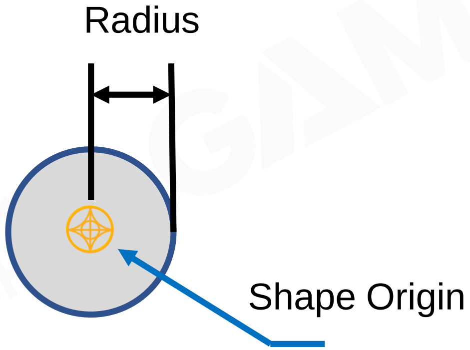
Kinematic Actor

- Ignoring physics rules
- Controlled by gameplay logic directly

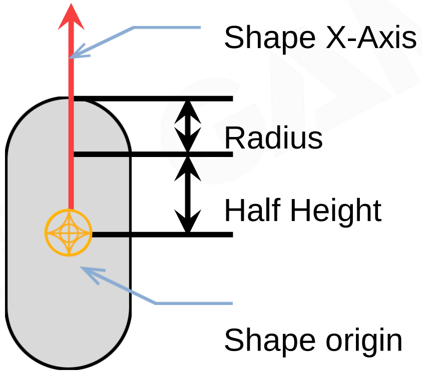


Actor Shapes

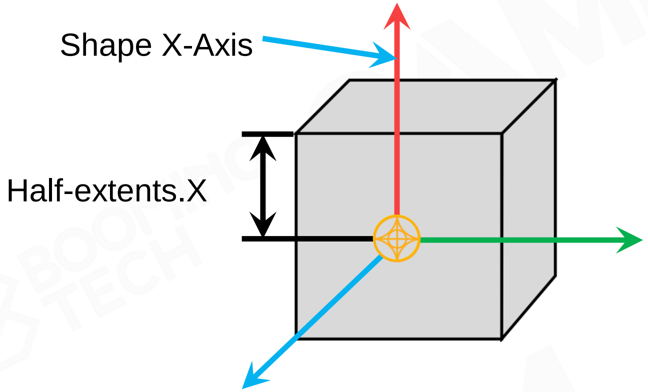
Spheres



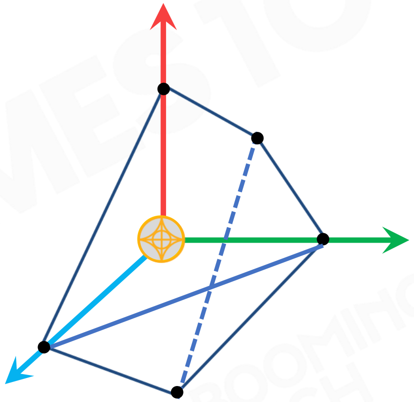
Capsules



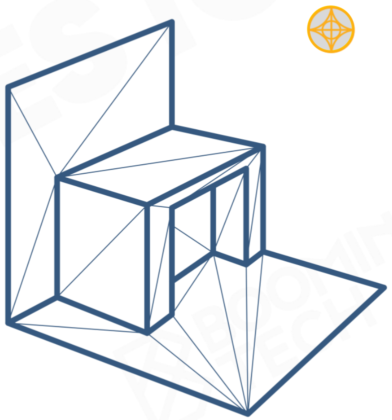
Boxes



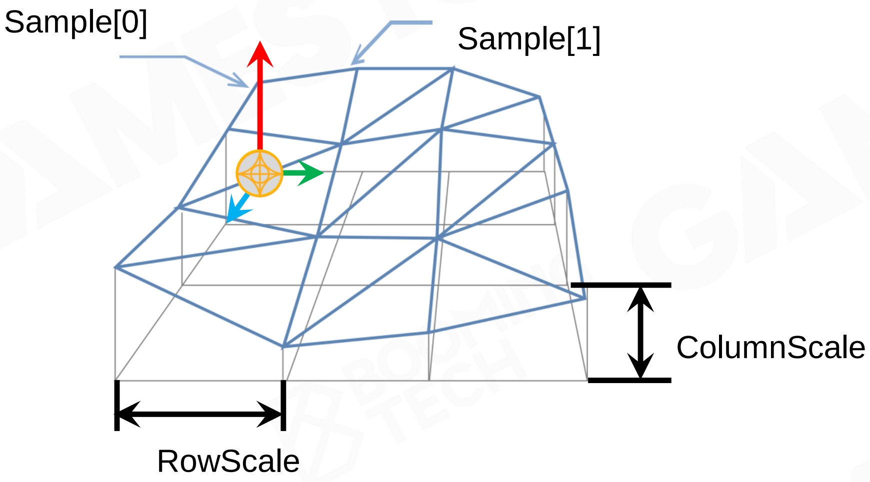
Convex Meshes



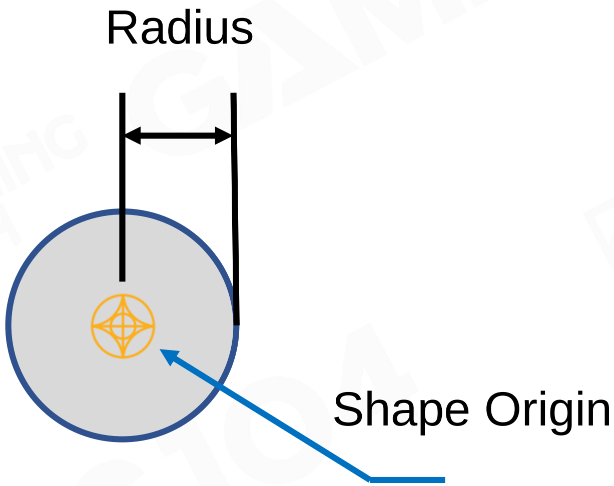
Triangle Meshes

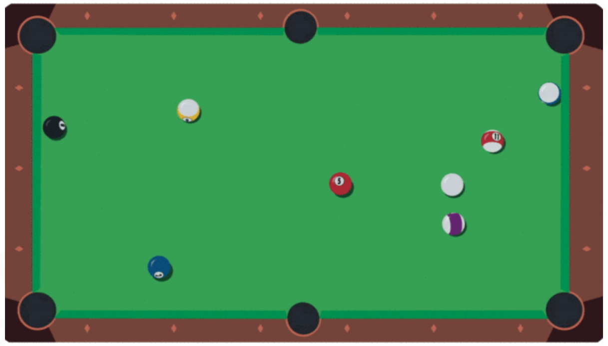


Height Fields

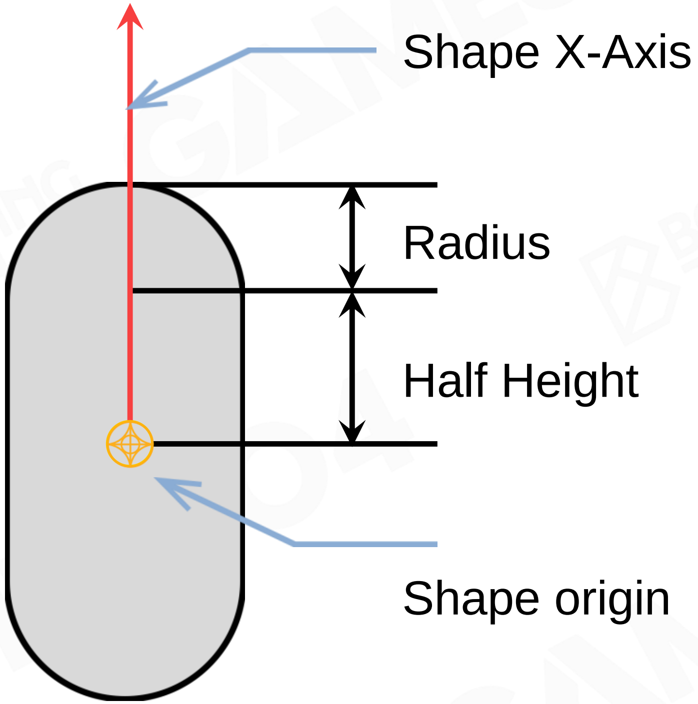


Shapes – Spheres

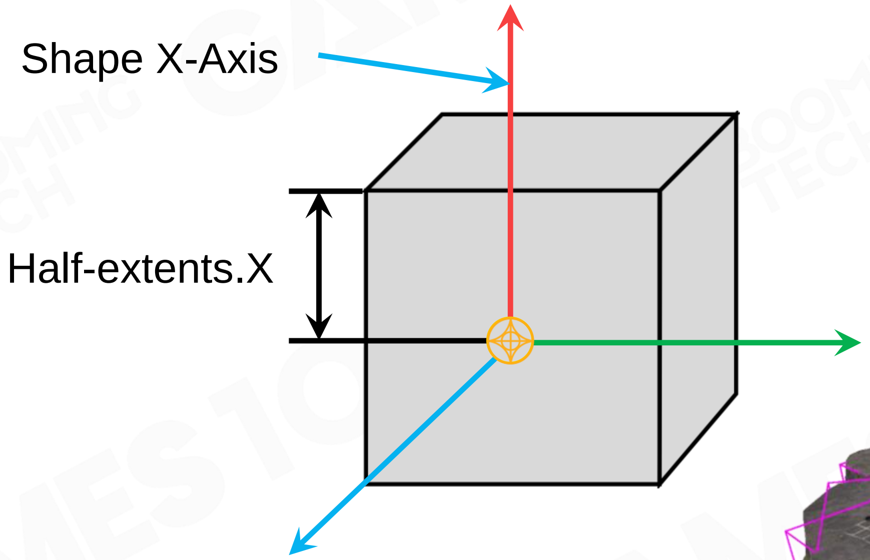




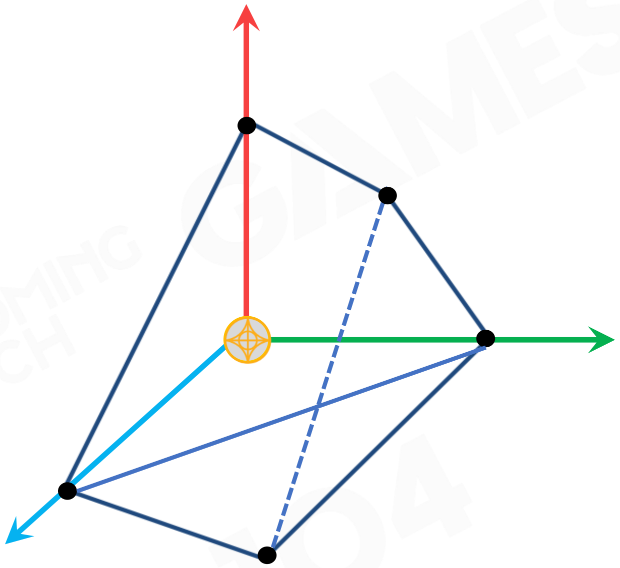
Shapes – Capsules



Shapes – Boxes

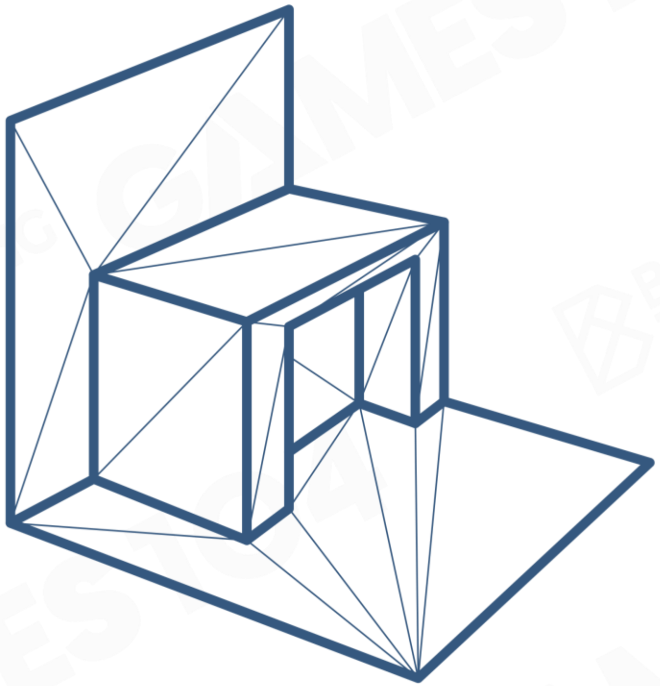


Shapes – Convex Meshes



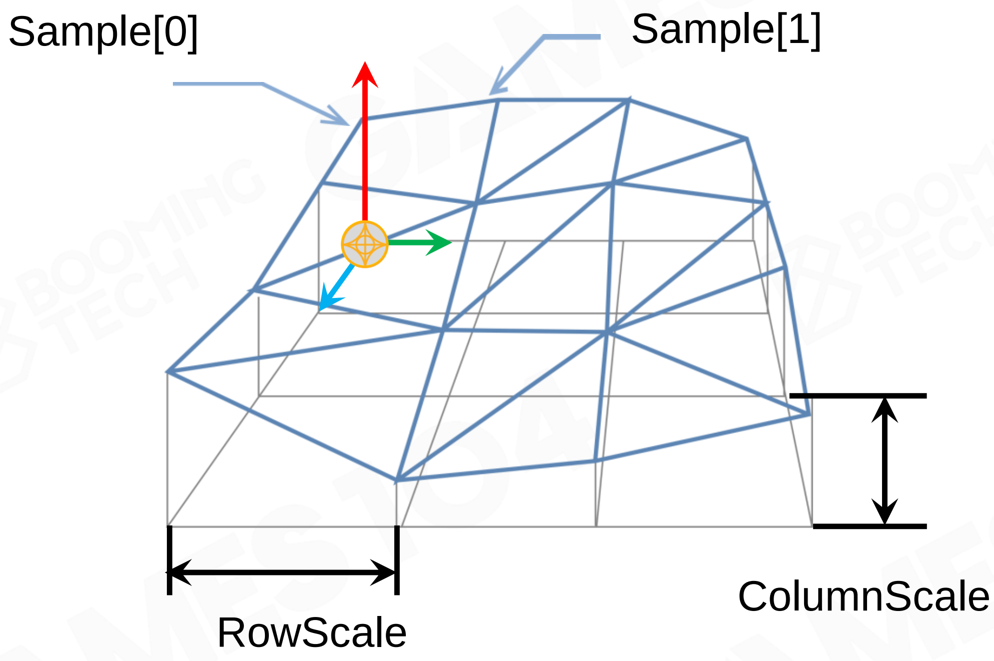
Vertices and faces limits of convex meshes

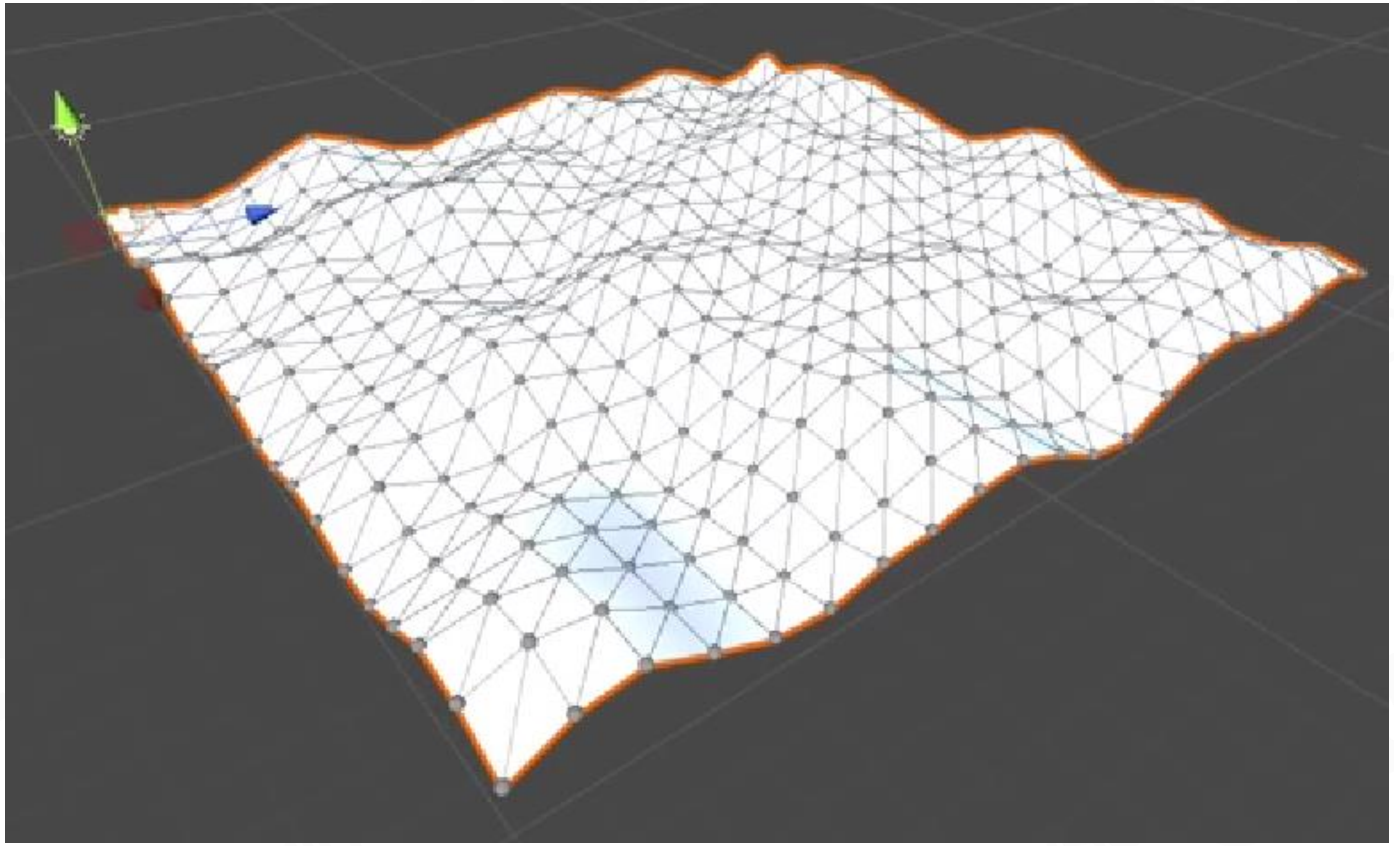
Shapes – Triangle Meshes



- Dynamic actors can't have triangle meshes

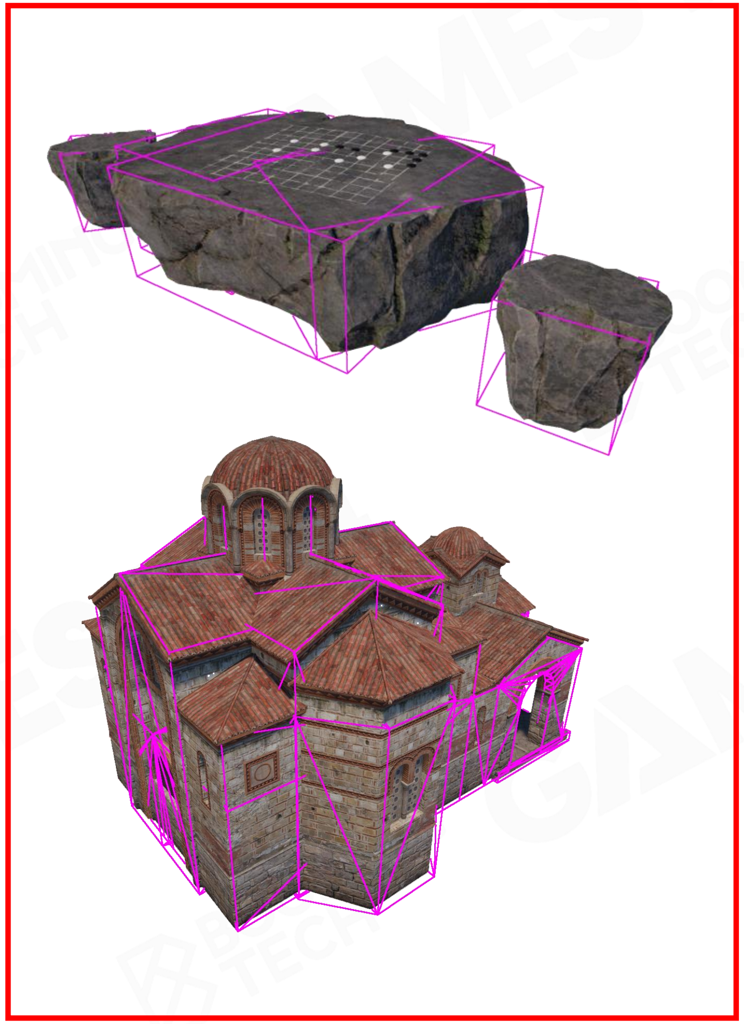
Shapes – Height Fields





Wrap Objects with Physics Shapes

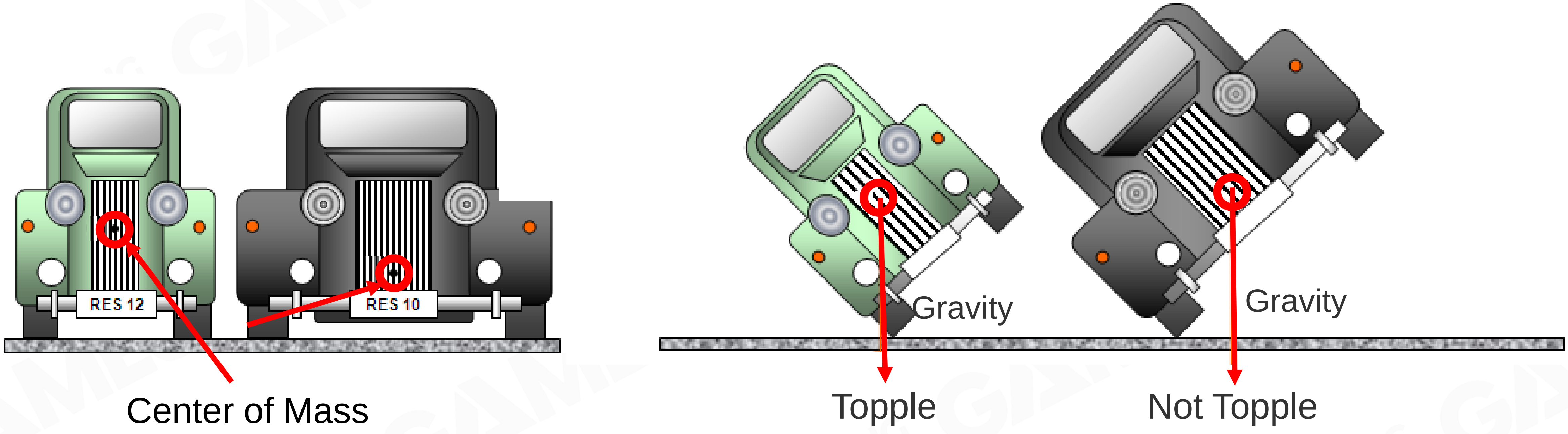
- Approximated Wrapping
  - Don’t need to be perfect
- Simplicity
  - Prefer simple shapes (avoid triangle mesh if possible)
  - Least shapes



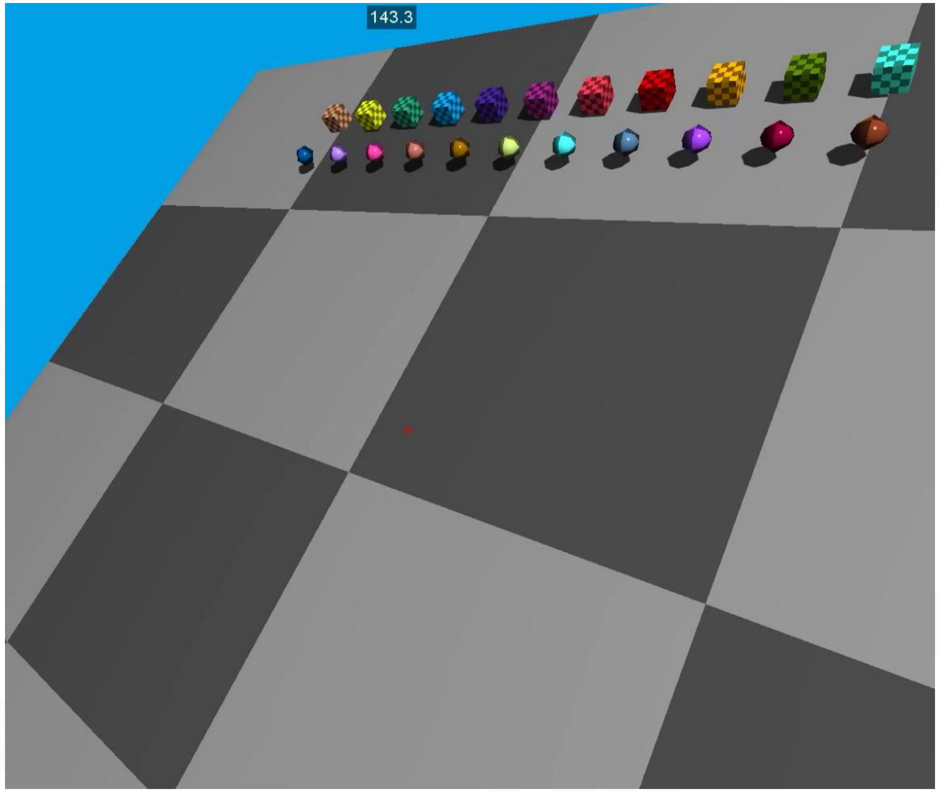
Shape Properties – Mass and Density

Gomboc Shape

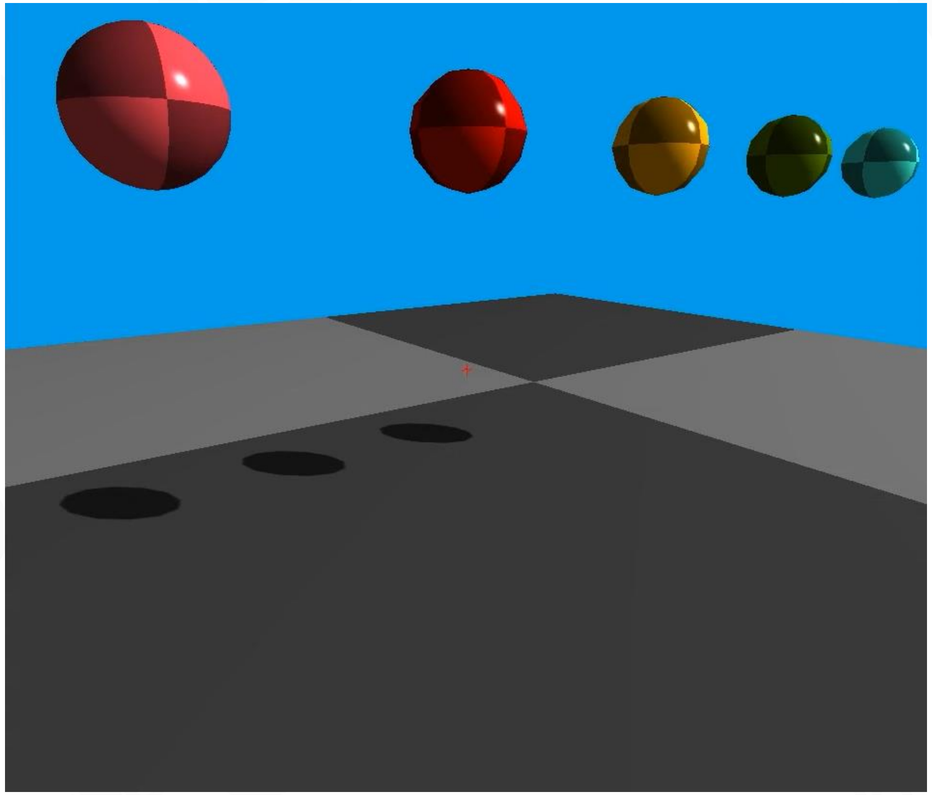
Shape Properties - Center of Mass



Shape Properties – Friction & Restitution



Different Friction Parameters



Different Restitution Parameters

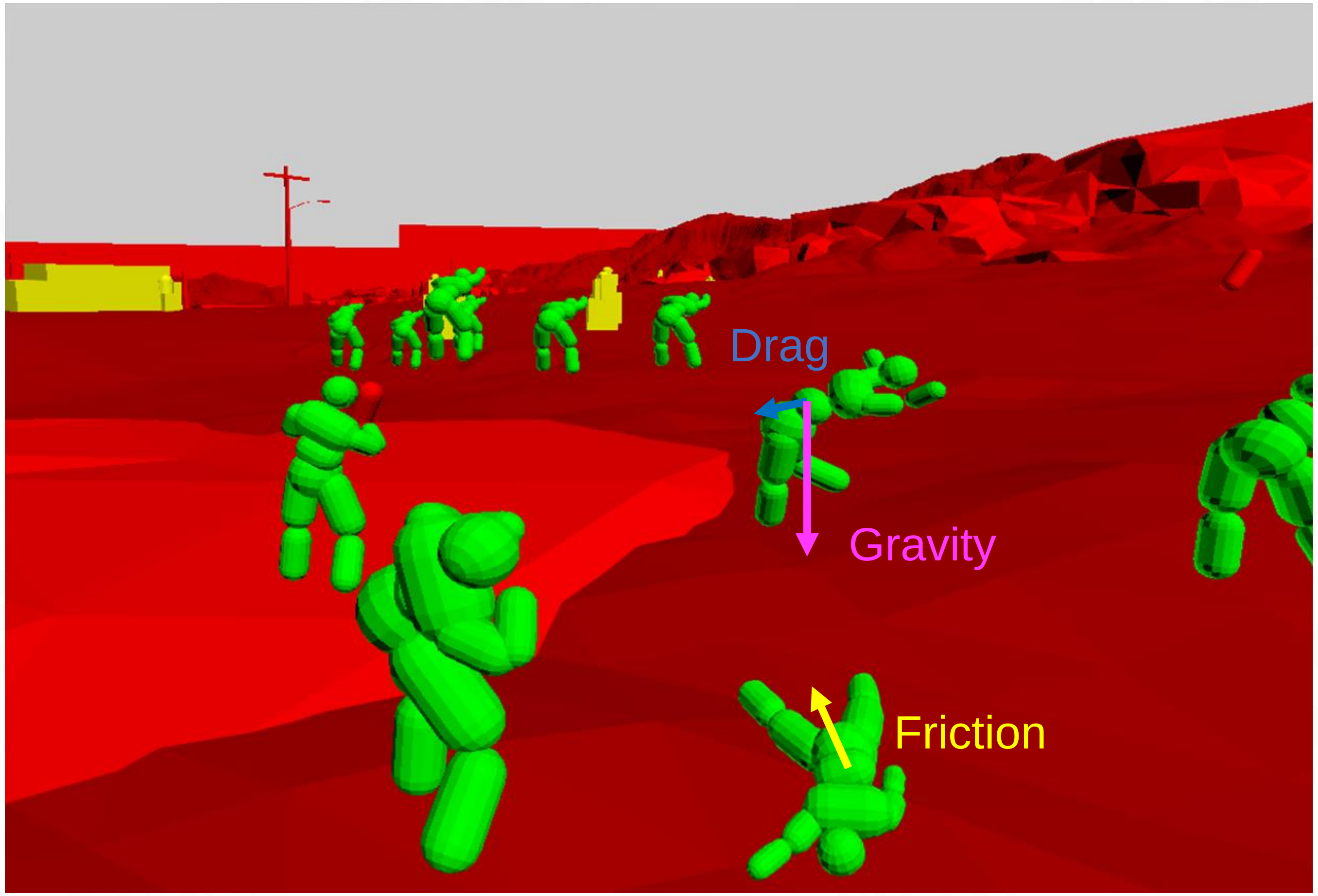
Forces

Force

- We can apply forces to give dynamic objects accelerations, therefore affecting their movements
- Examples
  - Gravity
  - Drag
  - Friction
  - …

Force

- We can apply forces to give dynamic objects accelerations, therefore affecting their movements
- Examples
  - Gravity
  - Drag
  - Friction
  - …

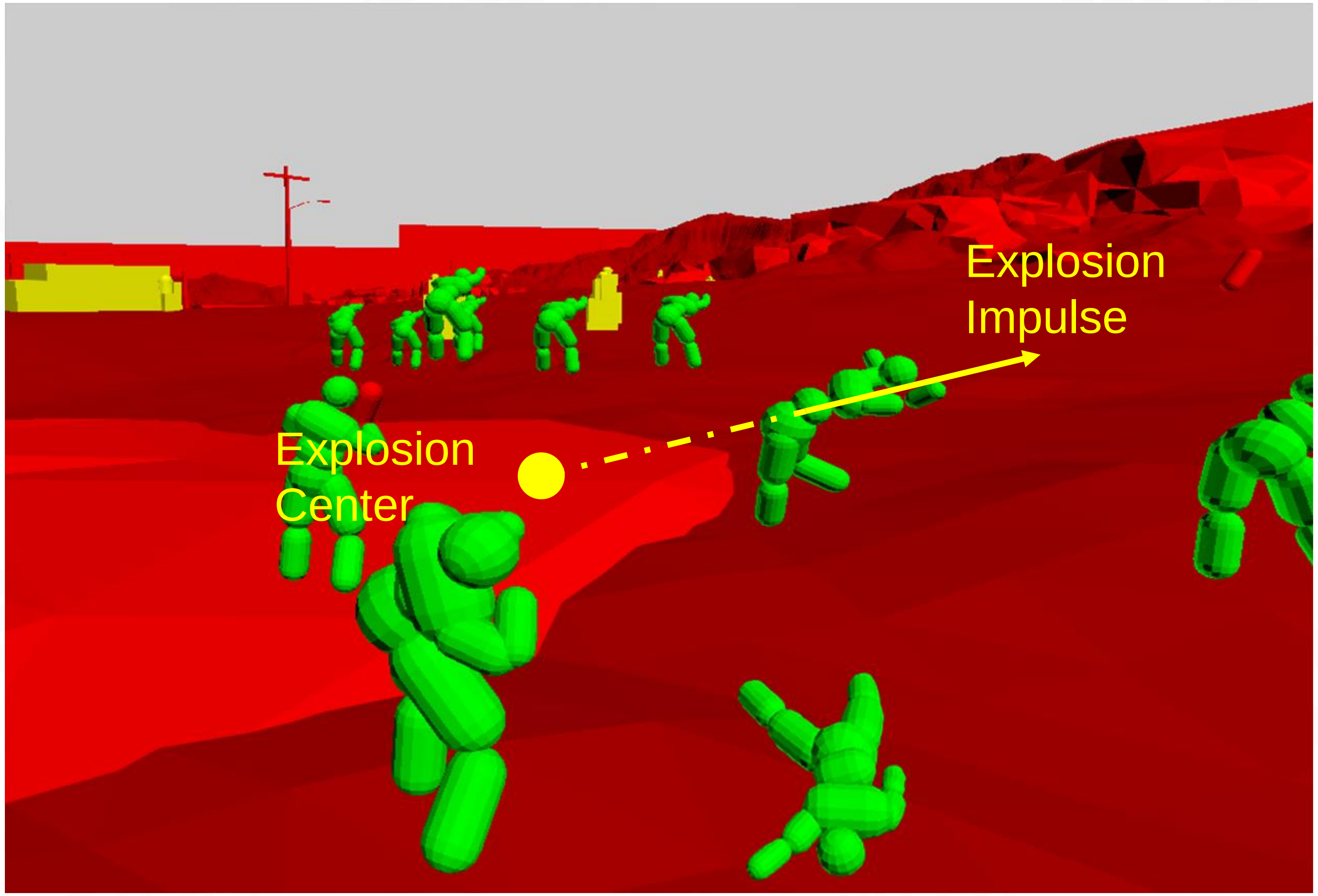


Impulse

- We can change velocity of actors immediately by applying impulses
- E.g. simulating an explosion

Impulse

- We can change velocity of actors immediately by applying impulses
- E.g. simulating an explosion



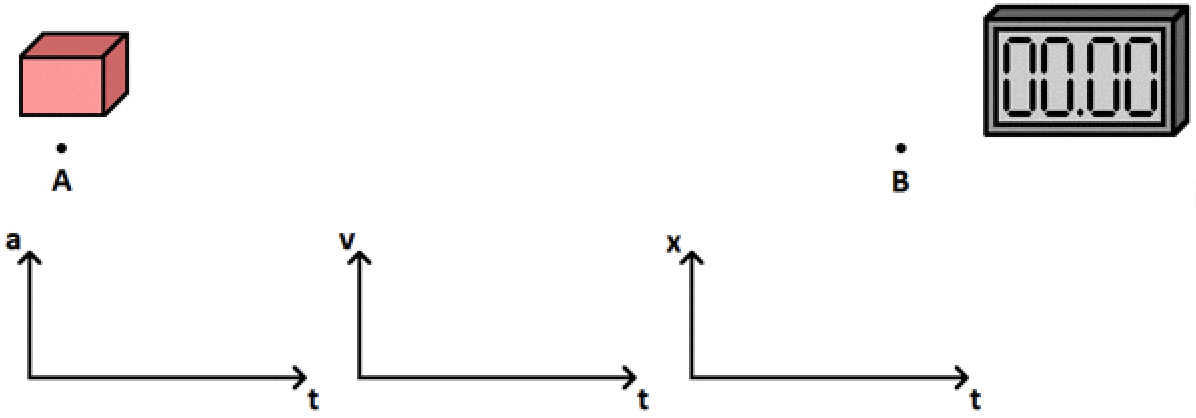
Movements

Newton's 1st Law of Motion

If there is no external force

$$\vec{v}(t+\Delta t)=\vec{v}(t)$$

$$\vec{x}(t+\Delta t)=\vec{x}(t)+\vec{v}(t)\Delta t$$



gifexperiments.blogspot.com

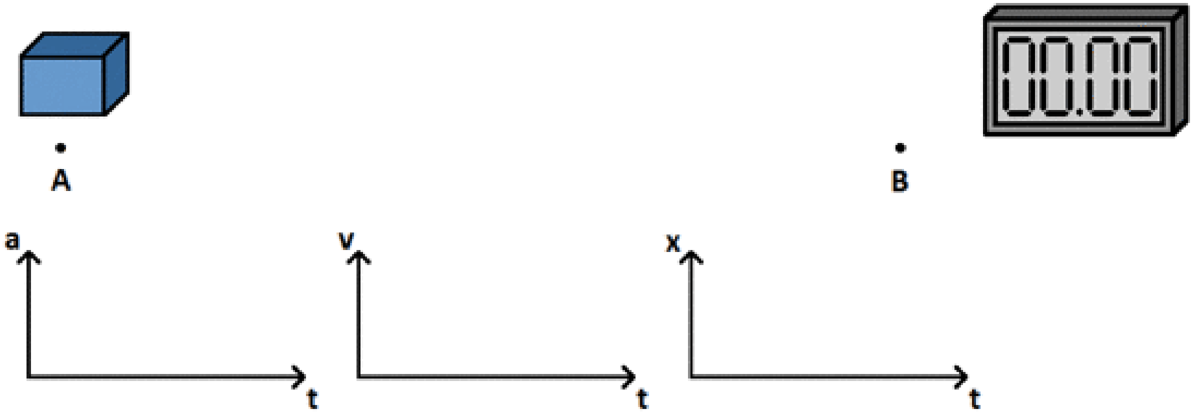
Newton's 2st Law of Motion

If there is external force

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

Force Mass Acceleration

$$\vec{a}(t) = \frac{\mathrm{d}\vec{v}(t)}{\mathrm{d}t} = \frac{\mathrm{d}^2\vec{x}(t)}{\mathrm{d}t^2}$$



gifexperiments.blogspot.com

Movement under Constant Force

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

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t$$
$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2$$

Movement under Varying Force

Newton's 2st Law of Motion

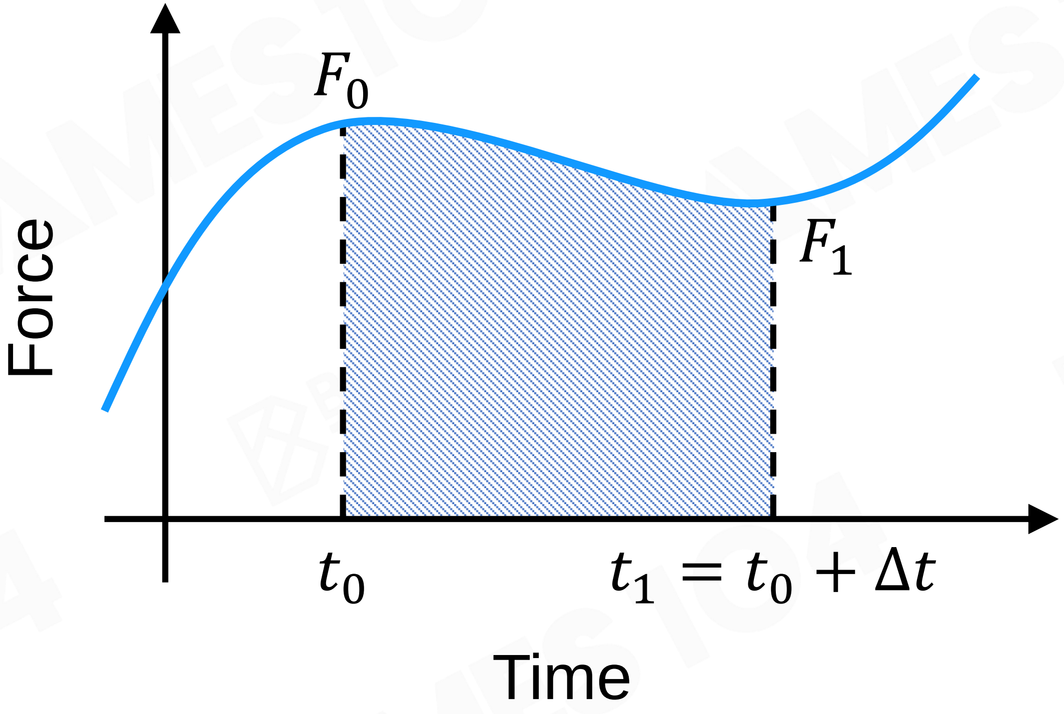
If there is *varying* external force

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

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

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \ ?$$

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \ ?$$



Movement under Varying Force

Newton's 2st Law of Motion

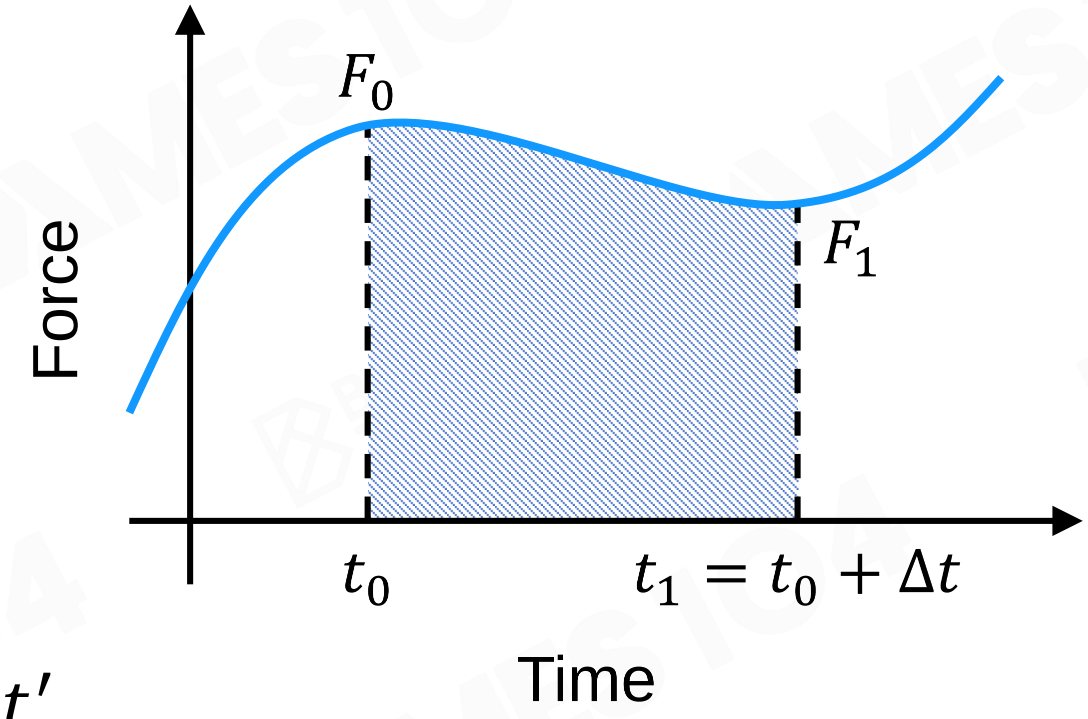
If there is *varying* external force

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

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

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \int_{t}^{t+\Delta t} \vec{a}(t')\mathrm{d}t'$$

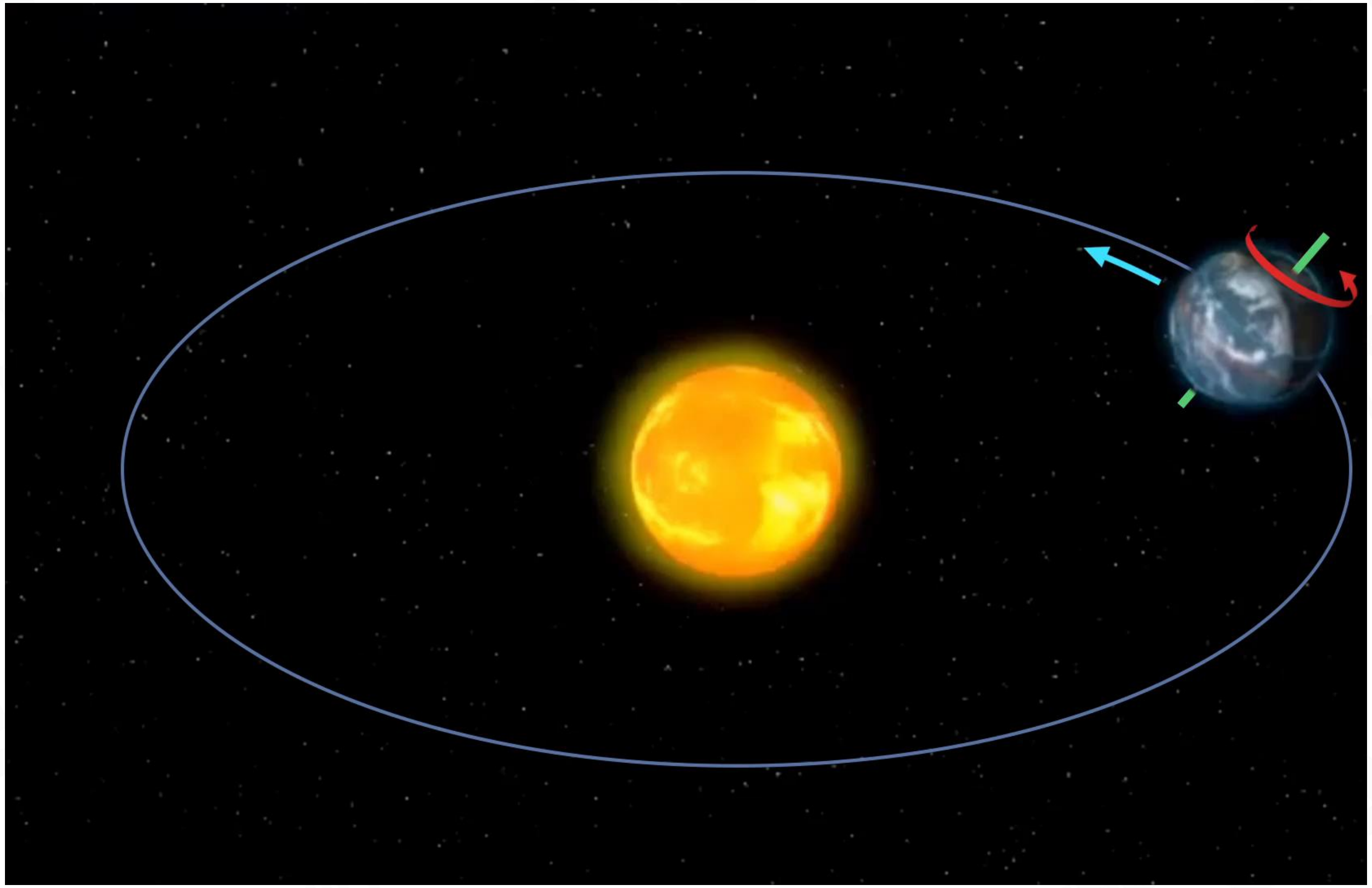
$$\vec{x}(t+\Delta t) = \vec{x}(t) + \int_{t}^{t+\Delta t} \vec{v}(t')\mathrm{d}t'$$



Example of Simple Movement

- Position
- Orientation
- Linear Velocity
- Angular Velocity

$$\mathbf{X}(t) = \begin{pmatrix} \vec{x}(t) \\ R(t) \\ \vec{v}(t) \\ \vec{\omega}(t) \end{pmatrix}$$

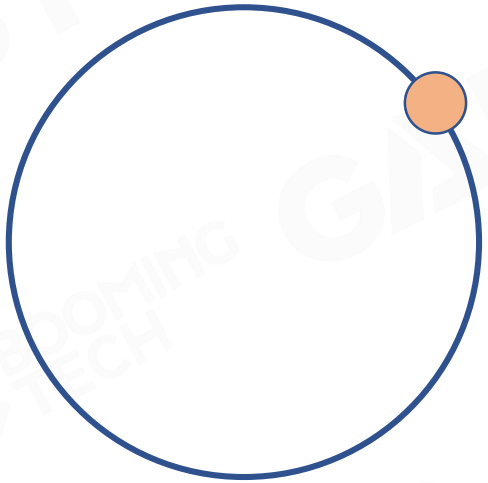


Earth In The Solar System

Motion in Reality

At time t

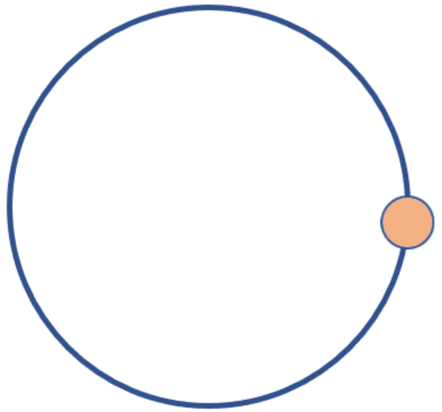
- Position: $\vec{x}(t)$
- Linear Velocity: $\vec{v}(t) = \frac{\mathrm{d}\vec{x}(t)}{\mathrm{d}t}$



Simulation in Game

At time t

- Position: $\vec{x}(t)$
- Linear Velocity: $\vec{v}(t) = \frac{\mathrm{d}\vec{x}(t)}{\mathrm{d}t}$

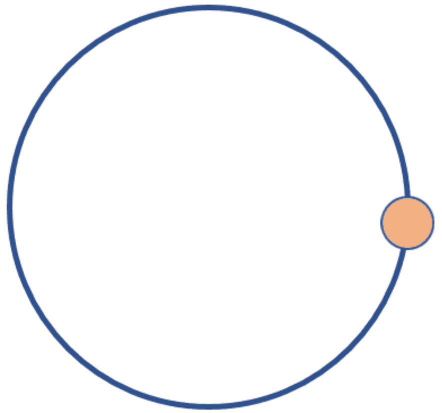


Simulation Step

Given $\vec{x}(t)$, $\vec{v}(t)$

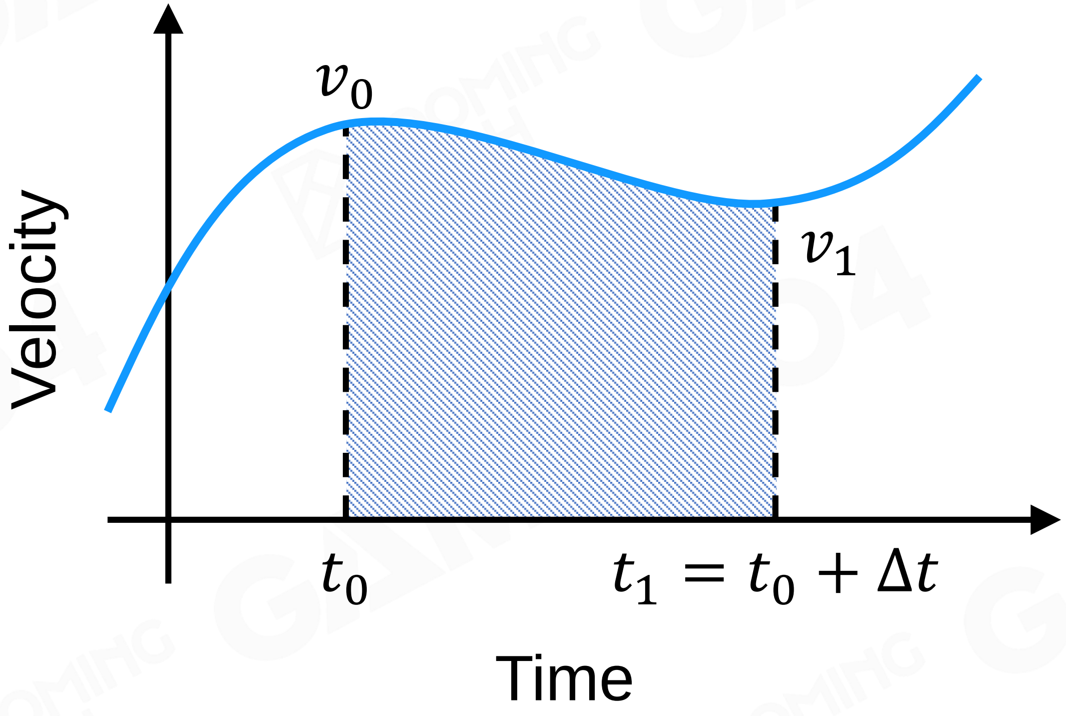
Compute $\vec{x}(t + \Delta t)$, $\vec{v}(t + \Delta t)$

Δt is the time step size



Time Integration

$$\vec{x}(t_1) = \vec{x}(t_0) + \int_{t_0}^{t_1} \vec{v}(t)\mathrm{d}t$$



Euler’s Method

200 CAPVT VII.

METHODVS GENERALIS
INTEGRALIA QVAECVNQVE PROXIME
INVENIENDI.

Problema 36.

297.

Formulae integralis cuiuscunque $y = \int X dx$ valorem vero proxime indagare.

Solutio.

Cum omnis formula integralis per ſe ſit indeterminata, ea ſemper ita determinari ſolet, vt ſi variabili x certus quidam valor puta a tribuatur, ipſum integrale $y = \int X dx$ datum valorem puta b obtineat. Integratione igitur hoc modo determinata, quaeſtio huc redit, ſi variabili x alius quicunque valor ab a diuerſus tribuatur, valor, quem tum integrale y ſit habiturum, definiatur. Tribuamus ergo ipſi x primo valorem parum ab a diſcrepantem, puta $x = a + \alpha$, vt α ſit quantitas valde parva: et quia functio X parum variatur, ſiue pro x ſcribatur a ſiue $a + \alpha$ eam tanquam conſtantem ſpectare licebit. Hinc ergo formulae differentialis $X dx$ integrale

CAPVT VII. 201

ntegrale erit $Xx + \text{Conſt.} = y$; ſed quia poſito $x = a$ fieri debet $y = b$, et valor ipſius X quaſi nanet immutatus, erit $Xa + \text{Conſt.} = b$, ideoque Conſt. $= b - Xa$, vnde conſequimur $y = b + X(x - a)$. Quare ſi ipſi x valorem $a + \alpha$ tribuamus, habebinus valorem conuenientem ipſius y, qui ſit $= b + \beta$; c iam ſimili modo ex hoc caſu definire poterinus y, ſi ipſi x tribuatur alius valor parum ſuerans $a + \alpha$, poſito igitur $a + \alpha$ loco x, valor pſius X inde ortus denuo pro conſtante haberi poerit, indeque fiet $y = b + \beta + X(x - a - \alpha)$. Hanc gitur operationem continuare licet quonsque lubueit, cuius ratio quo melius perſpiciatur, rem ita epraeſentemus:

$x = a$ fiat $X = A$ et $y = b$

$x = a'$ .. $X = A'$ .. $y = b' = b + A(a' - a)$

$x = a''$ .. $X = A''$ .. $y = b'' = b' + A'(a'' - a')$

$x = a'''$ .. $X = A'''$ .. $y = b''' = b'' + A''(a''' - a'')$

etc.

bi valores a, a', a'', a''' etc. ſecundum differenas valde paruas procedere ponuntur. Erit ergo $= b + A(a' - a)$ quippe in quam abit formula muenta $y = b + X(x - a)$ fit enim $X = A$, quia ponitur $x = a$, tum vero tribuitur ipſi x valor $= a'$; cui reſpondet $y = b'$, ſimili modo erit $b'' = A'(a'' - a')$; tum $b''' = b'' + A''(a''' - a'')$ etc. vti ſupra poſuimus.

Cc

CAPVT VII.

Reſtituendo ergo valores praecedentes habe-

$A'a' - a)$

$A(a' - a) + A'(a'' - a')$

$A(a' - a) + A'(a'' - a') + A''(a''' - a'')$

$A(a' - a) + A'(a'' - a') + A''(a''' - a'') + A(a'''' - a''')$

etc.

x quantumuis excedet a, ſeries a', a'', a''' etc. continuetur ad x, et vltimum aggregatum lorem ipſius y.

Coroll. 1.

98. Si incrementa, quibus x augetur, aeſtatuantur ſcilicet α, vt ſit $a' = a + \alpha$, $+ 2\alpha$, $a''' = a + 3\alpha$, etc. quibus valoribus ubſtitutis functio X abeat in A', A'', A''' etc. ltimus illorum valorum puta $a + n\alpha$ ſit $= x$ vero X, erit

$b + \alpha(A + A' + A'' + A''' \ldots\ldots + X)$.

Coroll. 2.

99. Valor ergo integralis y per ſummatioſeriei $A, A', A'' \ldots X$, cuius termini ...ula X formantur ponendo loco x ſucceſſiue $a, a + \alpha, a + 2\alpha \ldots\ldots a + n\alpha$, eruitur. Summa enim illius ſeriei per differentiam α multiplicata et ad b adiecta dabit valorem ipſius y, qui ipſi $x = a + n\alpha$ reſpondet.

Coroll. 3.

Institutiones calculi integralis (1768-70), p200-203.

Leonhard Euler

1707-1783

Explicit (Forward) Euler's Method (1/3)

Simplest estimation

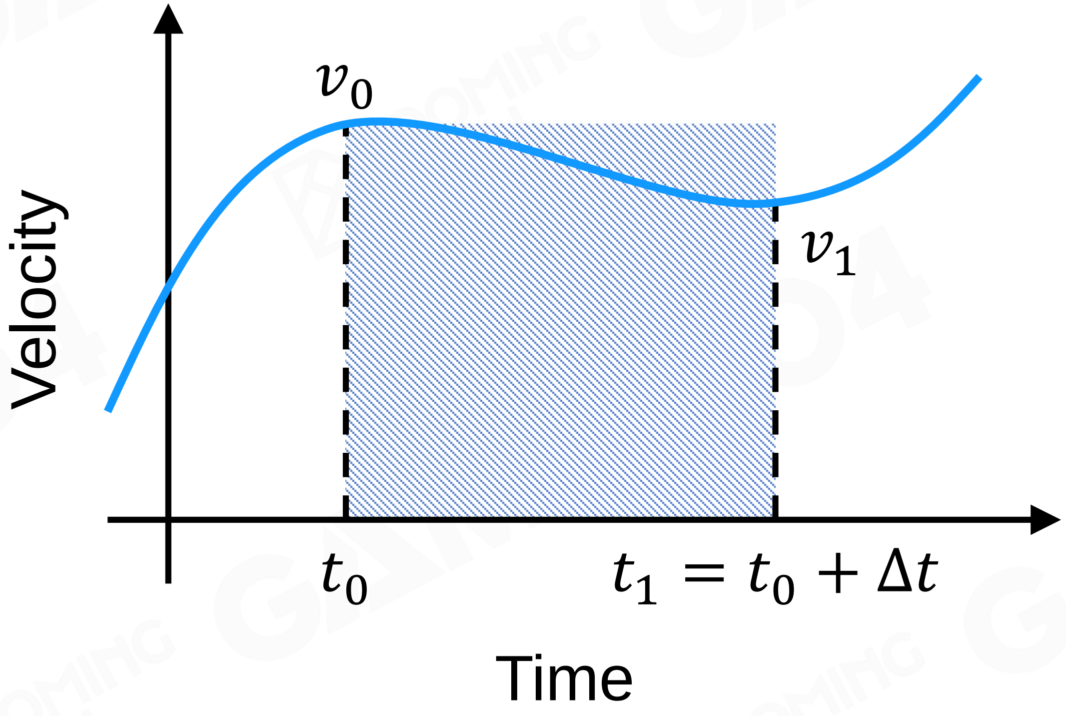
Assume the force is constant

during the time step

$$\begin{cases} \vec{v}(t_1) = \vec{v}(t_0) + M^{-1}\boxed{\vec{F}(t_0)}\Delta t \\ \vec{x}(t_1) = \vec{x}(t_0) + \boxed{\vec{v}(t_0)}\Delta t \end{cases}$$

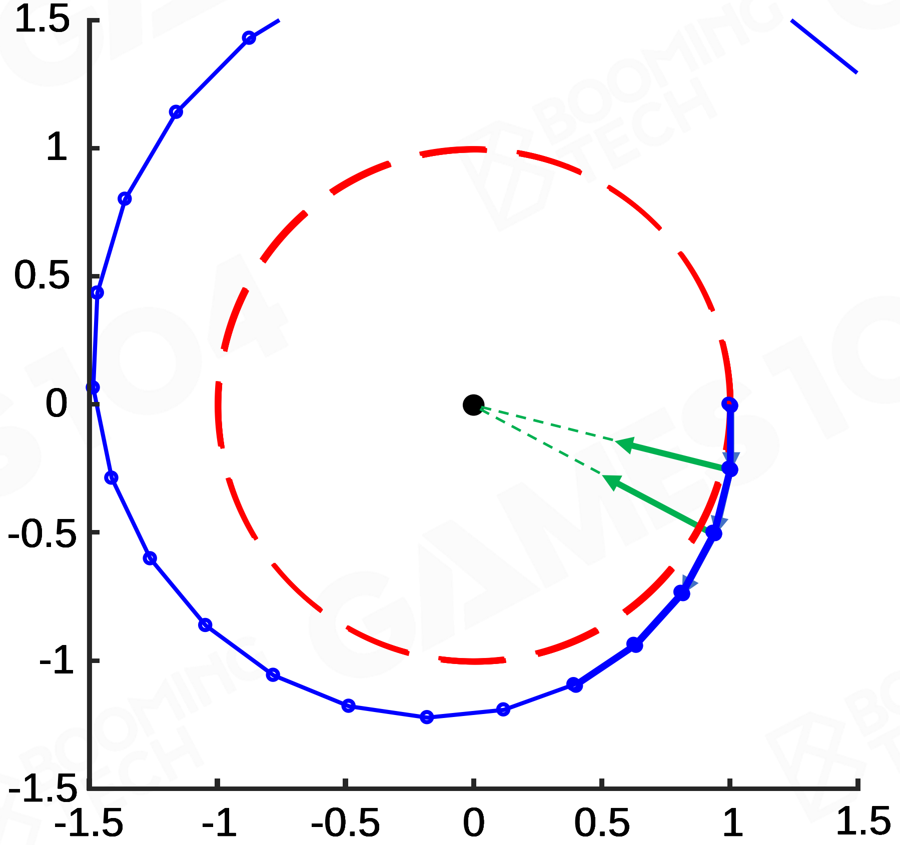
Current States

All quantities are known



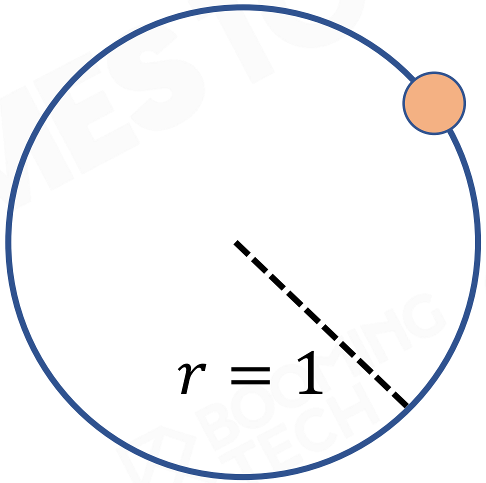
Explicit (Forward) Euler's Method (2/3)

$\Delta t = 0.25$



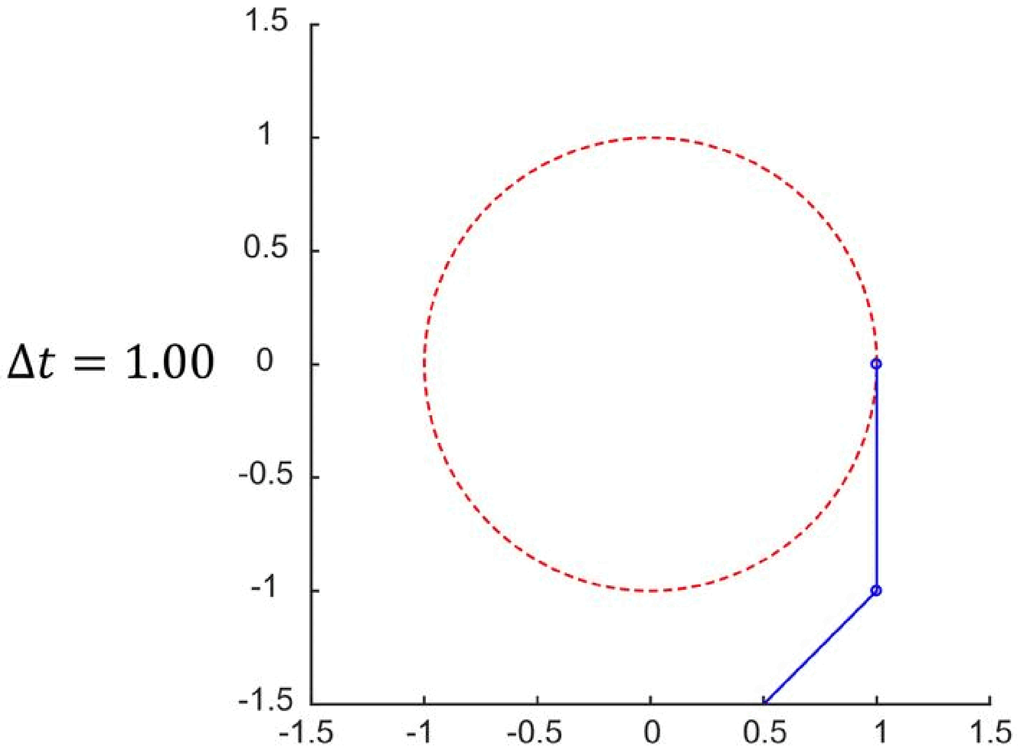
Example:

A particle moving around a circle



Explicit (Forward) Euler's Method (3/3)

The result of explicit Euler's method explodes!



Pros:

- Easy to calculate, efficient

Cons:

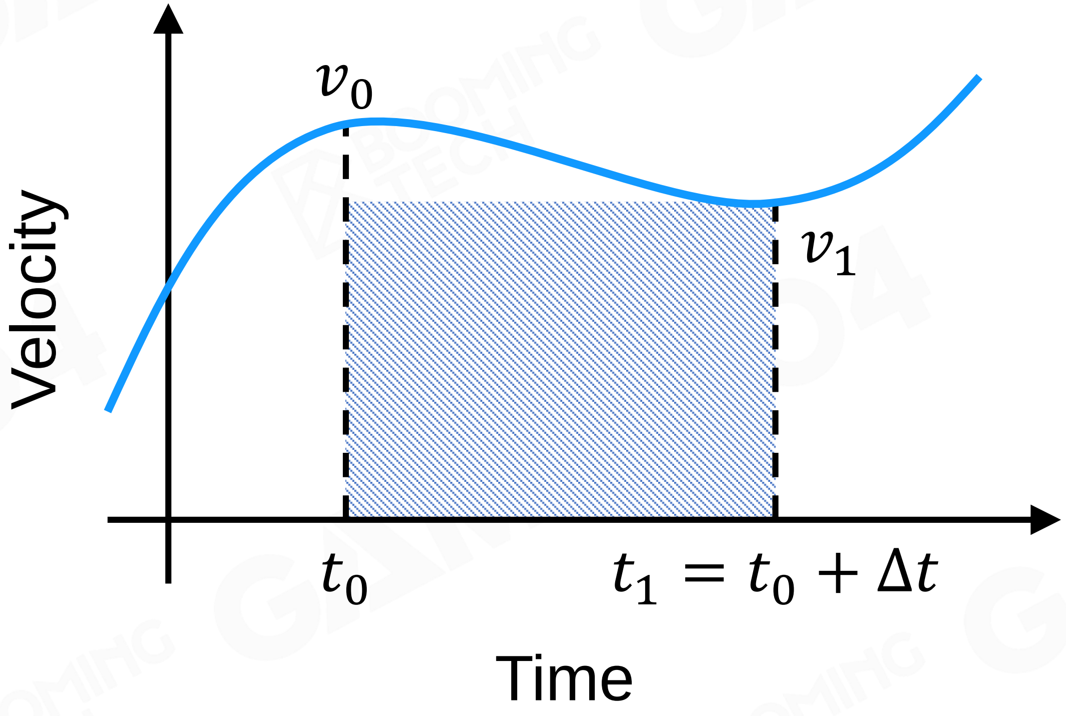
- Poor stability
- Energy growing as time progresses

Implicit (Backward) Euler's Method (1/2)

$$\begin{cases} \vec{v}(t_1) = \vec{v}(t_0) + M^{-1}\boxed{\vec{F}(t_1)}\Delta t \\ \vec{x}(t_1) = \vec{x}(t_0) + \boxed{\vec{v}(t_1)}\Delta t \end{cases}$$

Future states

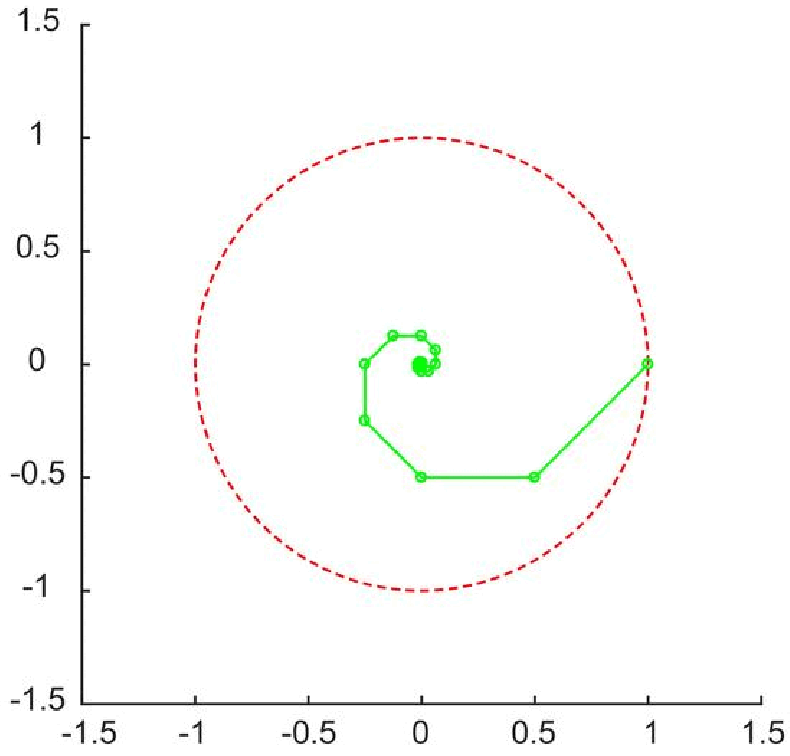
Unknown yet



Implicit (Backward) Euler's Method (2/2)

The result of implicit Euler's method spirals!

$\Delta t = 1.00$



Pros:

- Unconditionally stable

Cons:

- Expensive to solve
- Challenging to implement when non-linearity presents
- Energy attenuates as time progresses

Semi-implicit Euler's Method (1/2)

Explicit Euler's Method

$$\begin{cases} \vec{v}(t_1) = \vec{v}(t_0) + M^{-1}\vec{F}(t_0)\Delta t \\ \vec{x}(t_1) = \vec{x}(t_0) + \vec{v}(t_0)\Delta t \end{cases}$$

Implicit Euler's Method

$$\begin{cases} \vec{v}(t_1) = \vec{v}(t_0) + M^{-1}\vec{F}(t_1)\Delta t \\ \vec{x}(t_1) = \vec{x}(t_0) + \vec{v}(t_1)\Delta t \end{cases}$$

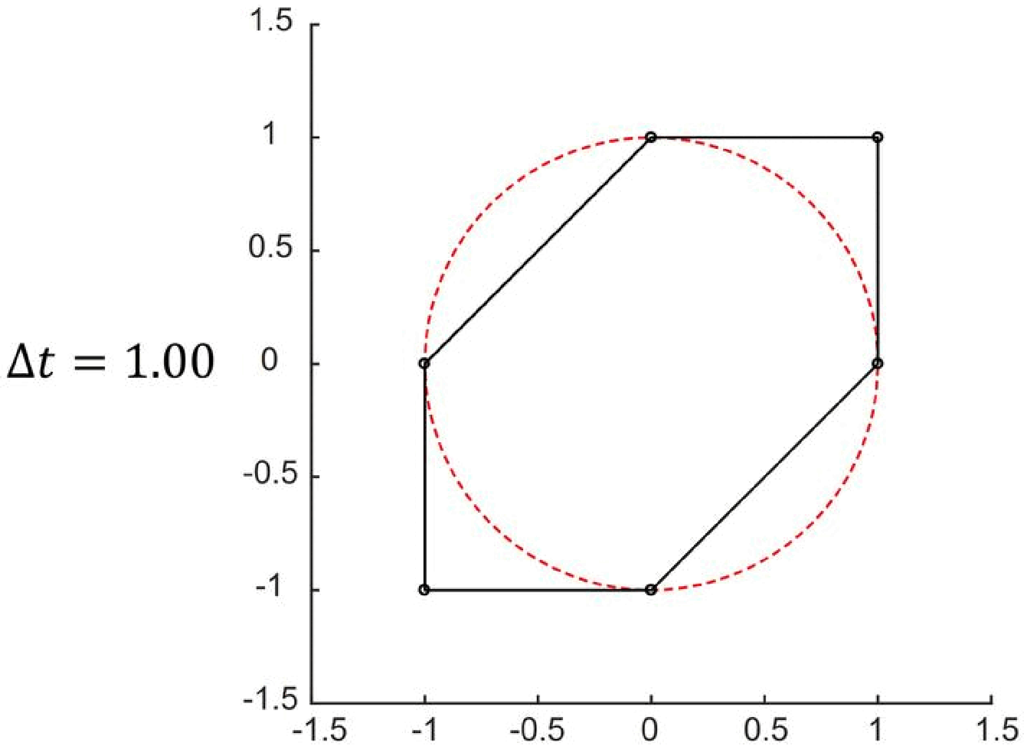
Current States

$$\begin{cases} \vec{v}(t_1) = \vec{v}(t_0) + M^{-1}\vec{F}(t_0)\Delta t \\ \vec{x}(t_1) = \vec{x}(t_0) + \vec{v}(t_1)\Delta t \end{cases}$$

Future states

Semi-implicit Euler's Method (2/2)

The result approximates the circle well if the timestep is small enough



- Conditionally stable
- Easy to calculate, efficient
- Preserves energy as time progresses

Rigid Body Dynamics

Particle Dynamics

- Position $\vec{x}$
- Linear Velocity $\vec{v} = \frac{\mathrm{d}\vec{x}}{\mathrm{d}t}$
- Acceleration $\vec{a} = \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{d^2\vec{x}}{dt^2}$
- Mass M
- Momentum $\vec{p} = M\vec{v}$
- Force $\vec{F} = \frac{d\vec{p}}{dt} = M\vec{a}$

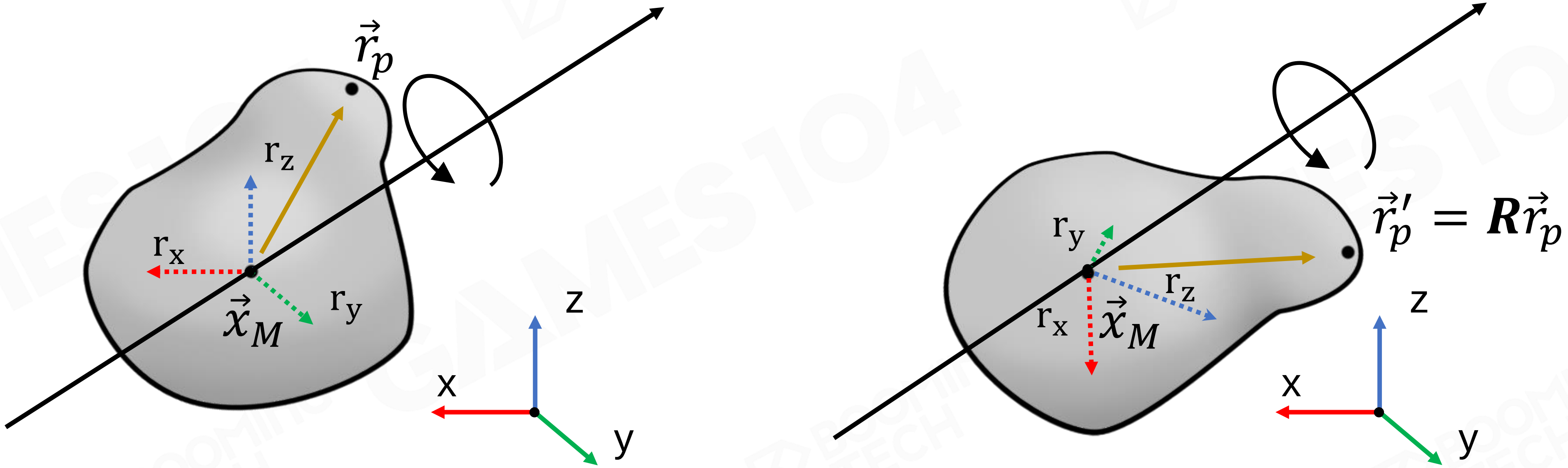
Rigid body Dynamics

Besides linear values, rigid body dynamics have angular values

- Orientation $\boldsymbol{R}$
- Angular velocity $\vec{\omega}$
- Angular acceleration $\vec{\alpha}$
- Inertia tensor $\mathbf{I}$
- Angular momentum $\vec{L}$
- Torque $\vec{\tau}$

Orientation – $\boldsymbol{R}$

A matrix $\mathbf{R}(\mathrm{t}) = \begin{bmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{bmatrix}$ or a quaternion $q = [s, \vec{v}]$



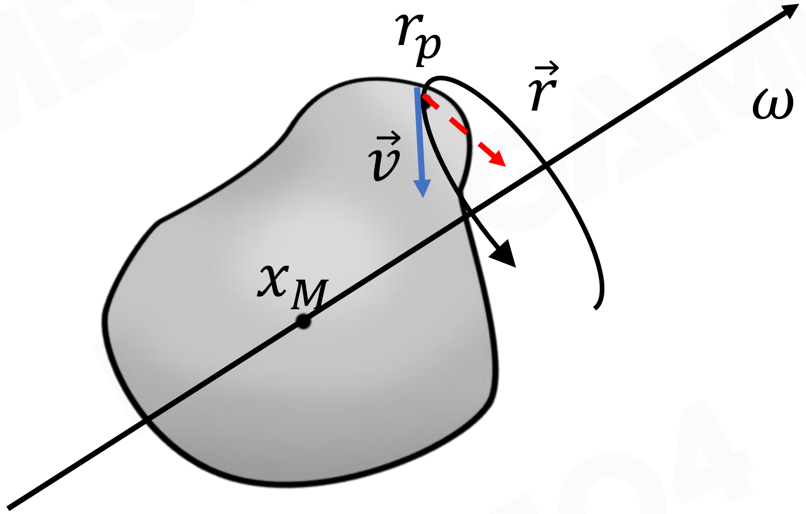
Angular Velocity – $\vec{\omega}$

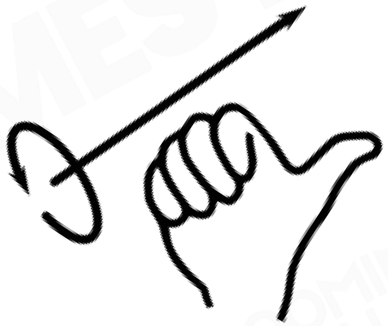
Direction of $\vec{\omega}$ is the direction of the rotation axis

θ : rotated angle in radians

$$\|\vec{\omega}\| = \frac{d\theta}{dt}$$

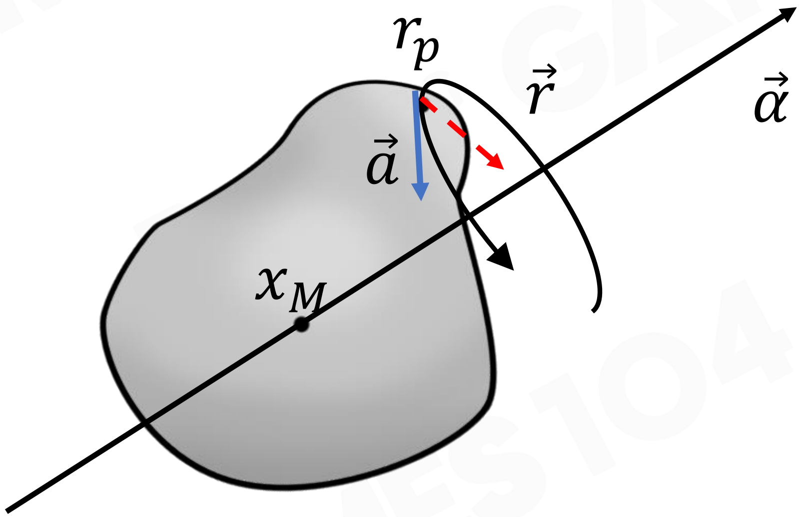
$$\vec{\omega} = \frac{\vec{v} \times \vec{r}}{\|\vec{r}\|^2}$$





Angular Acceleration – $\vec{\alpha}$

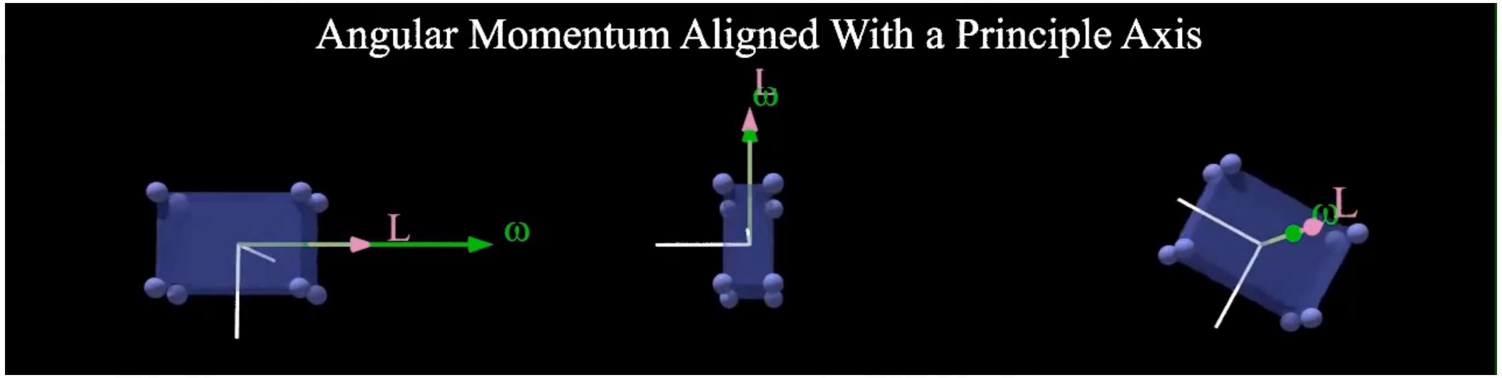
$$\vec{\alpha} = \frac{d\vec{\omega}}{dt} = \frac{\vec{a} \times \vec{r}}{\|\vec{r}\|^2}$$



Rotational Inertia – $\mathbf{I}$ (1/2)

- Rotational inertia describes the distribution of mass for a rigid body

$$\mathbf{I} = \mathbf{R} \cdot \mathbf{I}_0 \cdot \mathbf{R}^T$$



Rotational Inertia – $\mathbf{I}$ (2/2)

Total Mass: $M = m_1 + m_2$

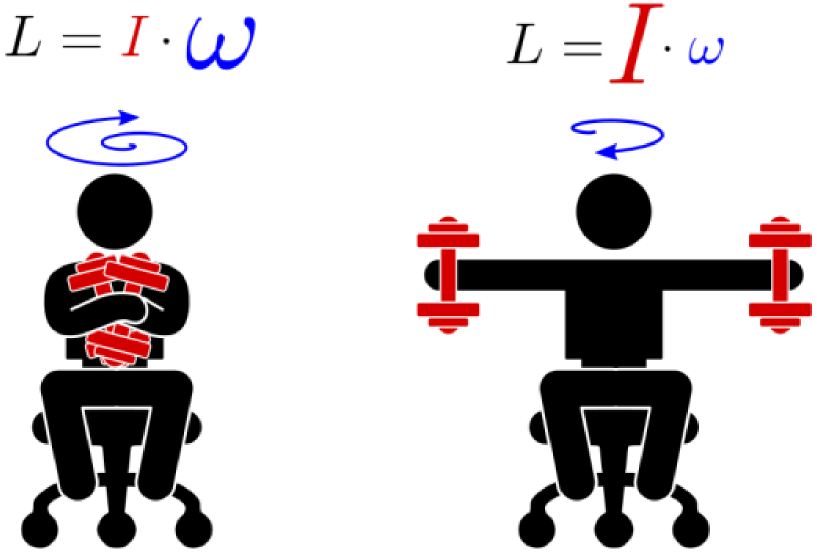
Center of Mass: $CoM = \frac{m_1}{M}(x_1, y_1, z_1) + \frac{m_2}{M}(x_2, y_2, z_2)$

Initial Inertia Tensor:

$$I_0 = \begin{bmatrix} m_1(y_1^2 + z_1^2) + m_2(y_2^2 + z_2^2) & -m_1x_1y_1 - m_2x_2y_2 & -m_1x_1z_1 - m_2x_2z_2 \\ -m_1y_1x_1 - m_2y_2x_2 & m_1(x_1^2 + z_1^2) + m_2(x_2^2 + z_2^2) & -m_1y_1z_1 - m_2y_2z_2 \\ -m_1z_1x_1 - m_2z_2x_2 & -m_1z_1y_1 - m_2z_2y_2 & m_1(x_1^2 + y_1^2) + m_2(x_2^2 + y_2^2) \end{bmatrix}$$

Angular Momentum – $\vec{L}$

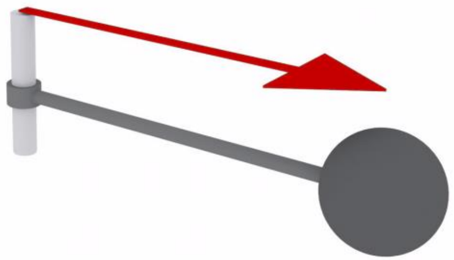
$$\vec{L} = \mathbf{I}\vec{\omega}$$



Torque – $\vec{\tau}$

We denote external force $\vec{F}$ exerted on position $\vec{r}$ on the rigid body, therefore

$$\vec{\tau} = \vec{r} \times \vec{F} = \frac{d\vec{L}}{dt}$$



$$\vec{\tau} = \vec{r} \times \vec{F}$$
$$\vec{L} = \vec{r} \times \vec{p}$$

Summary

- Angular Values vs. Linear Values
  - Orientation $\boldsymbol{R}$
  - Angular velocity $\vec{\omega} = \frac{\vec{v} \times \vec{r}}{\|\vec{r}\|^2}$
  - Angular acceleration $\vec{\alpha} = \frac{\mathrm{d}\vec{\omega}}{\mathrm{d}t} = \frac{\vec{a} \times \vec{r}}{\|\vec{r}\|^2}$
  - Inertia tensor $\mathbf{I} = \mathbf{R} \cdot \mathbf{I_0} \cdot \mathbf{R}^{\mathrm{T}}$
  - Angular momentum $\vec{L} = \boldsymbol{I}\vec{\omega}$
  - Torque $\vec{\tau} = \frac{d\vec{L}}{dt}$
  - Position $\vec{x}$
  - Linear velocity $\vec{v} = \frac{\mathrm{d}\vec{x}}{\mathrm{d}t}$
  - Linear acceleration $\vec{a} = \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{d^2\vec{x}}{dt^2}$
  - Mass $M = \sum m_i$
  - Linear momentum $\vec{p} = M\vec{v}$
  - Force $\vec{F} = \frac{d\vec{p}}{dt} = m\vec{a}$

Application – Billiard Dynamics (1/2)

Even though we have known the elements of rigid body dynamics, the physics in a light billiard game is still complicated…

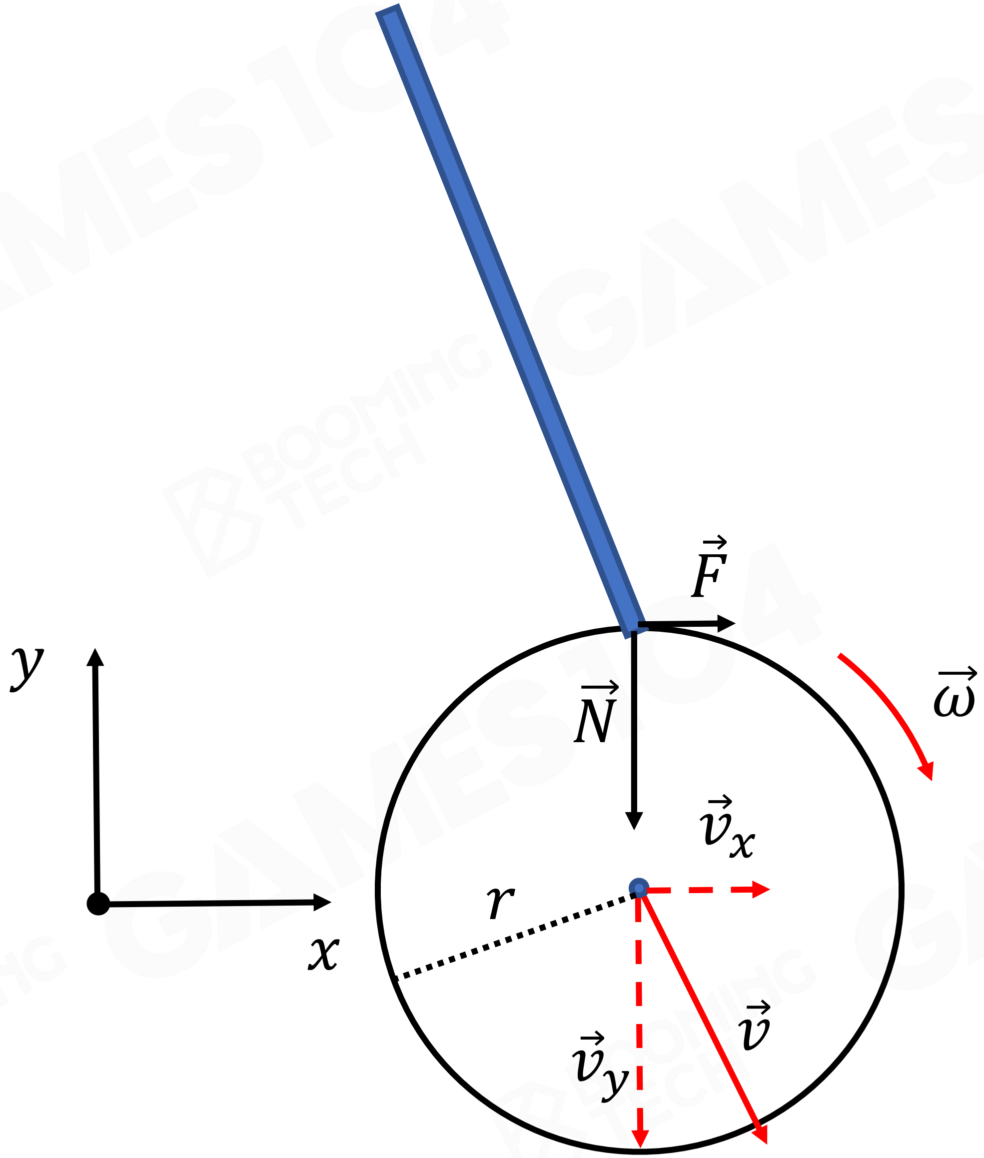
Application – Billiard Dynamics (2/2)

Friction Impulse: $\vec{p}_F = \int \vec{F} dt = m\vec{v}_x$

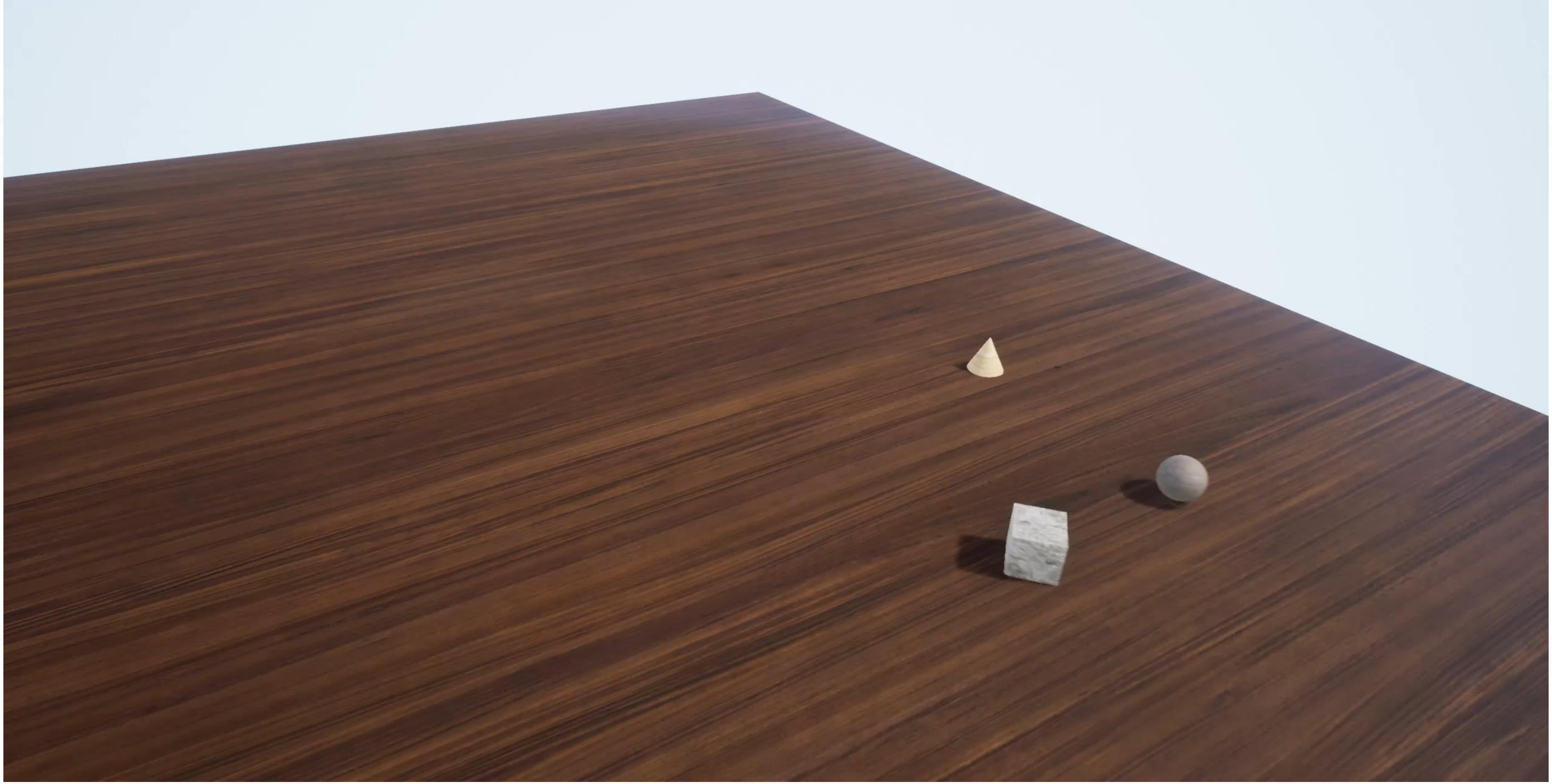
Pressure Impulse: $\vec{p}_N = \int \vec{N} dt = m\vec{v}_y$

Ball Angular Momentum: $\vec{L}_b = \boldsymbol{I}_b \vec{\omega} = \vec{p}_F \times \vec{r}_F$

Ball Linear Velocity: $\vec{v} = \vec{v}_x + \vec{v}_y$



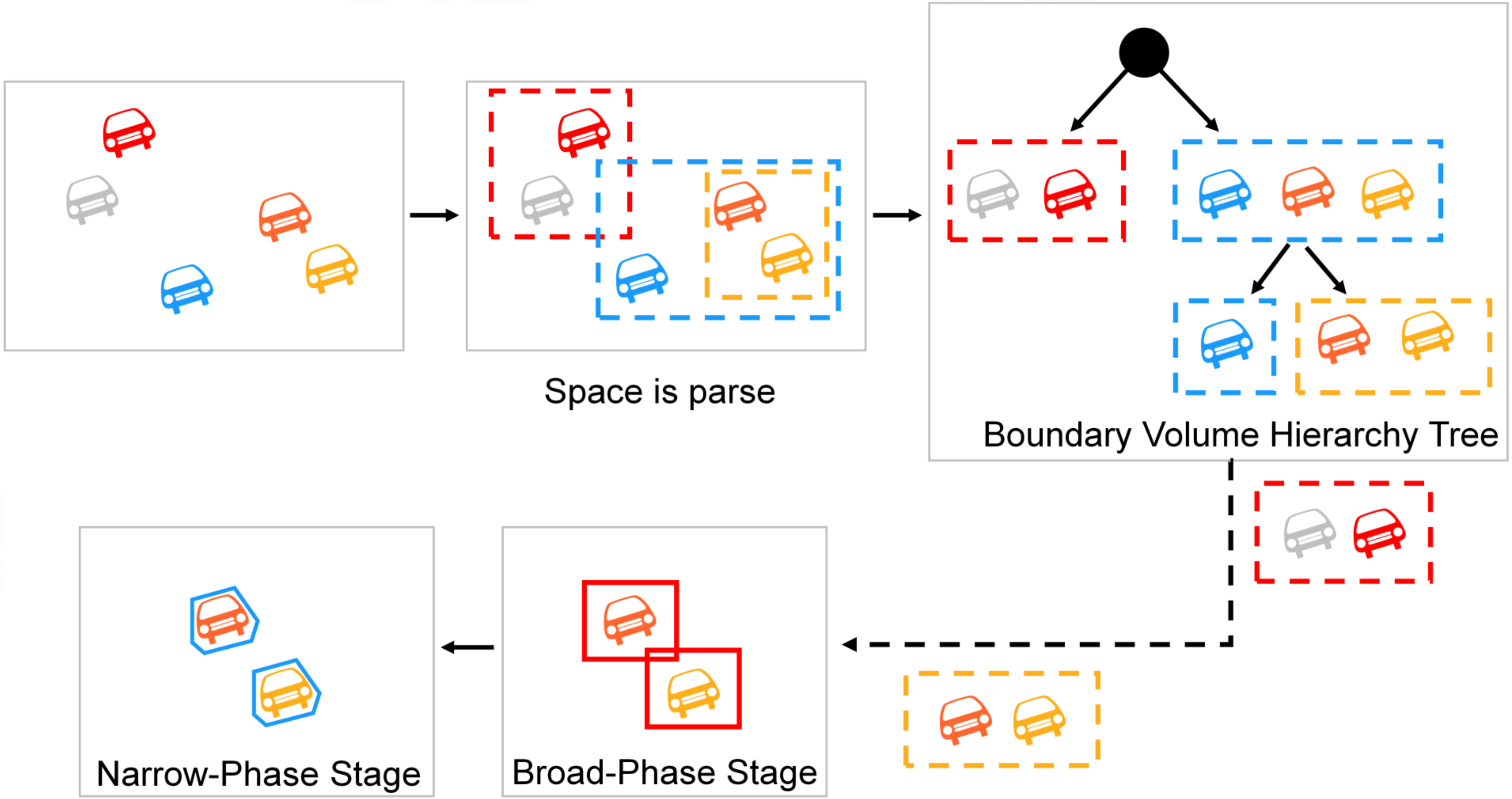
Collision Detection



Collision Detection – Two Phases

- Broad phase
  - Find intersected rigid body AABBs
  - Potential overlapped rigid body pairs
- Narrow phase
  - Detect overlapping precisely
  - Generate contact information

Broad Phase and Narrow Phase

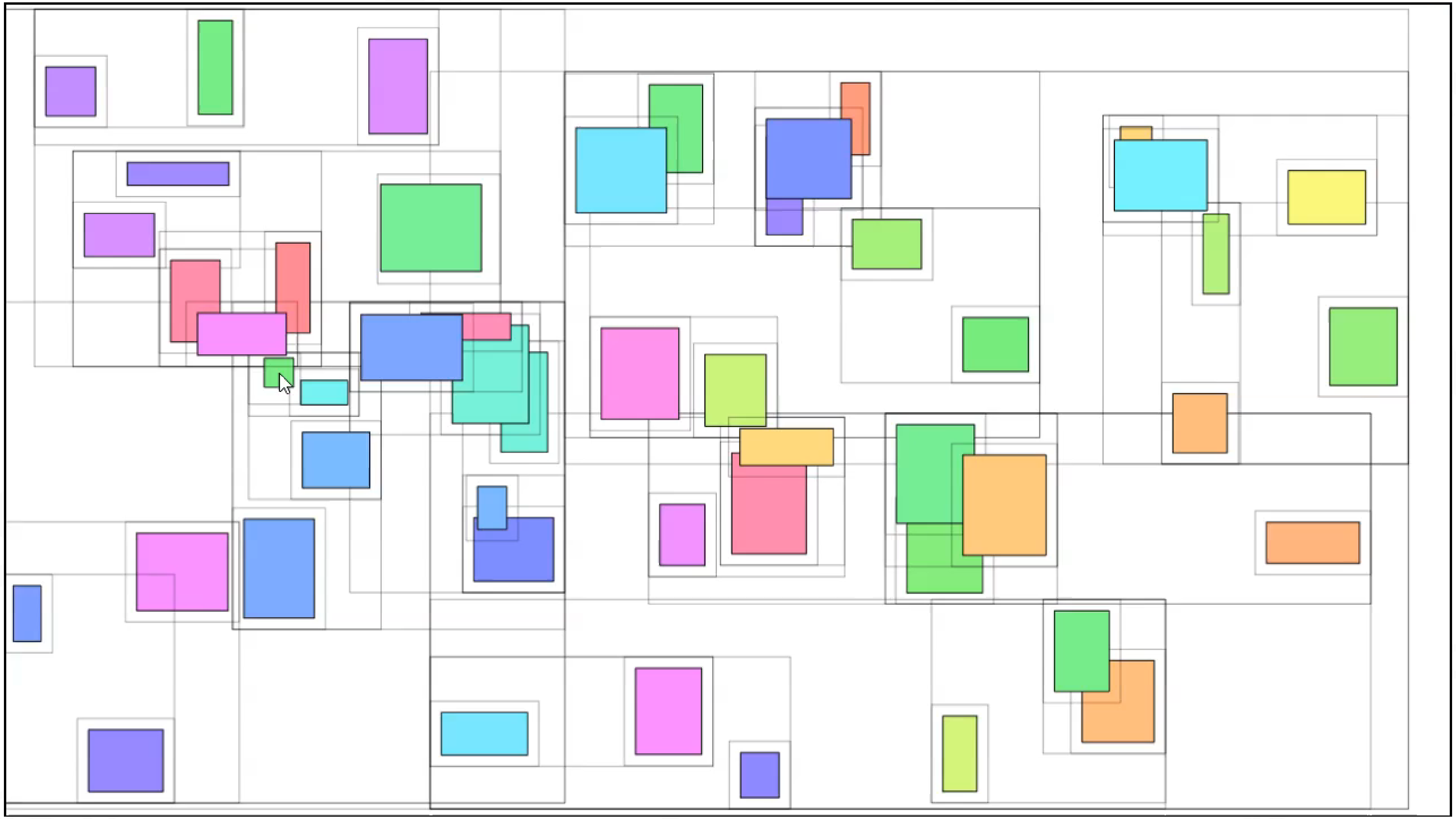


Broad Phase

- Objective
  - Find intersected rigid body AABBs
  - Potential overlapped rigid body pairs

- Two approaches
  - Space partitioning
    - i. e. Boundary Volume Hierarchy (BVH) Tree
  - Sort and Sweep

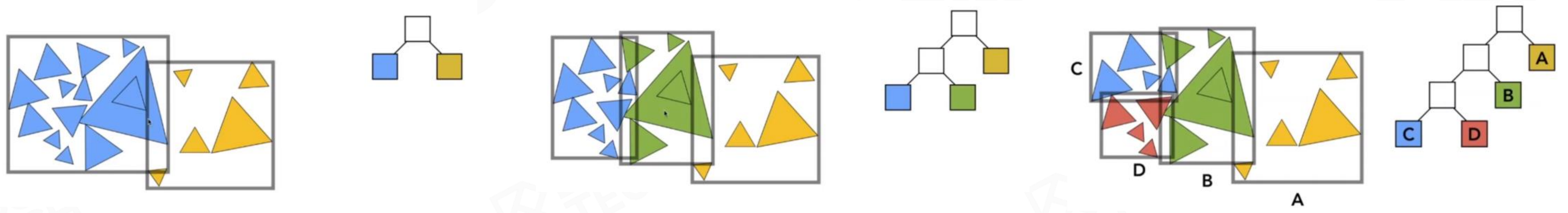
Broad Phase - BVH Tree (1/2)



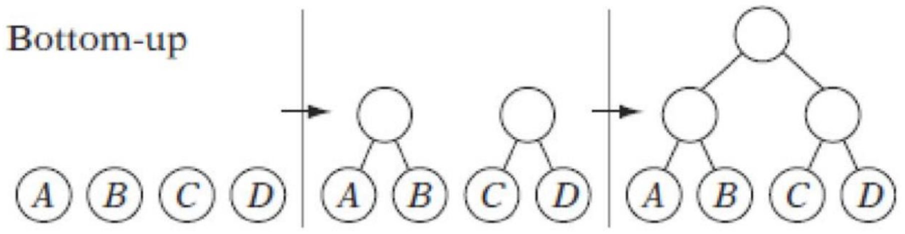
Broad Phase - BVH Tree (2/2)

Top-down

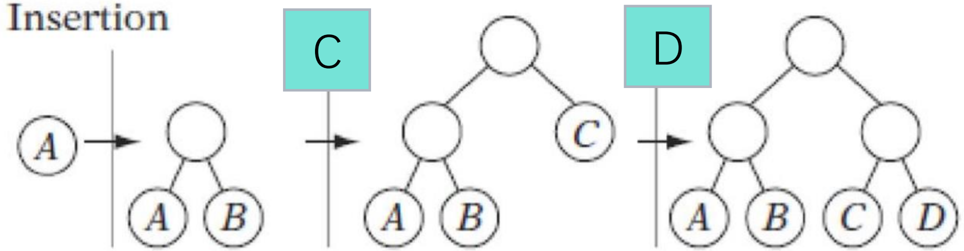
Recap: Dynamic BVH Tree



Bottom-up



Incremental tree-insertion

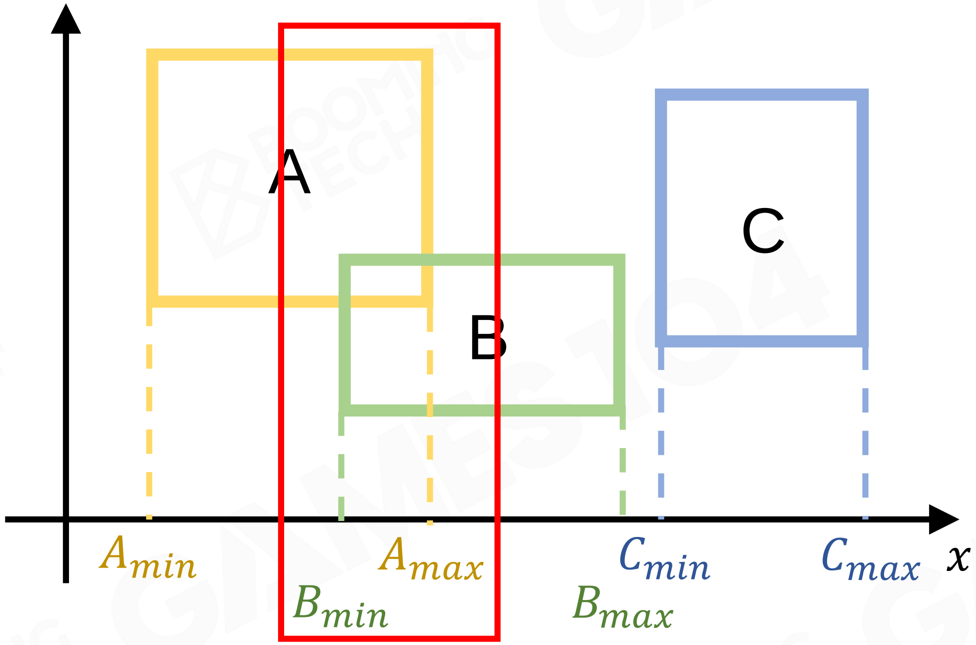


Broad Phase - Sort and Sweep (1/2)

Sorting Stage (Initialize)

For each axis

- Sort AABB bounds along each axis when initializing the scene
- Check AABB bounds of actors along each axis
- $A_{max} \geq B_{min}$ indicates potential overlap of A and B



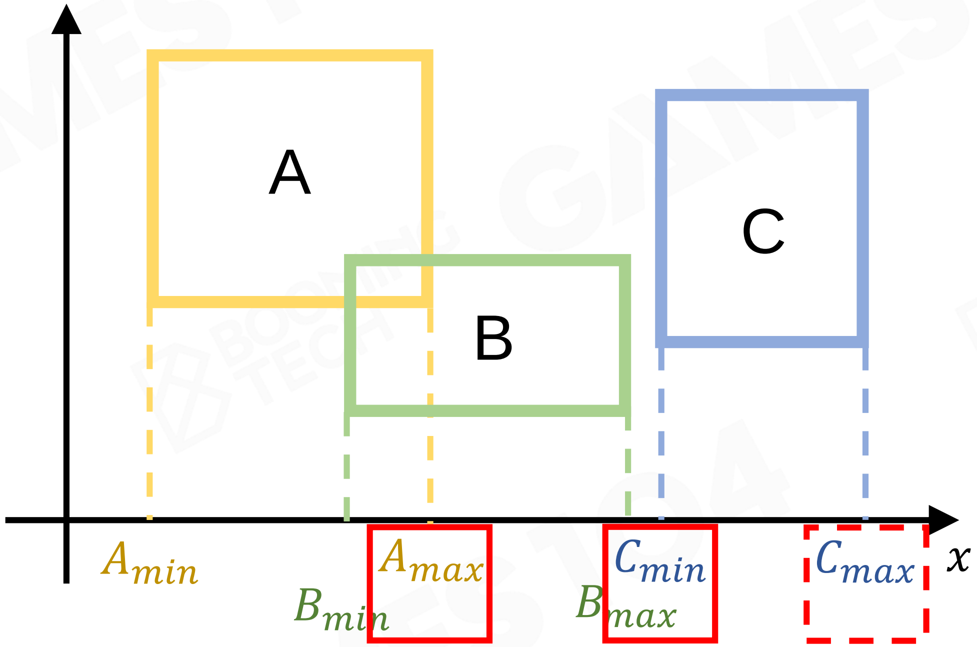
Sorted x-bounds: $[A_{min}, B_{min}, A_{max}, B_{max}, C_{min}, C_{max}]$

Overlaps Set: { (A, B) }

Broad Phase - Sort and Sweep (2/2)

Sweeping Stage (Update)

- Only check swapping of bounds
  - temporal coherence
  - local steps from frame to frame
- Swapping of min and max indicates add/delete potential overlap pair from overlaps set
- Swapping of min and min or max and max does not affect overlaps set



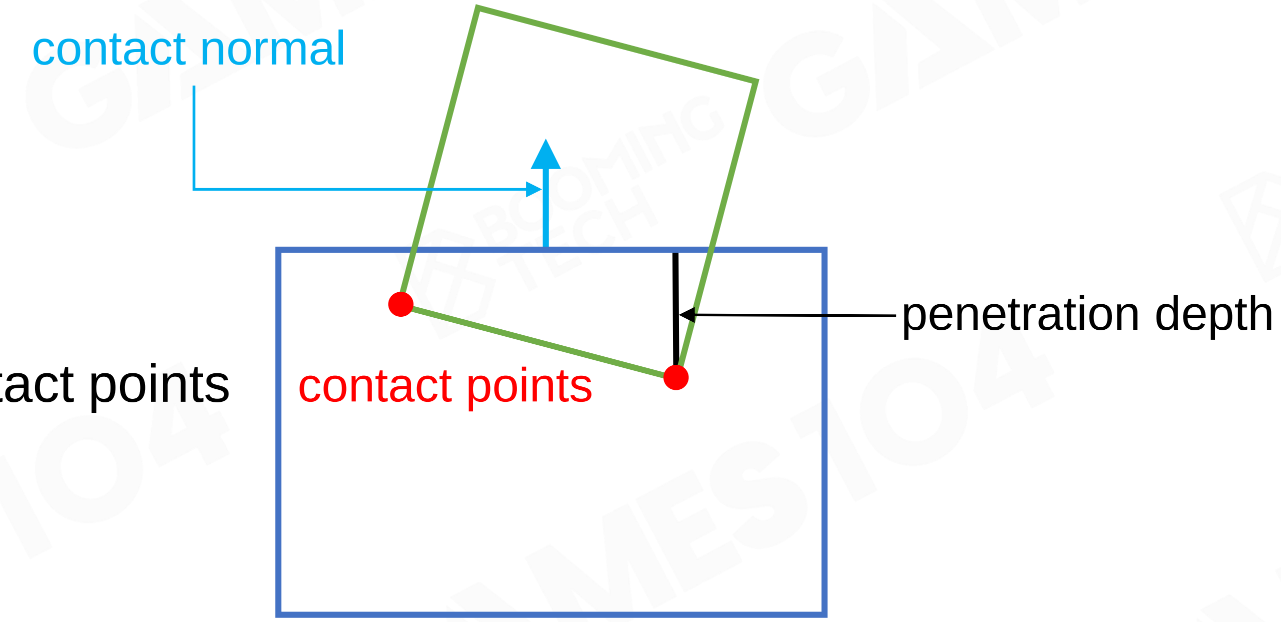
Sorted x-bounds: [$A_{min}, B_{min}, A_{max}, C_{min}, B_{max}, C_{max}$]

Overlaps Set: { (A, B), (B, C) }

No change on overlaps set

Narrow Phase – Objectives

- Detect overlapping precisely
- Generate contact information
  - Contact manifold
    - approximated with a set of contact points
  - Contact normal
  - Penetration depth



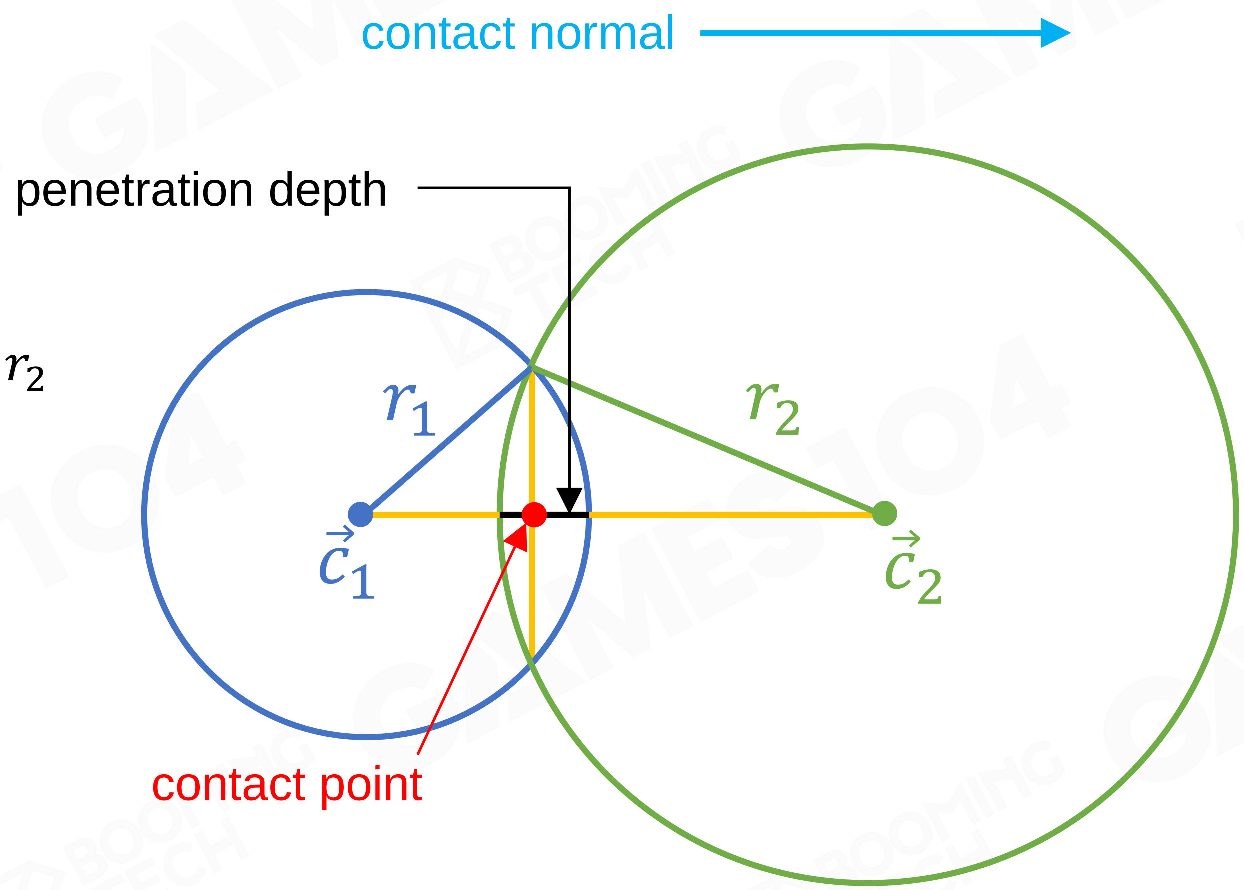
Narrow Phase – Approaches

- Three approaches
  - Basic Shape Intersection Test
  - Minkowski Difference-based Methods
  - Separating Axis Theorem

Basic Shape Intersection Test (1/3)

Sphere-Sphere Test

- overlap: $|\vec{c}_2 - \vec{c}_1| - r_1 - r_2 \leq 0$
- contact information:
  - contact normal: $\vec{c}_2 - \vec{c}_1 / |\vec{c}_2 - \vec{c}_1|$
  - penetration depth: $|\vec{c}_2 - \vec{c}_1| - r_1 - r_2$



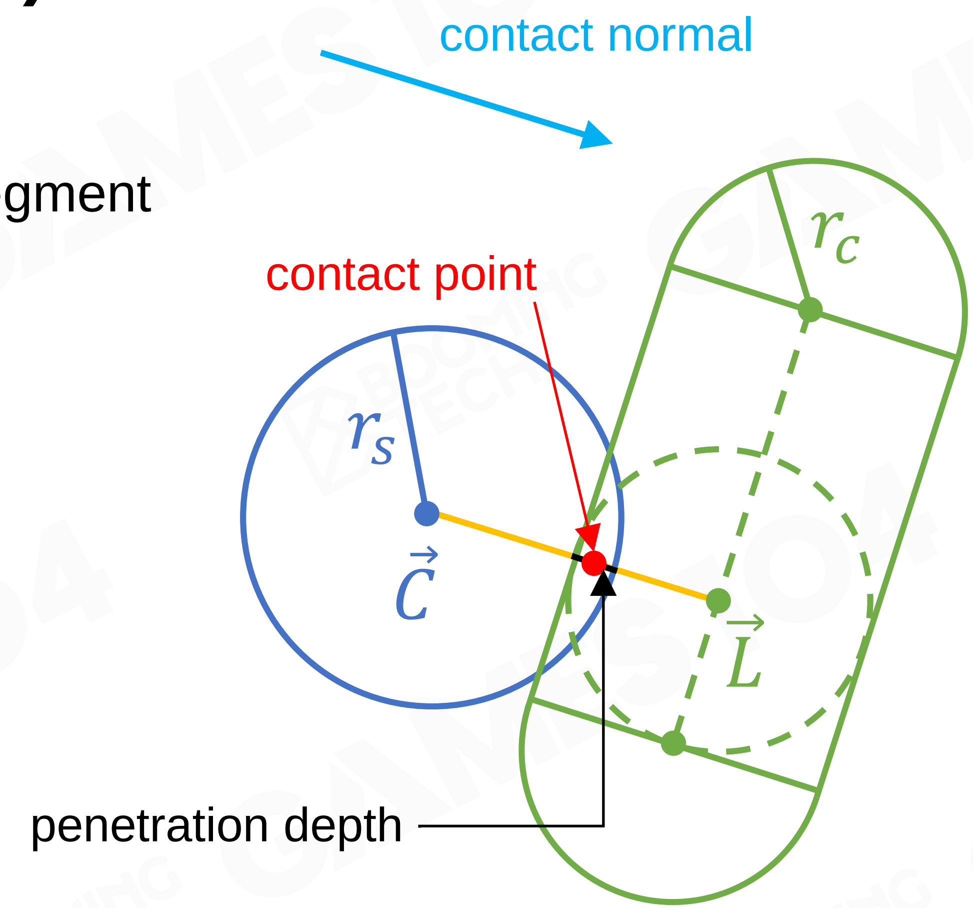
Basic Shape Intersection Test (2/3)

Sphere-Capsule Test

$\vec{L}$ is the closest point on the inner capsule segment

- overlap: $|\vec{C} - \vec{L}| - r_S - r_C \leq 0$
- contact information:
- contact normal: $\vec{L} - \vec{C} / |\vec{L} - \vec{C}|$

penetration depth: $|\vec{C} - \vec{L}| - r_S - r_C$



Basic Shape Intersection Test (3/3)

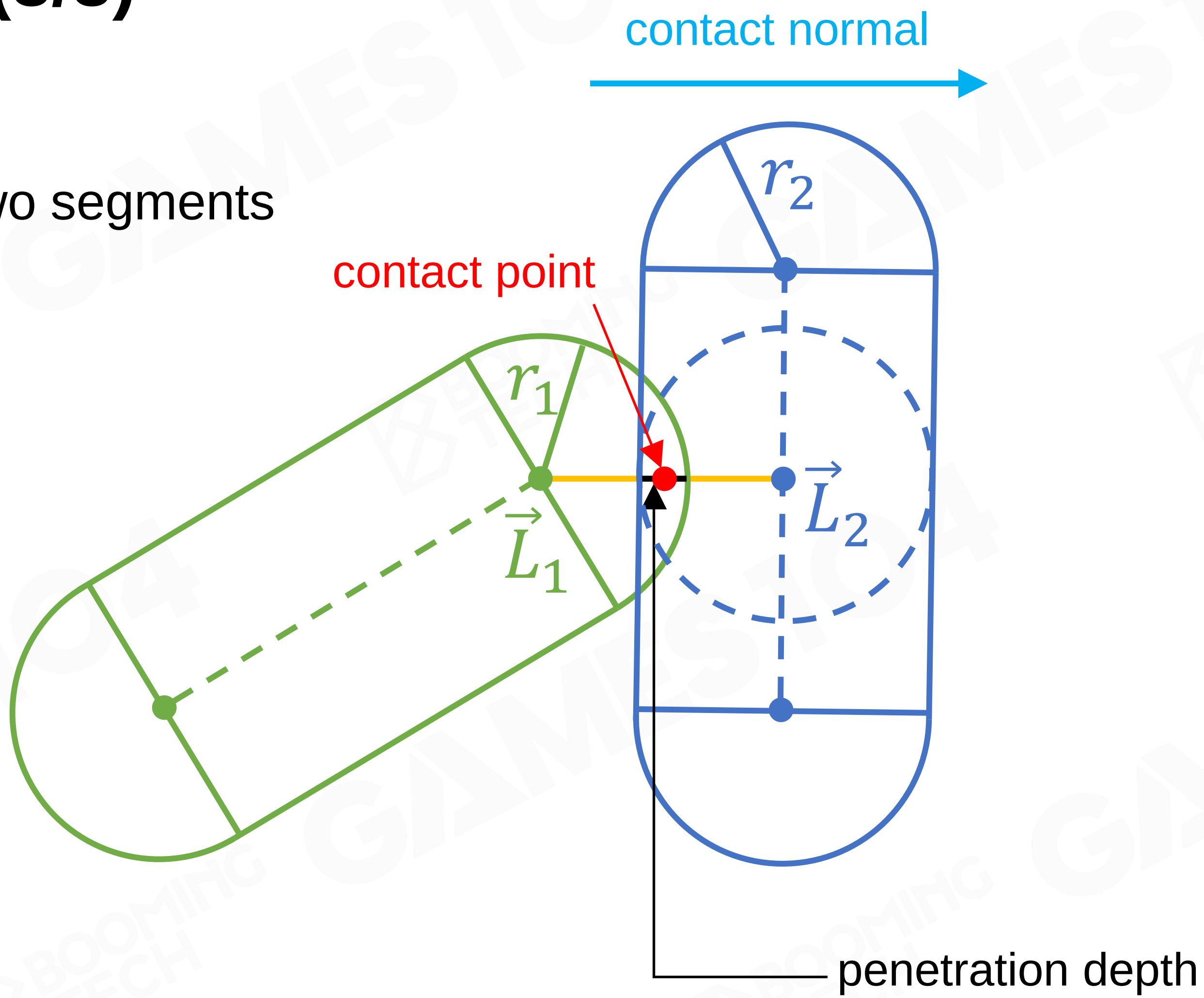
- Capsule-Capsule Test

  $\vec{L}_1$ and $\vec{L}_2$ are the closest points on the two segments

  overlap: $\left|\vec{L}_2 - \vec{L}_1\right| - r_1 - r_2 \leq 0$

  contact normal: $\vec{L}_2 - \vec{L}_1 / \left|\vec{L}_2 - \vec{L}_1\right|$

  penetration depth: $\left|\vec{L}_2 - \vec{L}_1\right| - r_1 - r_2$



Minkowski Difference-based Methods - Concepts

- Minkowski Sum
  - Points from A + Points from B =

    Points in Minkowski Sum of A and B

    $$A \oplus B = \{ \vec{a} + \vec{b} : \vec{a} \in A, \vec{b} \in B \}$$

$A: \{ \vec{a}_1, \vec{a}_2 \}$

$B: \{ \vec{b}_1, \vec{b}_2, \vec{b}_3 \}$

$$A \oplus B = \{ \vec{a}_1 + \vec{b}_1, \vec{a}_1 + \vec{b}_2, \vec{a}_1 + \vec{b}_3, \vec{a}_2 + \vec{b}_1, \vec{a}_2 + \vec{b}_2, \vec{a}_2 + \vec{b}_3 \}$$

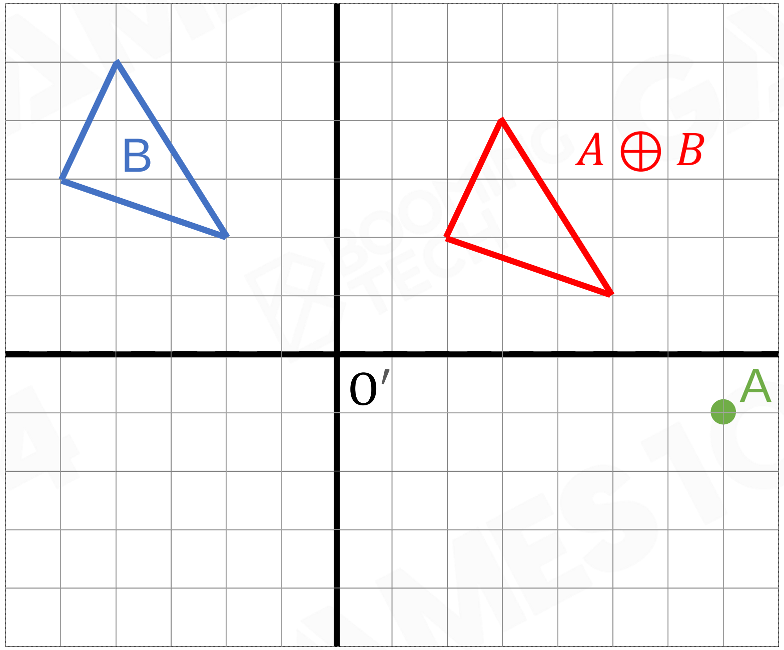
Hermann Minkowski
1864 - 1909

Minkowski Sum (1/3)

- Points from A + Points from B =

  Points in Minkowski Sum of A and B

$$A \oplus B = \{\, \vec{a} + \vec{b} : \vec{a} \in A, \vec{b} \in B \,\}$$

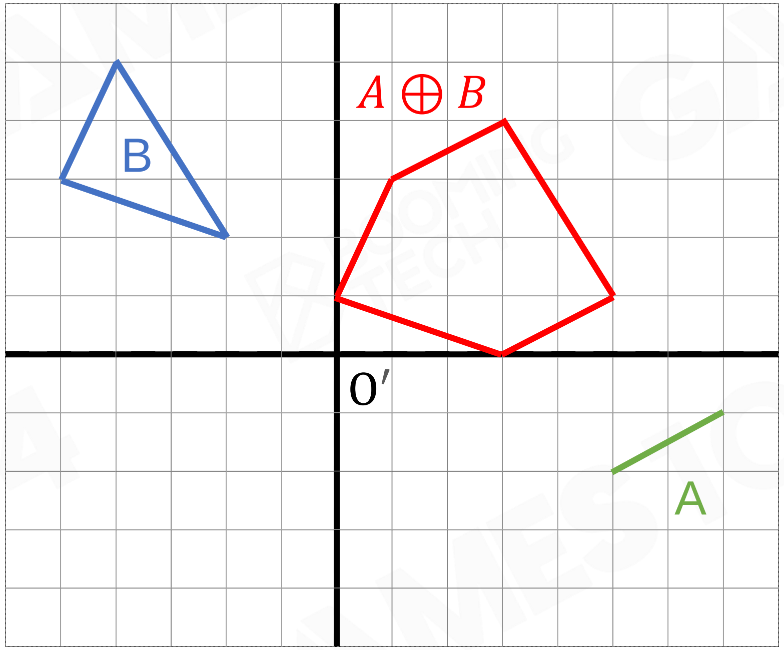


Minkowski Sum (2/3)

- Points from A + Points from B =

  Points in Minkowski Sum of A and B

$$A \oplus B = \{ \vec{a} + \vec{b} : \vec{a} \in A, \vec{b} \in B \}$$

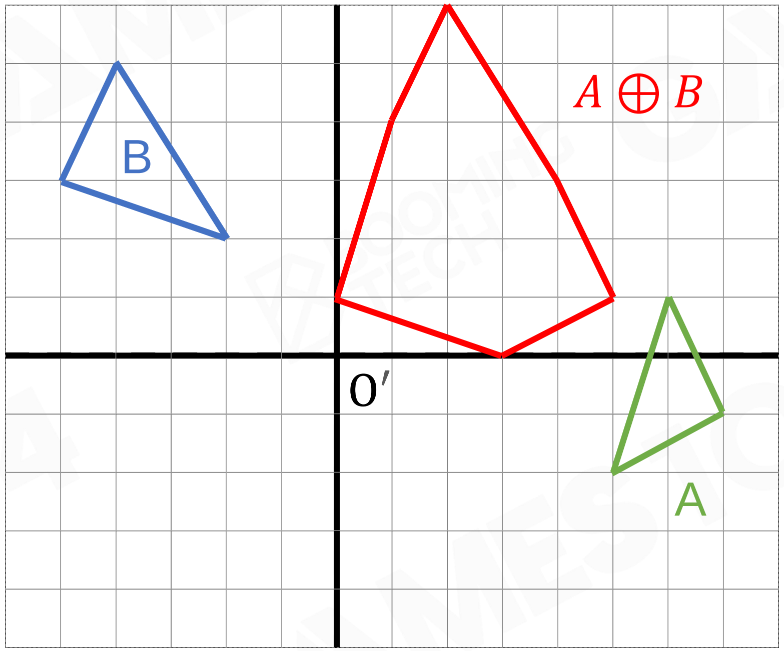


Minkowski Sum (3/3)

- Points from A + Points from B =

  Points in Minkowski Sum of A and B

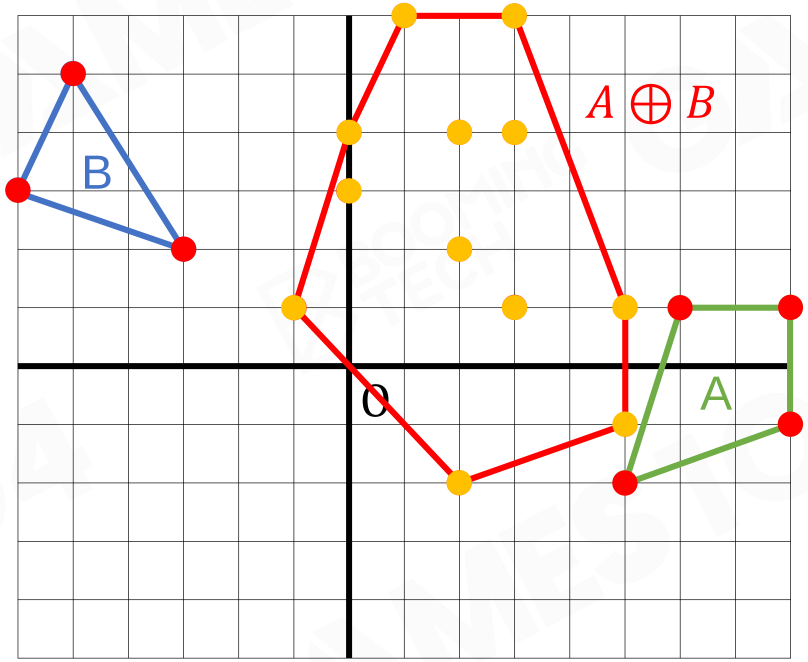
$$A \oplus B = \{ \vec{a} + \vec{b} : \vec{a} \in A, \vec{b} \in B \}$$



Minkowski Sum - Convex Polygons

$$A \oplus B = \{\, \vec{a} + \vec{b} : \vec{a} \in A, \vec{b} \in B \,\}$$

- Theorem
  - For convex polygons A and B, $A \oplus B$ is also a convex polygon
- The vertices of $A \oplus B$ are the sum of the vertices of A and B



Minkowski Difference

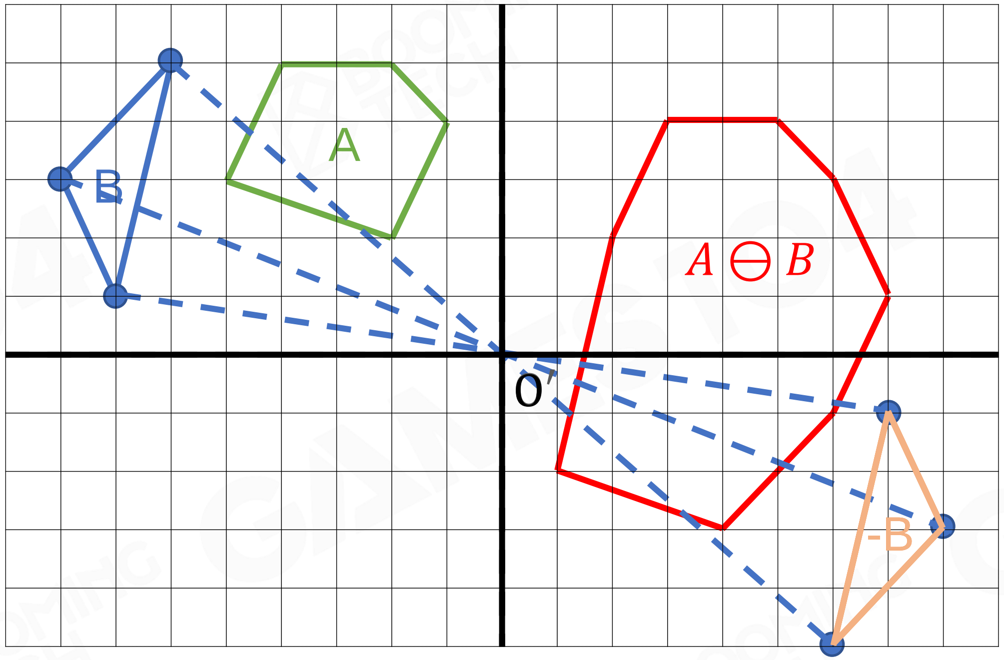
- Points from A – Points from B =

  Points in Minkowski Difference of A and B

$$A \ominus B = \{ \vec{a} - \vec{b} : \vec{a} \in A, \vec{b} \in B \}$$

- Minkowski sum of A and mirrored B

$$A \ominus B = A \oplus (-B)$$

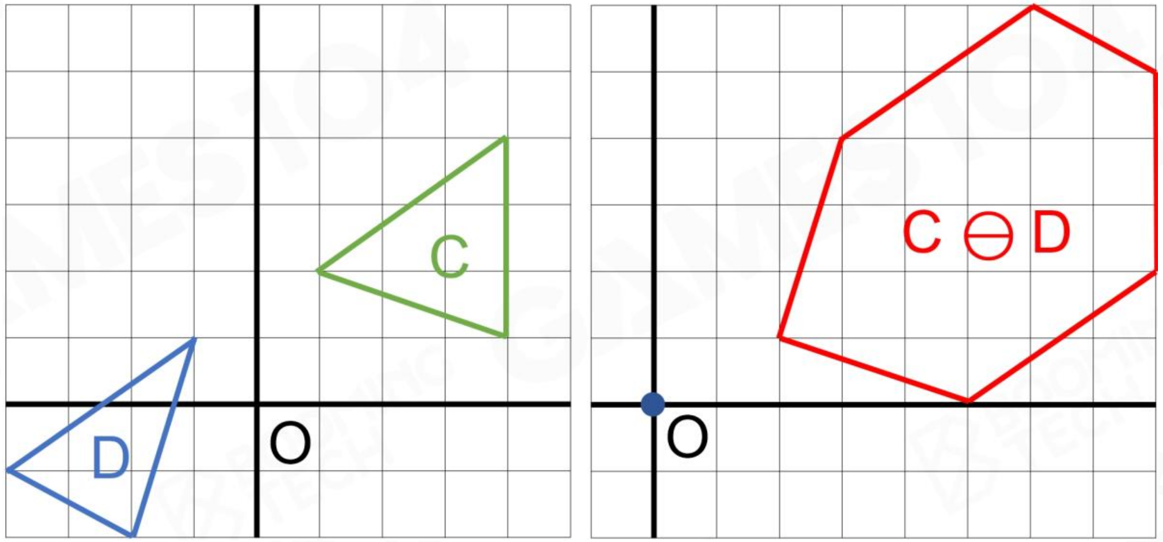


Origin and Minkowski Difference

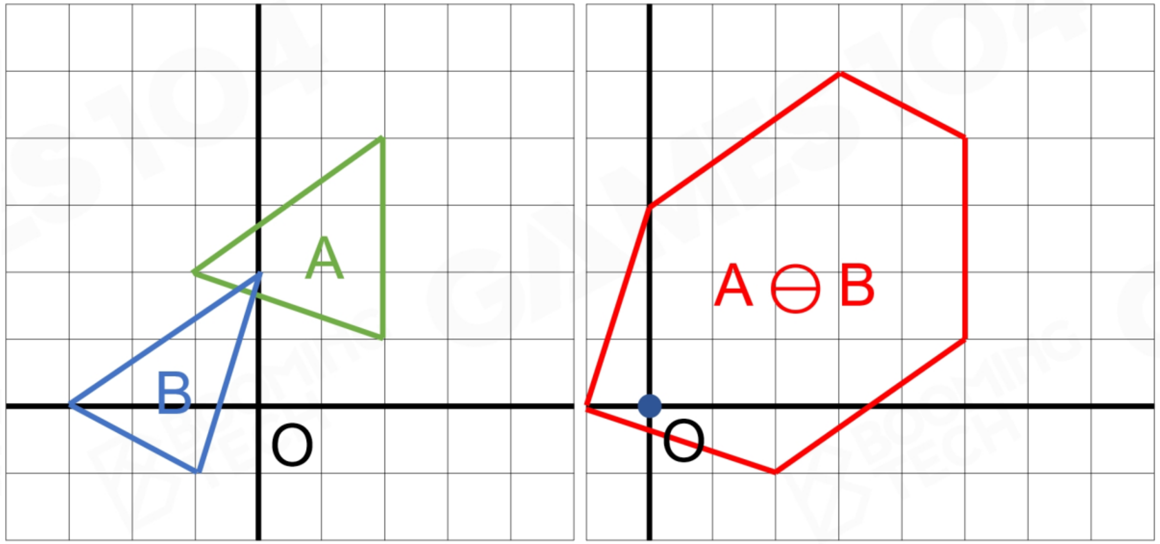
$$A \ominus B = \{\, \vec{a} - \vec{b} : \vec{a} \in A, \vec{b} \in B \,\}$$

- Same point in A and B
- The origin is in the Minkowski Difference!

Seperated Case

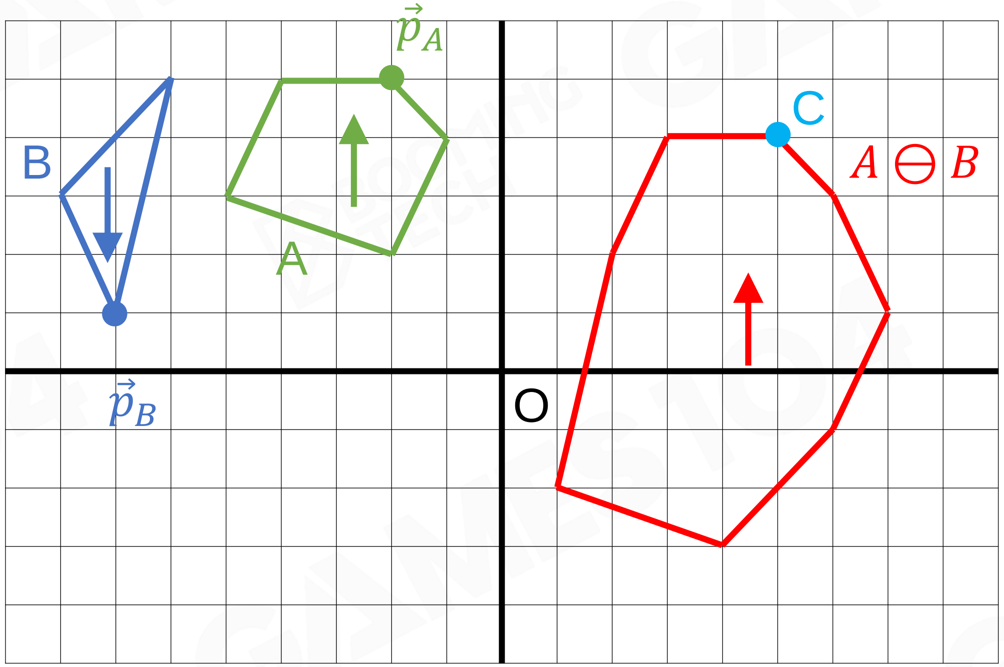


Overlapped Case



GJK Algorithm – Walkthrough (Separation Case) (1/5)

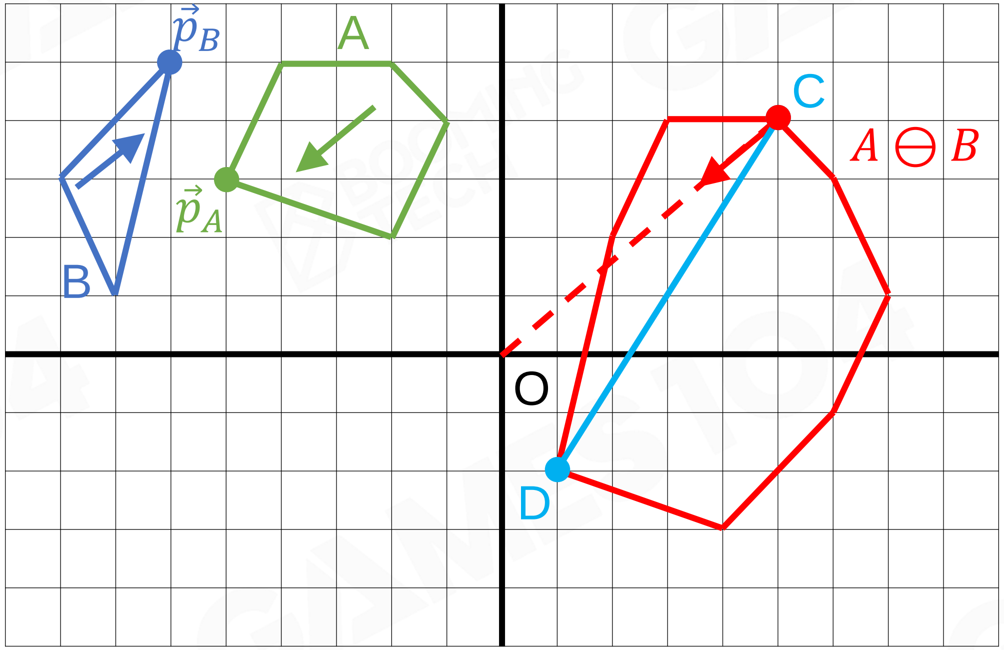
- Determine iteration direction
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference



Simplex Set: {C}

GJK Algorithm – Walkthrough (Separation Case) (2/5)

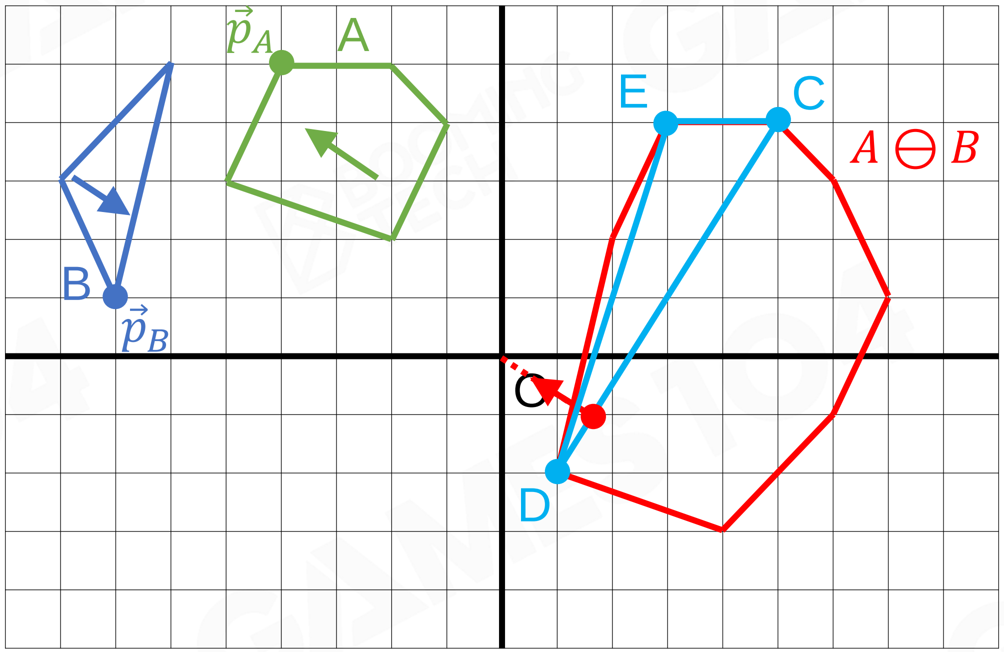
- Determine iteration direction
  - Check if origin is in the simplex
  - Find nearest point to origin in the simplex
  - If nearest distance reduced, continue iterating
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference



Simplex Set: {C} D}

GJK Algorithm – Walkthrough (Separation Case) (3/5)

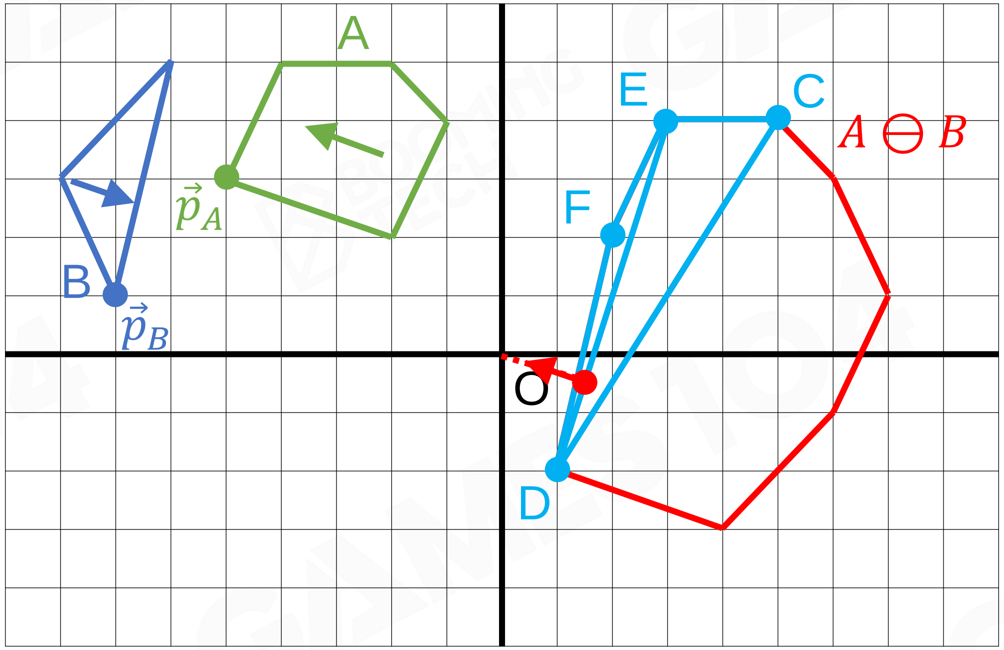
- Determine iteration direction
  - Check if origin is in the simplex
  - Find nearest point to origin in the simplex
  - If nearest distance reduced, continue iterating
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference



Simplex Set: {C, D} E}

GJK Algorithm – Walkthrough (Separation Case) (4/5)

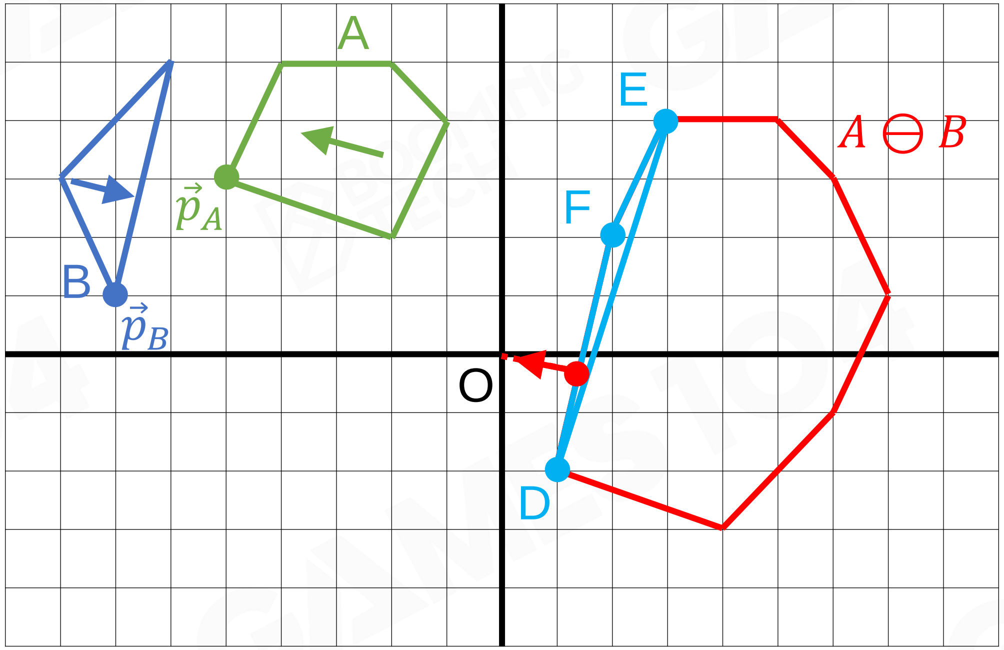
- Determine iteration direction
  - Check if origin is in the simplex
  - Find nearest point to origin in the simplex
  - If nearest distance reduced, continue iterating
- Remove point having no contribution to the new nearest point from simplex
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference



Simplex Set: {D, E}

GJK Algorithm – Walkthrough (Separation Case) (5/5)

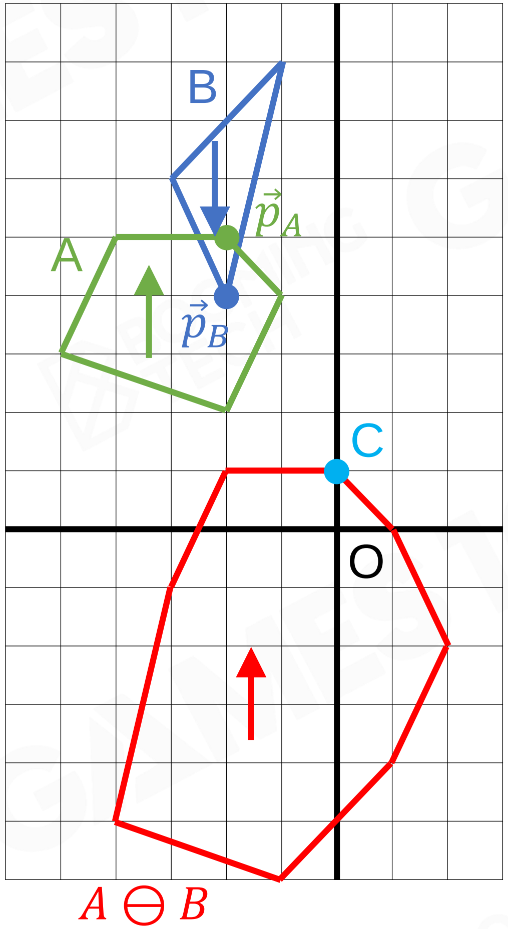
- Determine iteration direction
  - Check if origin is in the simplex
  - Find nearest point to origin in the simplex
  - If nearest distance reduced, continue iterating
- Remove point having no contribution to the new nearest point from simplex
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference



Simplex Set: {D, E} F}

GJK Algorithm – Walkthrough (Overlapped Case) (1/3)

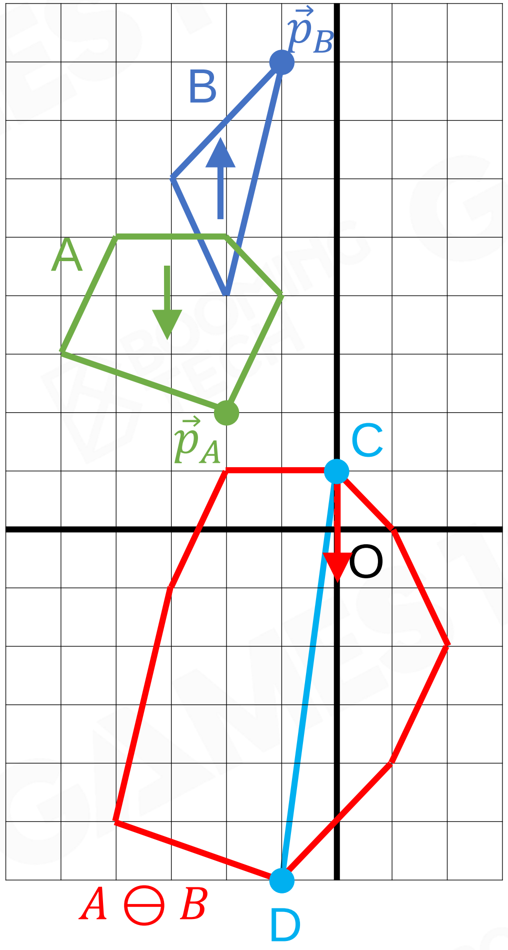
- Determine iteration direction
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference



Simplex Set: {C}

GJK Algorithm – Walkthrough (Overlapped Case) (2/3)

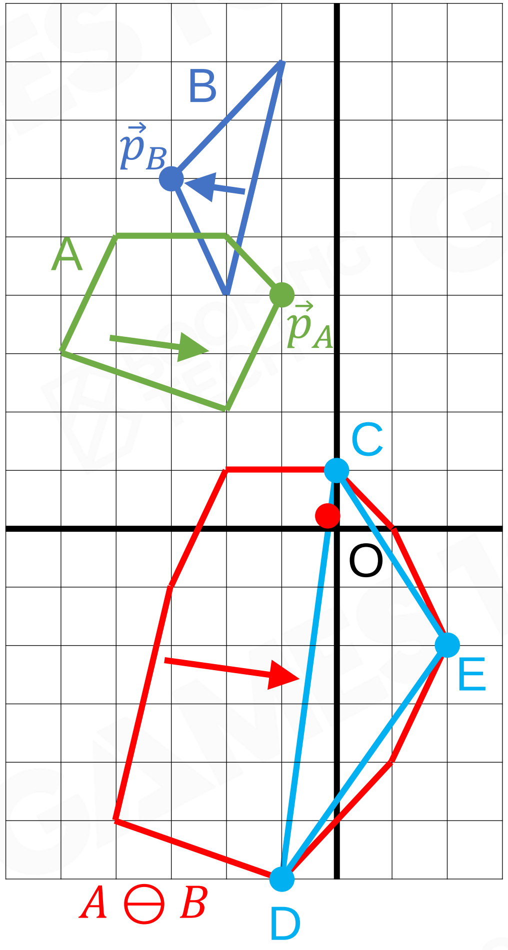
- Determine iteration direction
  - Check if origin is in the simplex
  - Find nearest point to origin in the simplex
  - If nearest distance reduced, continue iterating
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference



Simplex Set: {C} D}

GJK Algorithm – Walkthrough (Overlapped Case) (3/3)

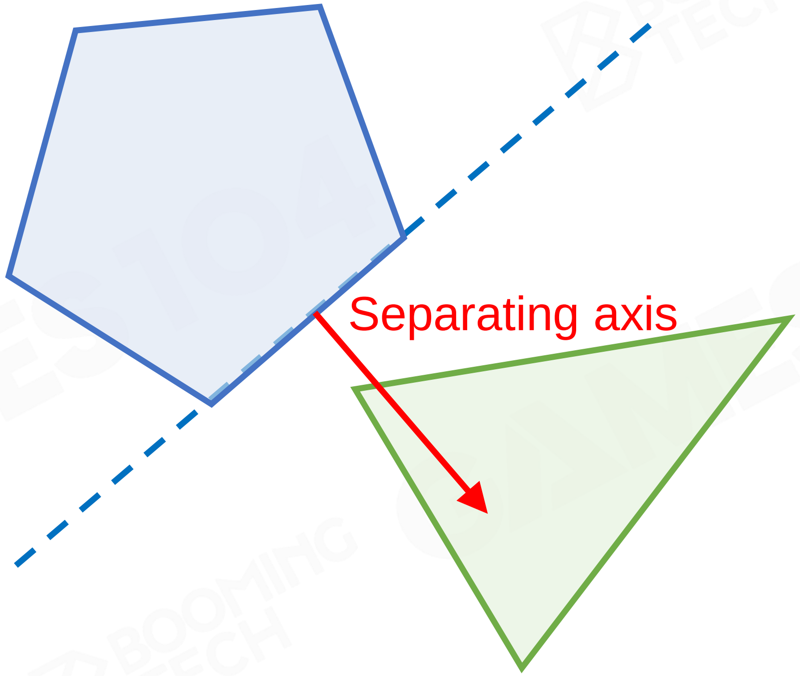
- Determine iteration direction
  - Check if origin is in the simplex
  - Find nearest point to origin in the simplex
  - If nearest distance reduced, continue iterating
- Find supporting points $\vec{p}_A$ and $\vec{p}_B$
- Add new point $\vec{p}_A - \vec{p}_B$ to iteration simplex on Minkowski difference

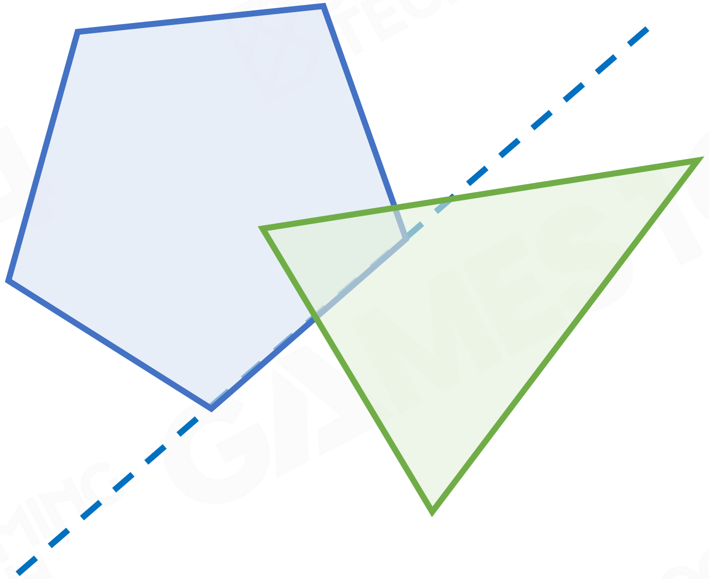


Simplex Set: {C, D} E}

Separating Axis Theorem (SAT) - Convexity

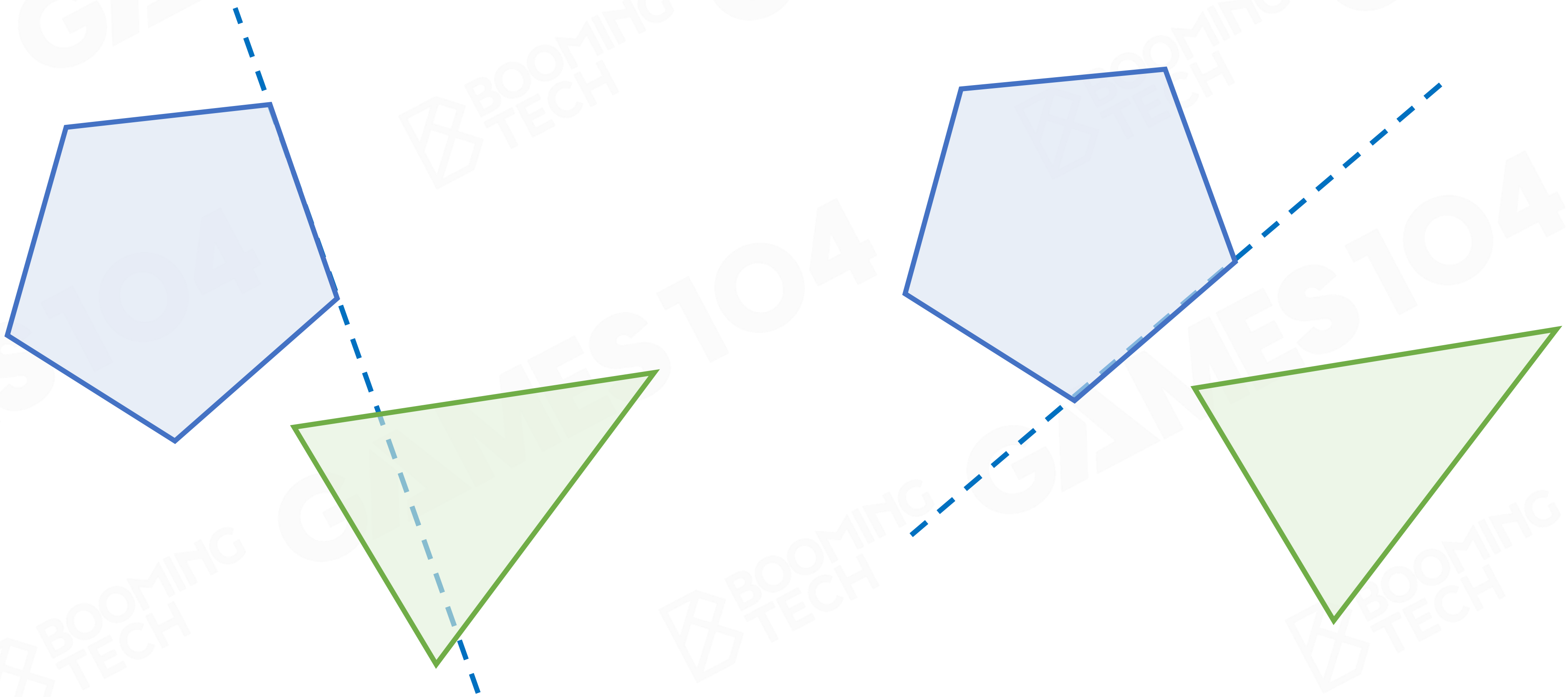
- Edges can separate two convex polygons due to convexity





Separating Axis Theorem (SAT) – Necessity for overlapping

- An edges failed to separate the polygons is not sufficient for overlapping
- All edges must be checked until a separating axis is found



Separating Axis Theorem (SAT) - Separating Criteria

$$d = \vec{n} \cdot (\vec{s} - \vec{p})$$

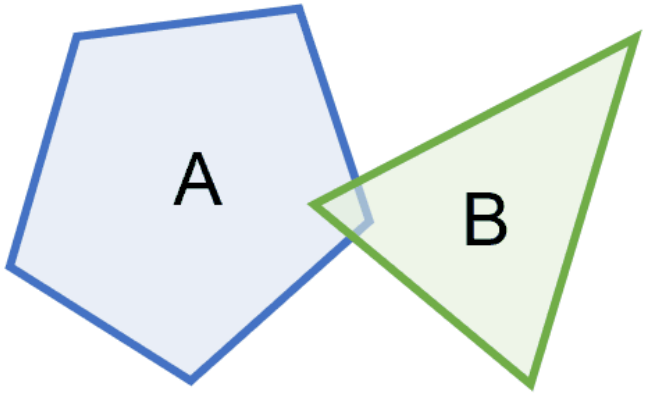
$d > 0$

$d \leq 0$

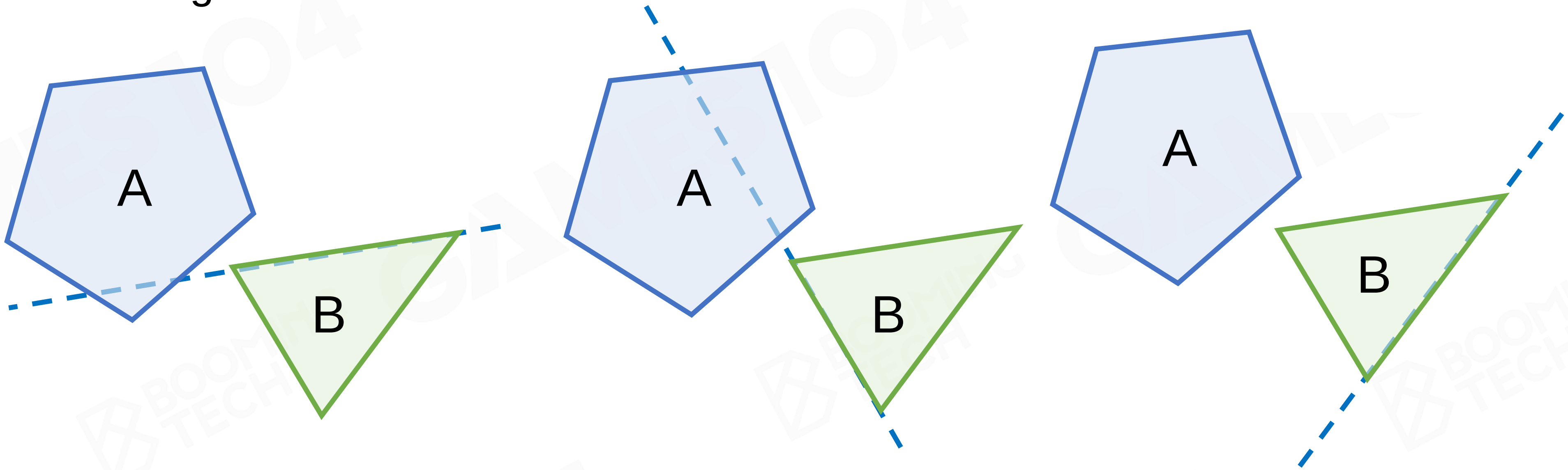
Penetration depth is $|d|$

Separating Axis Theorem (SAT) – 2D Case (1/2)

- Check edges from A and vertices from B
- Check vertices from A and edges from B



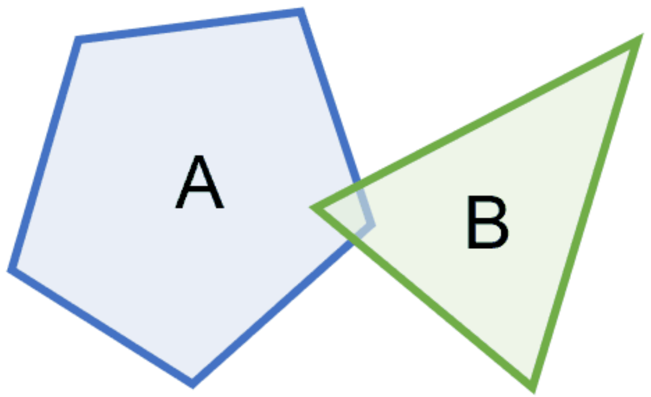
All edges from B failed



Separating Axis Theorem (SAT) – 2D Case (2/2)

Algorithm 1 SAT-2D

for each edge e_A from A **do**
  overlapped $\Leftarrow$ false
  for each vertex v_B from B **do**
    if projection of v_B on normal of $e_A \leq 0$ **then**
      overlapped $\Leftarrow$ true, break
    end if
  end for
  if not overlapped **then**
    A and B are separated, terminate
  end if
end for
for each edge e_B from B **do**
  overlapped $\Leftarrow$ false
  for each vertex v_A from A **do**
    if projection of v_A on normal of $e_B \leq 0$ **then**
      overlapped $\Leftarrow$ true, break
    end if
  end for
  if not overlapped **then**
    A and B are separated, terminate
  end if
end for
A and B are overlapped, terminate



Separating Axis Theorem (SAT) – Optimization for 2D Case

- Check edges from A and vertices from B
- Check vertices from A and edges from B

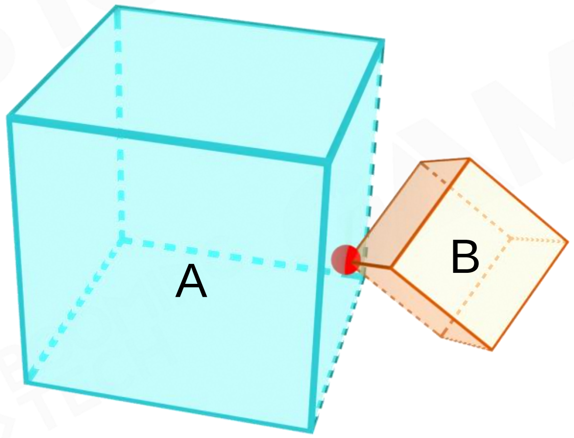
Optimization

- Cache the last separating axis

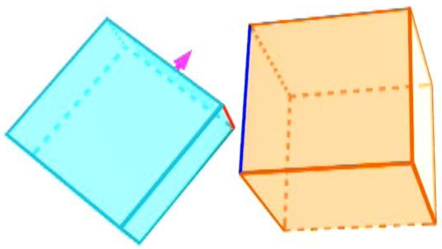
Algorithm 2 SAT-2D-Optimized

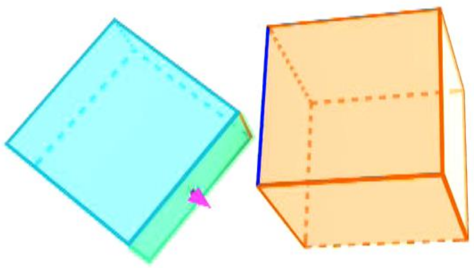
```
overlapped ⇐ false
for each vertex v_B from B do
    if projection of v_B on separating_axis_A ≤ 0 then
        overlapped ⇐ true, break
    end if
end for
if not overlapped then
    A and B are separated, terminate
end if
for each edge e_A from A do
    overlapped ⇐ false
    for each vertex v_B from B do
        if projection of v_B on normal of e_A ≤ 0 then
            overlapped ⇐ true, break
        end if
    end for
    if not overlapped then
        update separating_axis_A ⇐ normal of e_A
        A and B are separated, terminate
    end if
end for
Similar for edges from B
...
```

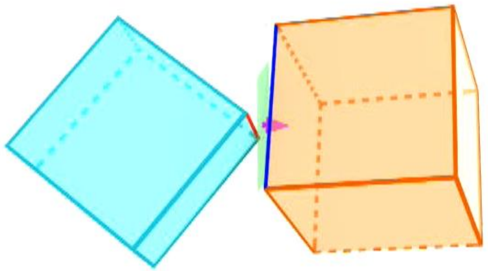
Separating Axis Theorem (SAT) – 3D Case



- Check faces from A and vertices from B
  - Separating axis: face normals of A
- Check vertices from A and faces from B
  - Separating axis: face normals of B
- Check edges from A and edges from B
  - Separating axis: cross product of two edges



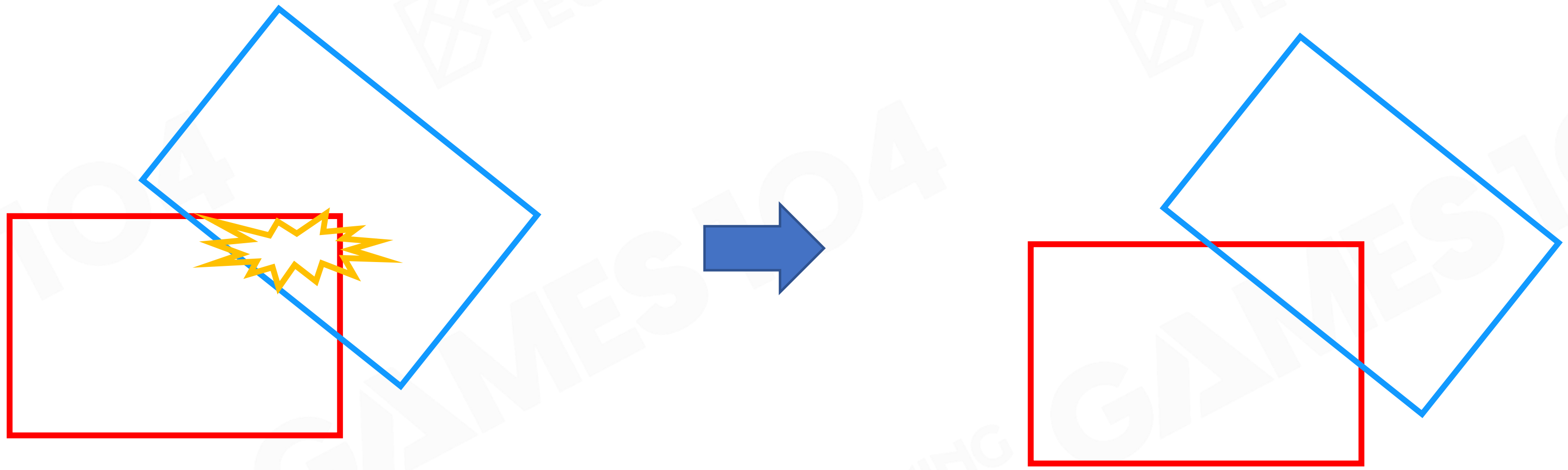




Collision Resolution

Collision Resolution

- We have determined collisions precisely
- We have obtained collision information
- Next, let’s deal with collision resolution

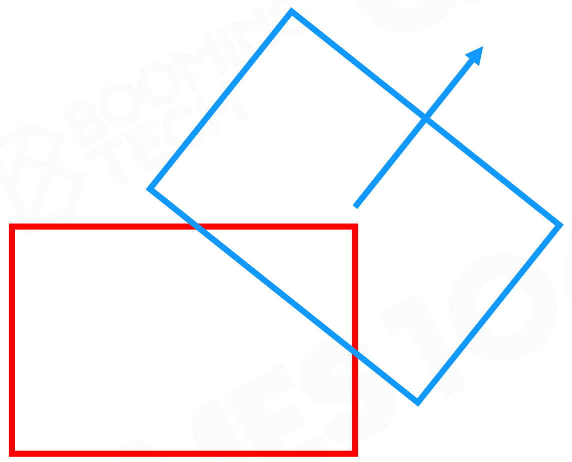


Approaches

- Three approaches
  - Applying Penalty Force
  - Solving Velocity Constraints
  - Solving Position Constraints (will be covered in the next lecture)

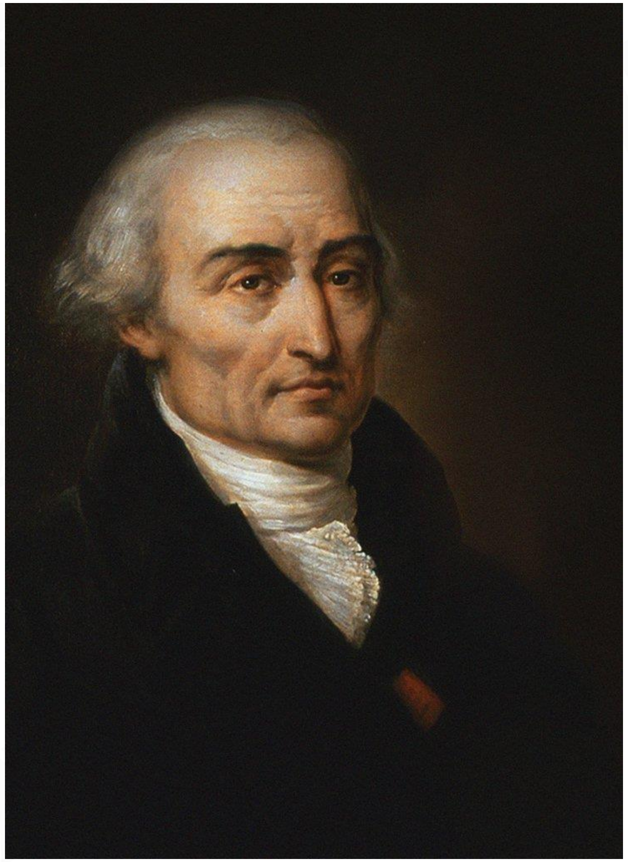
Applying Penalty Force

- Rarely used in games
- Large forces and small time steps are needed to make colliding actors look rigid



Solving Constraints (1/2)

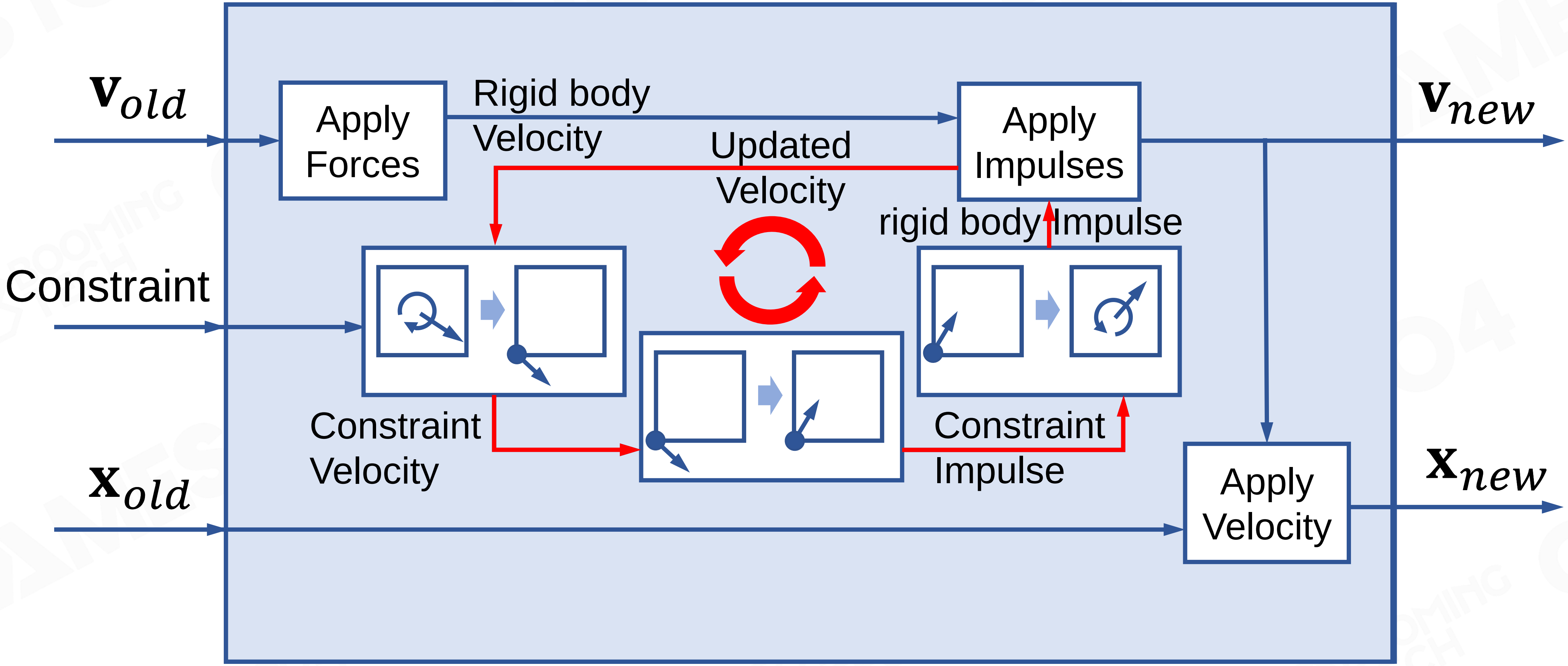
- Modelling constraints based on Lagrangian mechanics
  - Collision constraints
    - Non-penetration
    - Restitution
    - Friction
- Iterative solver



Joseph-Louis Lagrange

(1736 - 1813)

Solving constraints (2/2)



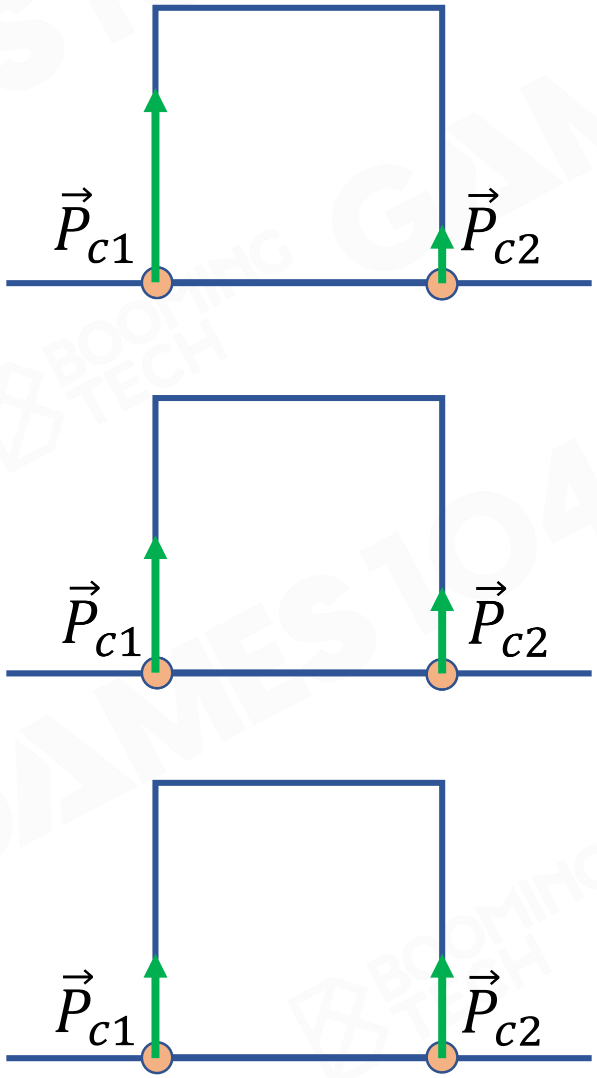
Solving Velocity Constraints

Approaches:

- Sequential impulses
- Semi-implicit integration
- Non-linear Gauss-Seidel Method

Characteristics:

- Fast, stable for most cases
- Commonly used in most physics engines



Scene Query

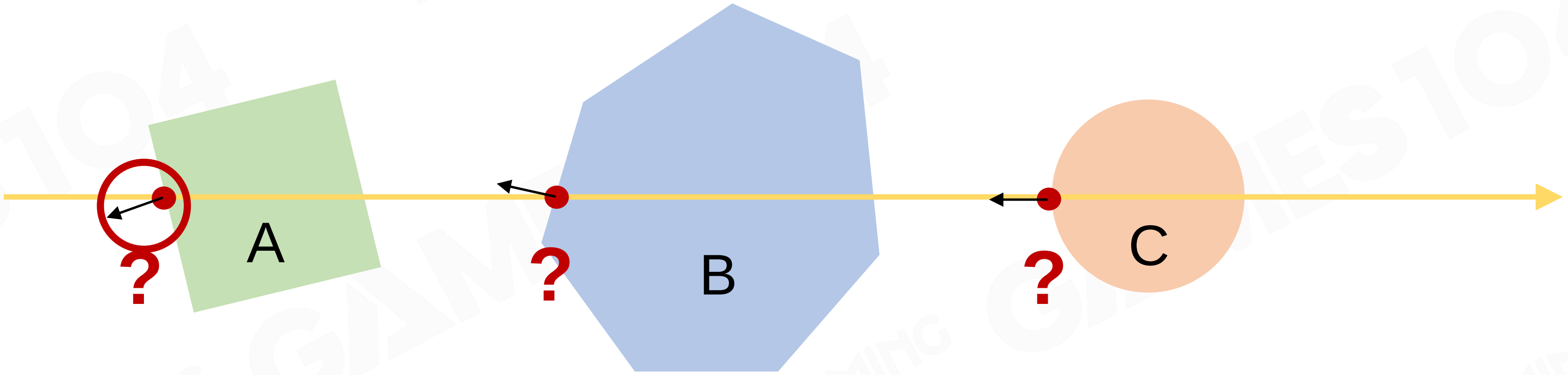
Raycast (1/3)

- Intersect a user-defined ray with the whole scene
- Point, direction, distance and query mode can be defined

Raycast (2/3)

Raycast (3/3)

- Mutiple hits — looks for all blocking hits, picks the one with the minimum distance
- Closest hit — looks for all blocking hits
- Any hit — any hit encountered will do



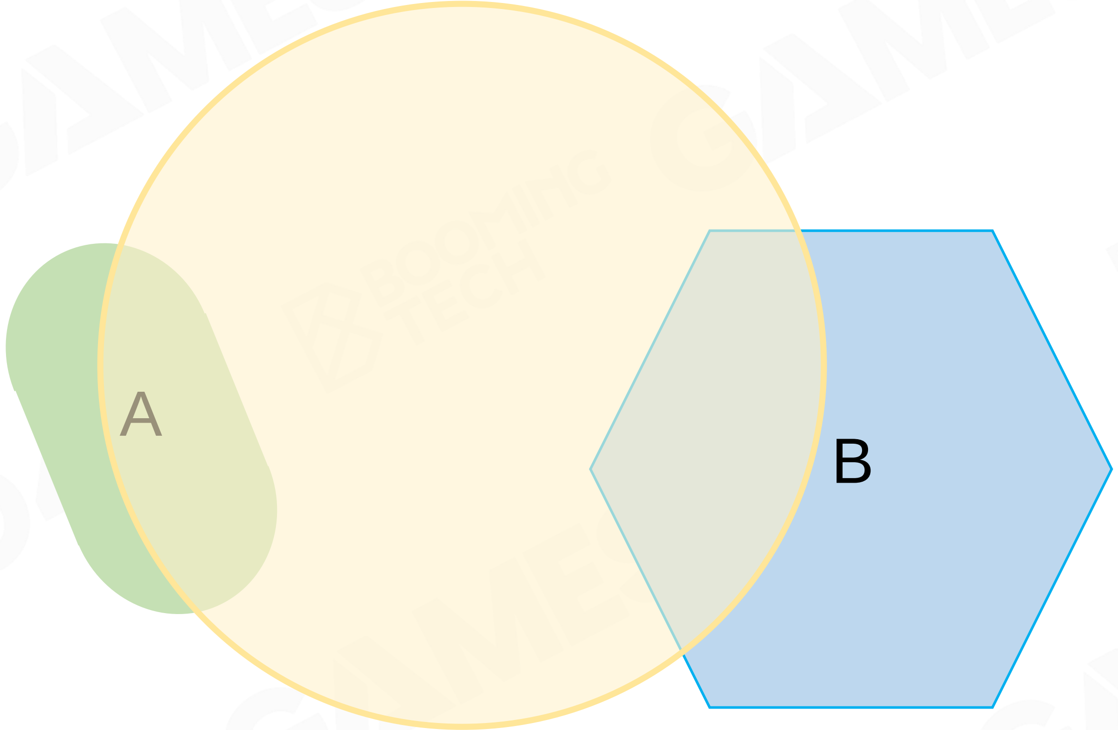
Sweep (1/2)

- Geometrically similar to raycast
- Shape and pose can be defined
- Box, sphere, capsule and convex

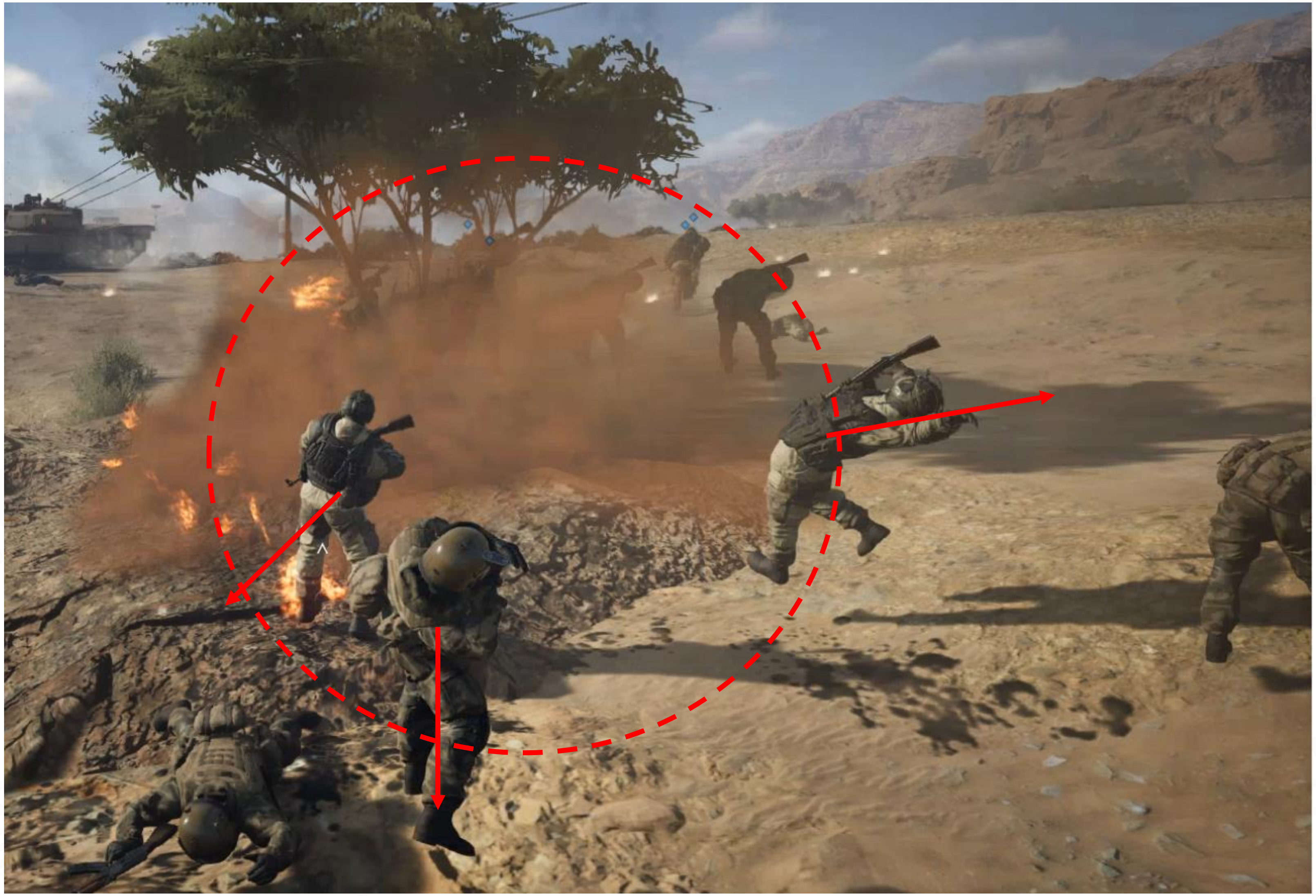
Sweep (2/2)

Overlap (1/2)

- Search a region enclosed by a specified shape for any overlapping objects in the scene
- Box, sphere, capsule and convex



Overlap (2/2)



Collision Group

- Actor has a collision group property

  Player : Pawn

  Obstacle : Static

  Movable box : Dynamic

  Trigger box : Trigger

  …

- Scene query can filter collision groups

  Player moving query collision group:

  (Pawn, Static, Dynamic)

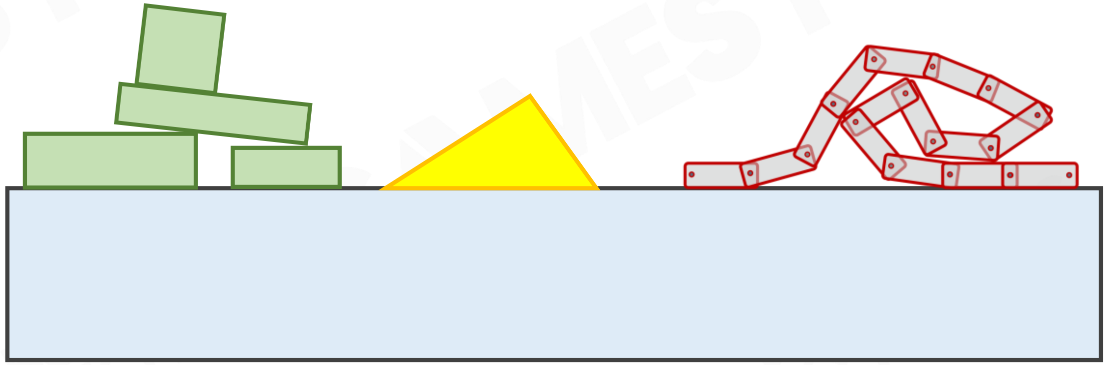
  Trigger query collision group:

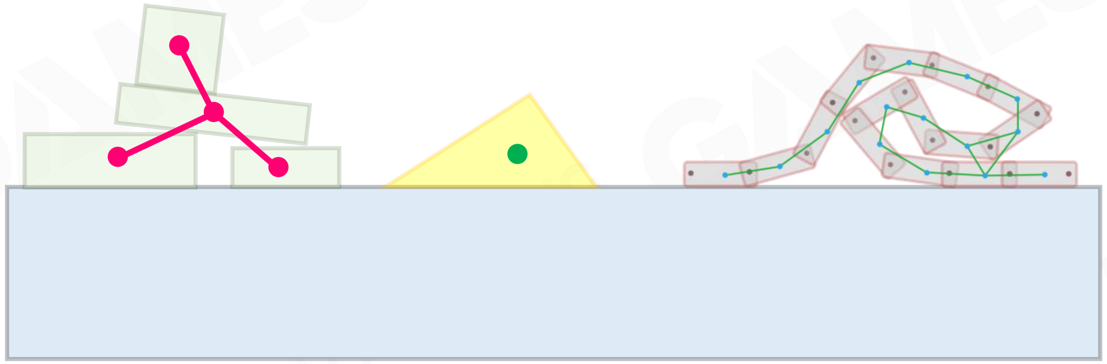
  (Pawn)

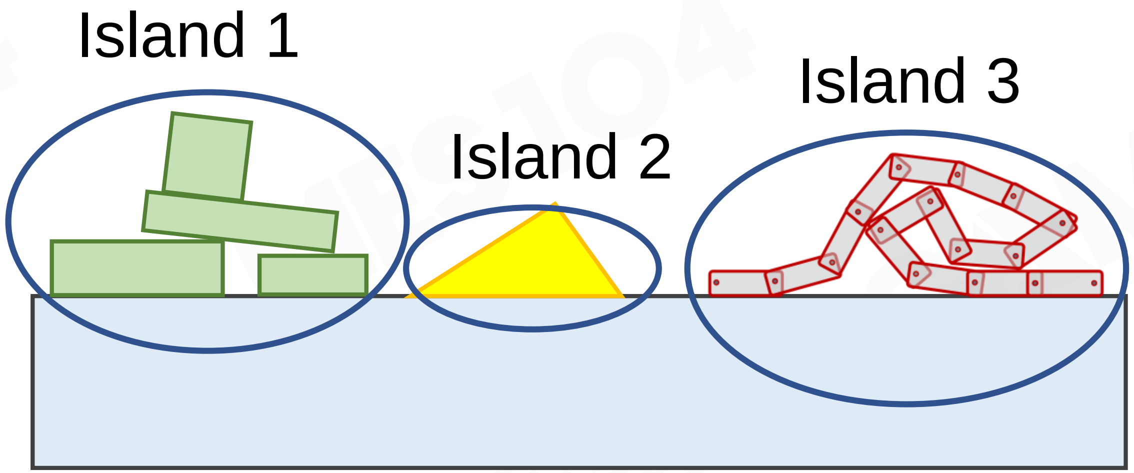
  …

Efficiency, Accuracy, and Determinism

Simulation Optimization – Island

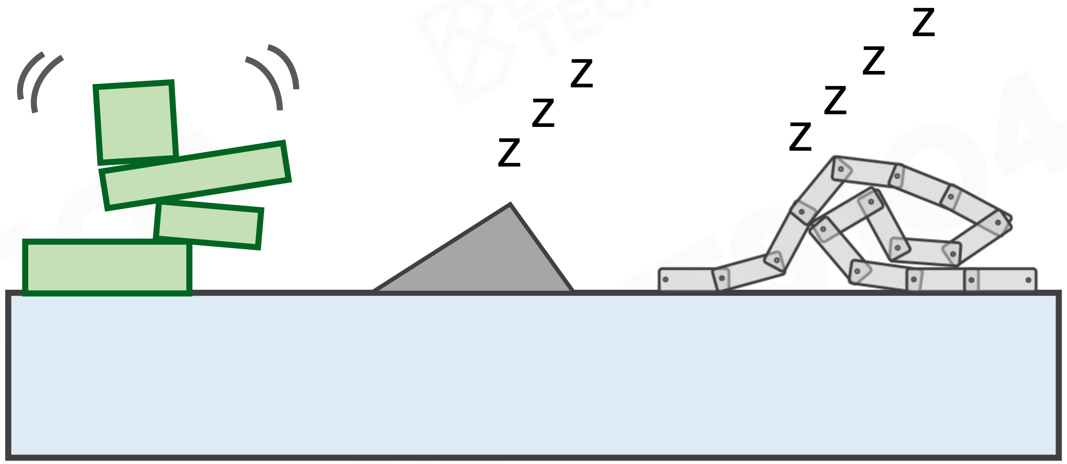






Simulation Optimization – Sleeping

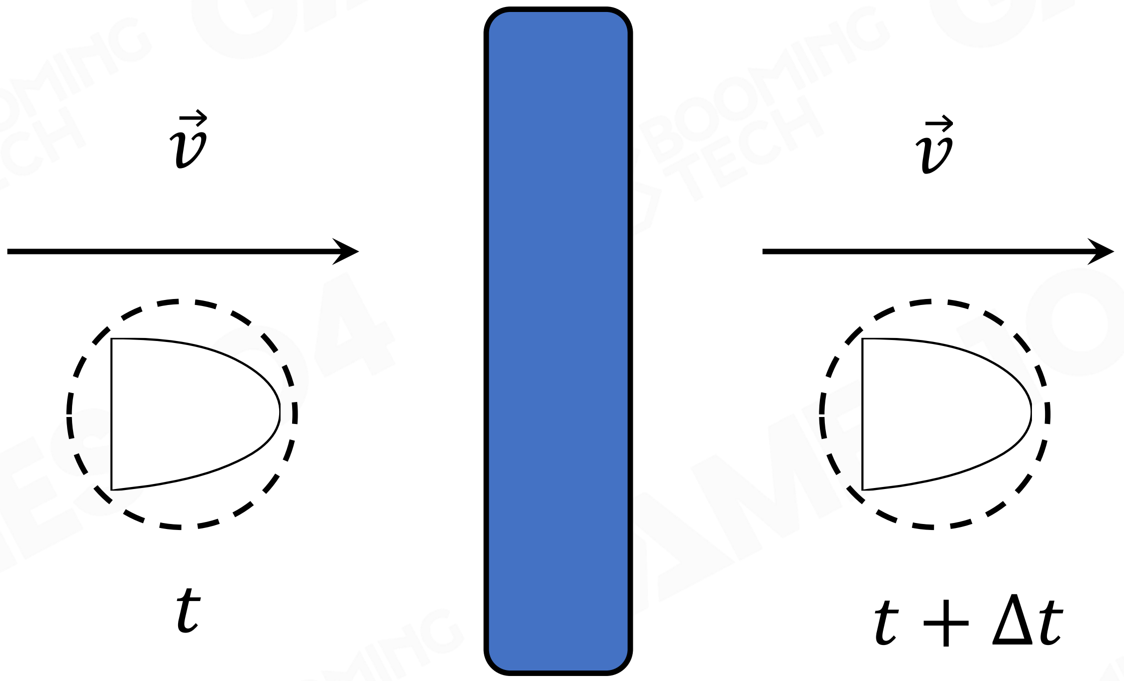
- Simulating and solving all rigid bodies uses lots of resources
- Introducing sleeping
  - A rigid body does not move for a period of time
  - Until some external force acts on it



Continuous Collision Detection (1/4)

Continuous Collision Detection (2/4)

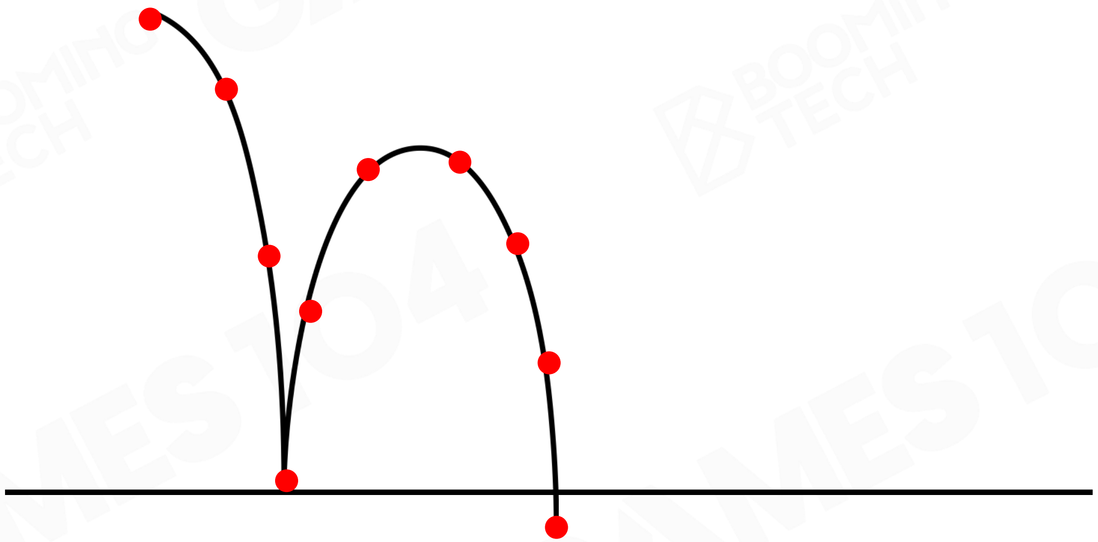
- Thin obstacle vs. fast moving actors
- Tunneling



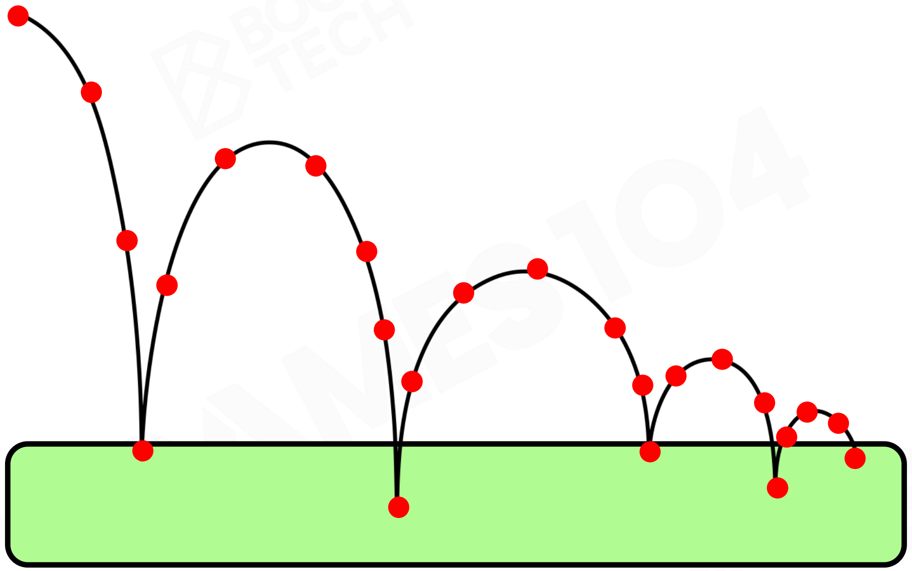
Continuous Collision Detection (3/4)

- Solution to tunnelling

Let it be – some thing unremarkable

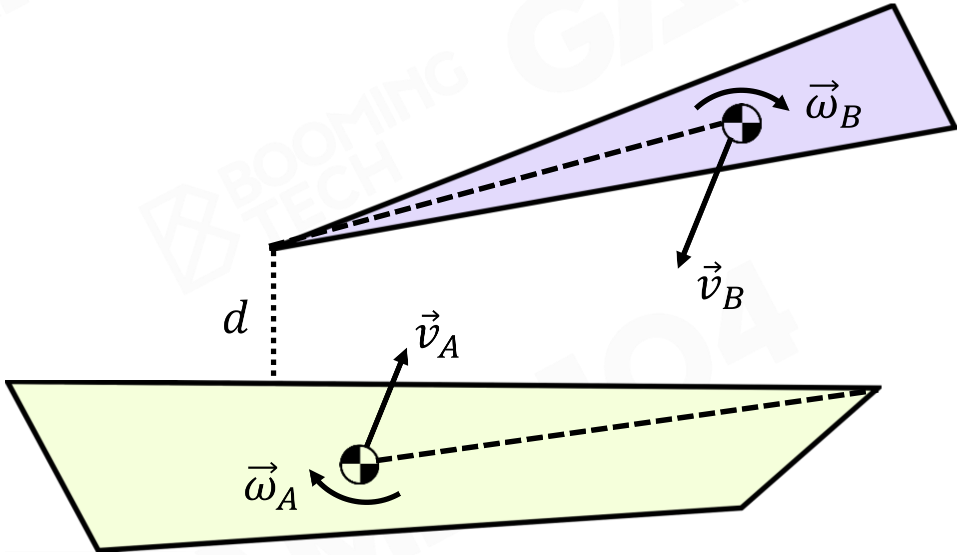


Make the floor thicker – boundary air wall



Continuous Collision Detection (4/4)

- Time-of-Impact (TOI) – Conservative advancement
  - Estimate a “safe” time substep A and B won’t collide
  - Advance A and B by the “safe” substep
  - Repeat until the distance is below a threshold



Deterministic Simulation (1/4)

- Multiplayer game with gameplay-impacting physics
- Small error causes butterfly effect
- Synchronizing states requires bandwidth
- Synchronizing inputs requires deterministic simulations

Deterministic Simulation (2/4)

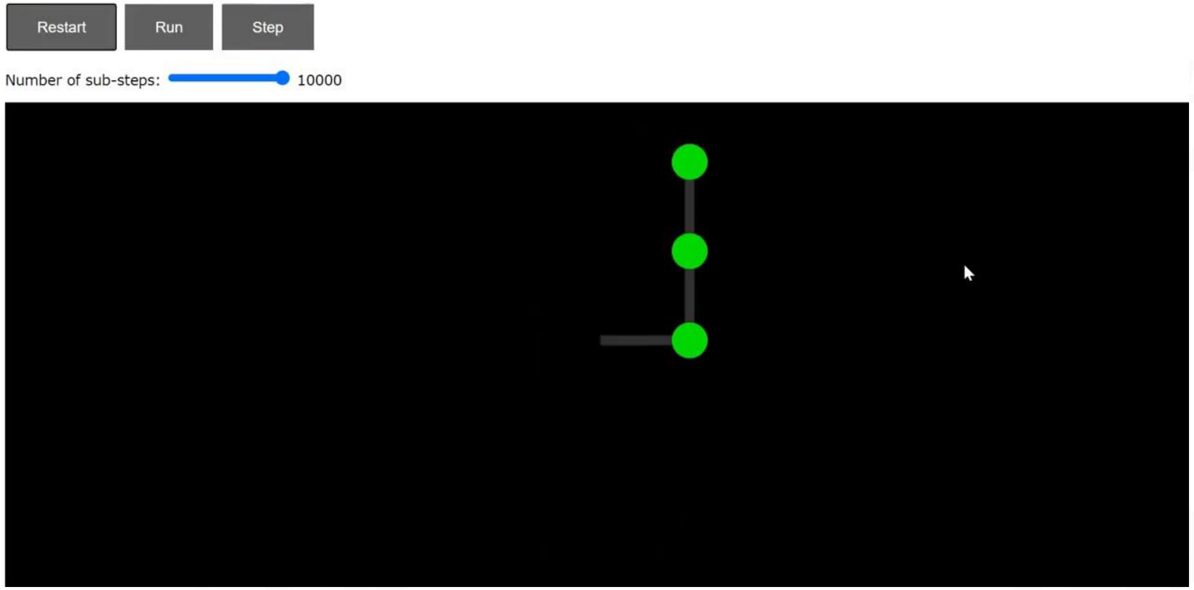
Non-deterministic Simulation

Deterministic Simulation (3/4)

Same old states + same inputs = same new states

Requirements

- Fixed step of physics simulation
- Deterministic simulation solving sequence
- Float point consistency



Deterministic Simulation (4/4)

Deterministic Simulation

Physics is Not Easy

Lecture 10 Contributor

- 一将
- 灰灰
- 新之助
- BOOK
- Wood
- 爵爷
- 乐酱
- 大喷
- Qiuu
- Adam
- Olorin
- 喵小君
- 呆呆兽
- 蒙蒙
- 人工非智能
- Hoya
- 达拉崩吧
- 蓑笠翁
- 晨晨
- Kun

Q&A

Enjoy ;) Coding

Course Wechat

Follow us for further information