Voice from Community

- Where to ask questions?
  - GitHub discussion is the best channel
  - WeChat group
- Will Mini Engine integrate a physics engine?
- Need more reference, links of papers and codes, or materials Course Team referred when preparing the PPT slides
- Would like to know the logic behind the topics. Why the Team choose those topics over others?
- Used same engine on different devices running the same code and same scenario of a still object, the outcome of rendering varies?

Pilot Engine V0.0.6 Released – 31 May

New Feature

- Jolt Physics integration

Refactoring

- Rendering
    - swap data context (finished)

Bugfixes

- Fixed bugs in mipmap level of color grading texture
- Fixed bugs in render system when reloading levels

Contributors

BoomingTechDev, hyv1001, and 2 other contributors

Naming of Mini Engine - Piccolo

OneOX Engine含义：与课程名同系列，代表小引擎的起源。虚拟世界由0和1组成，X代表无线可能，意味着小	99	12.9%
Aria含义：旅途中遇到的人随口哼出的曲调，希望所有正在烦恼不敢踏出第一步的人都能勇敢开启自己的旅途。	229	29.7%
摇光 Alkaid含义：摇光，也称为瑶光，英文名：Alkaid、η UMa，中文又名破军星。摇光是北斗七星的最	310	40.3%
☆ Piccolo含义：短笛，意义是短小精悍的笛子也可以奏响美妙的乐章。	350	45.5%
Ceres（克瑞斯）含义：给予大地生机，教授人类耕种。可以让小引擎从星星之火，变得更加强壮。	72	9.4%

Rewarding List： (10)

家子颜	鄙姓金名聪	塞伦斯	蜩 蜩	李同学
王十一	核 桃	Frozen	张 茂	otaku/小许

Lecture 11

Physics System

Applications

WANG XI

GAMES 104

2022

Outline of Physics System

01.

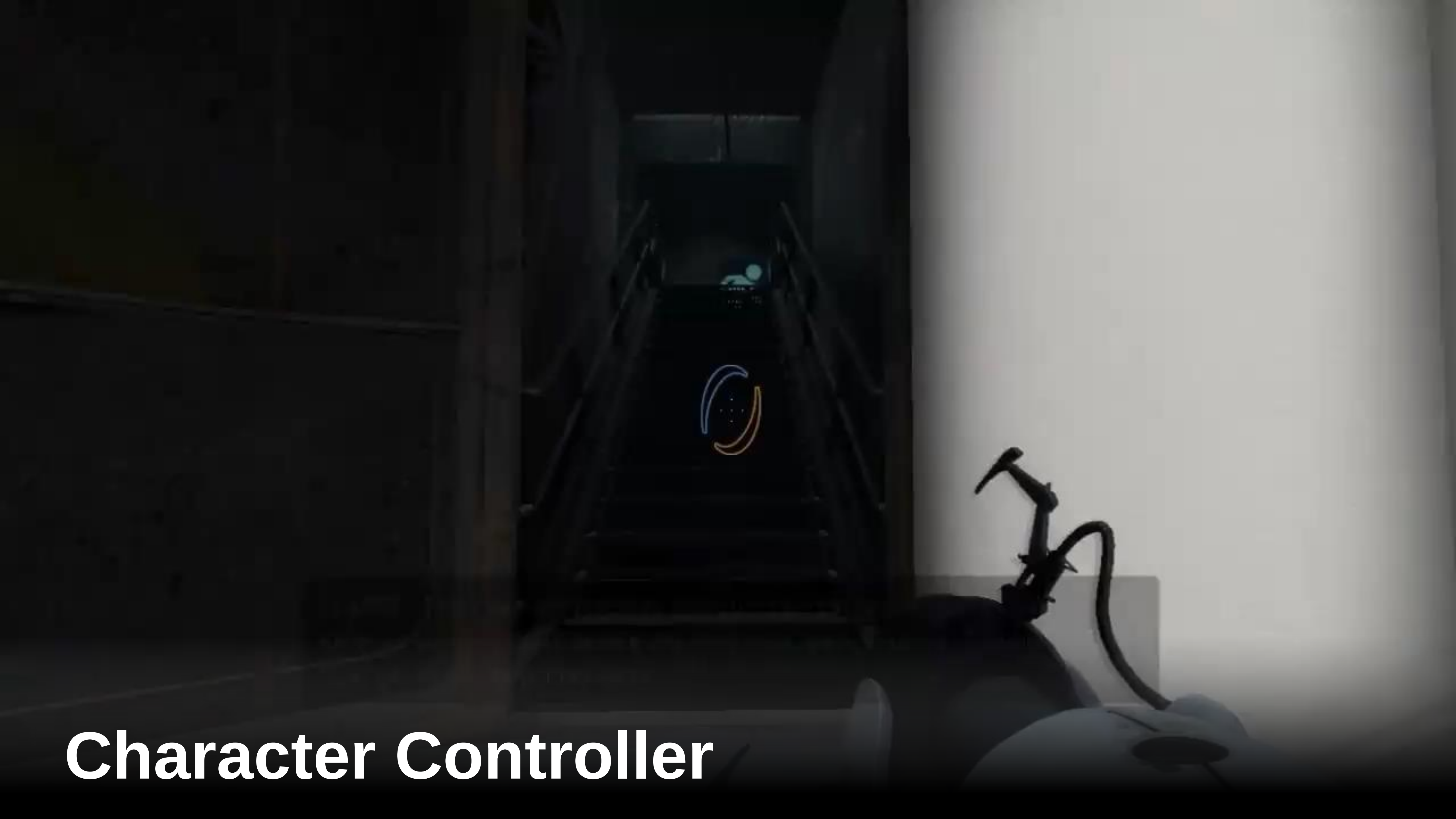
Basic Concepts

- Physics World
- Simulation
- Rigid Body Dynamics
- Collision Detection
- Collision Resolution
- Scene Query
- Miscellaneous

02.

Applications

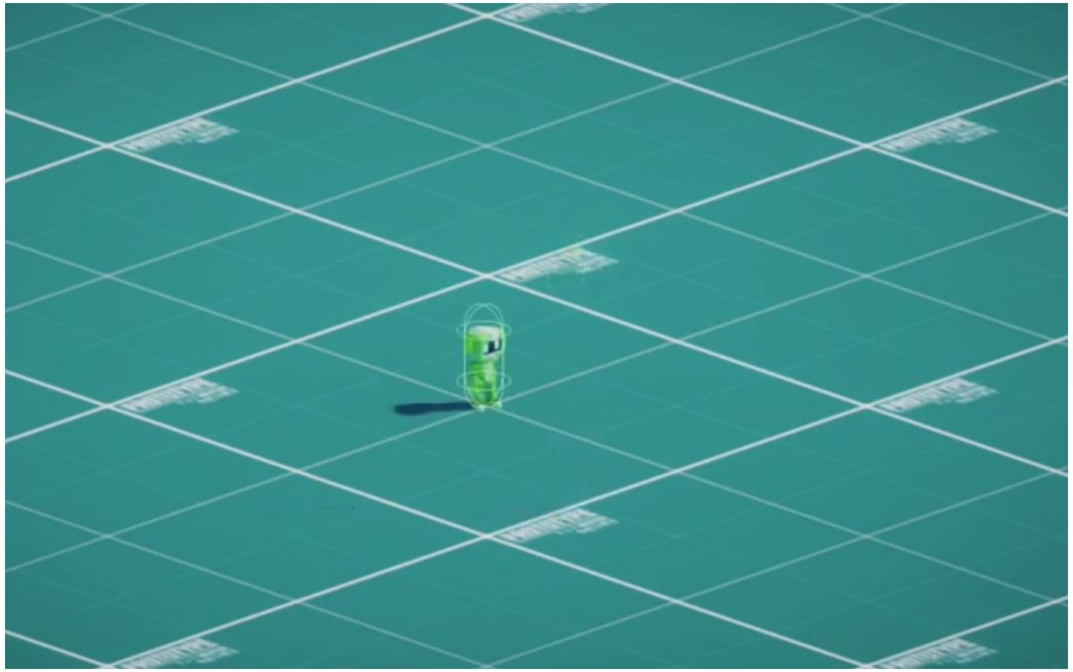
- **Character Controller**
- **Ragdoll**
- **Destruction**
- **Cloth**
- **Vehicle**
- **Advanced: PBD/XPBD**



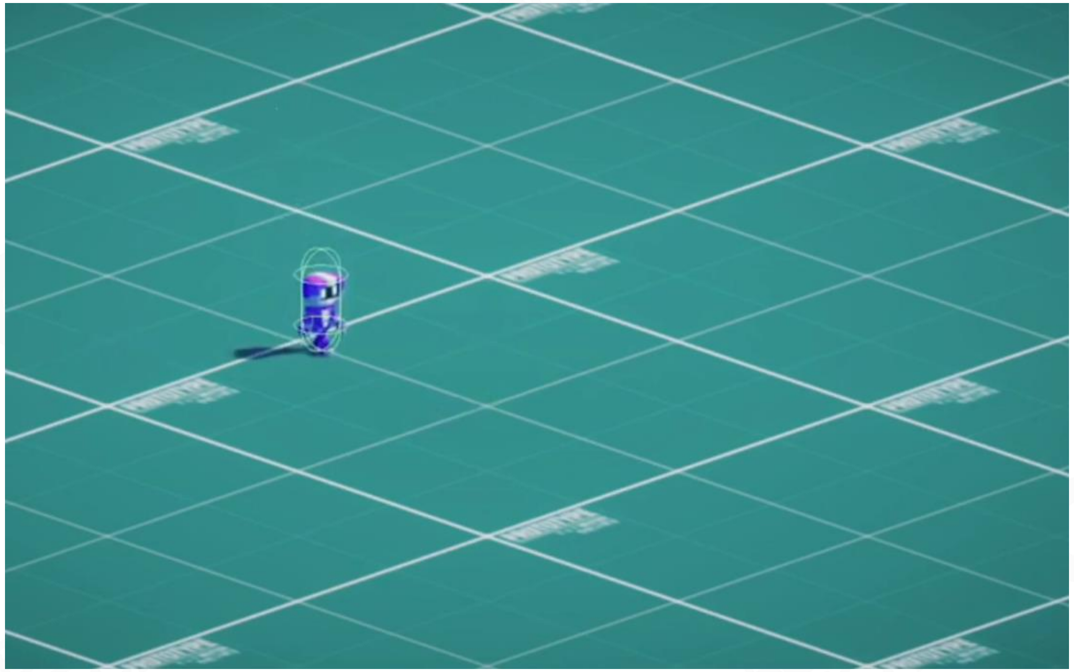
Character Controller

Character Controller vs. Rigid Body Dynamics

- Controllable rigid body interaction
- Almost infinite friction / No restitution
- Accelerate and brake change direction almost instantly and teleport



Character controller



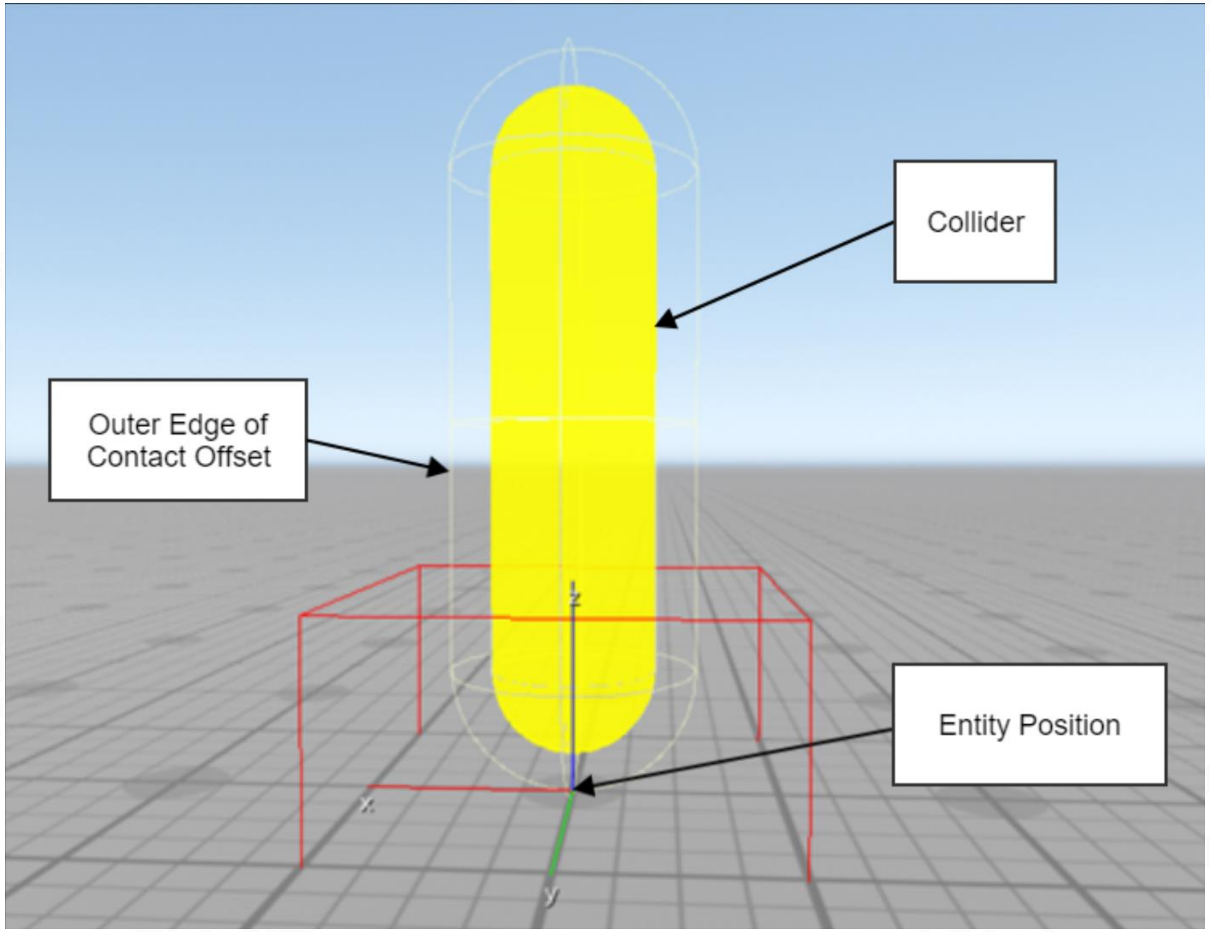
Dynamic actor

Legacy Hack in Character Control

- A lot of carefully tweaked values provide a good feeling
- Legacy code in the industry

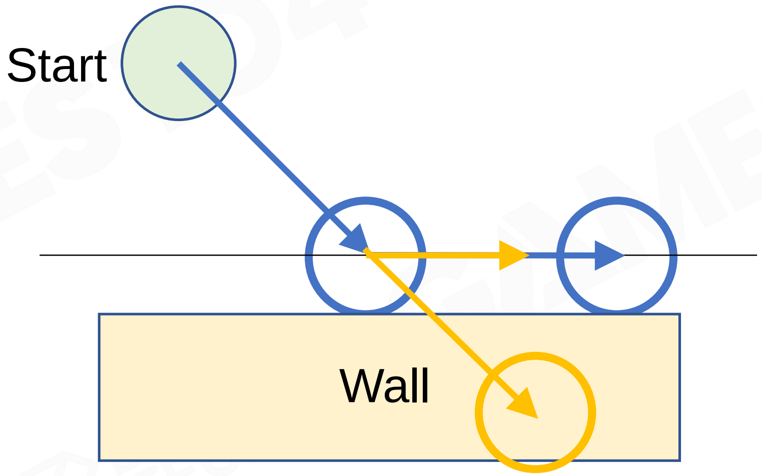
Build a Controller in Physics System

- Kinematic Actor
  - Not affected by physics rules
  - Push other objects
- Shape: Humanoids
  - Capsule
  - Box
  - Convex



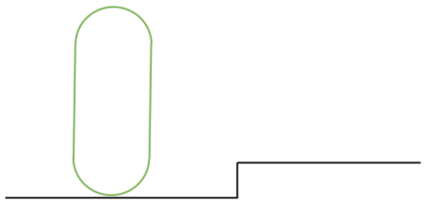
Collide with environment

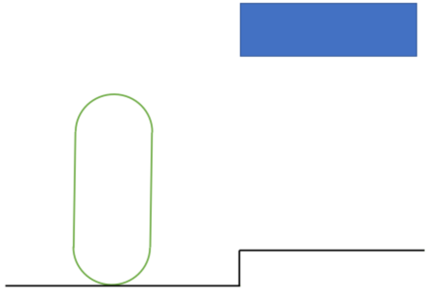
- Collision detection with environment
  - Sweep test
- Auto slide with wall
  - Calculate tangent direction
  - Move along tangent direction

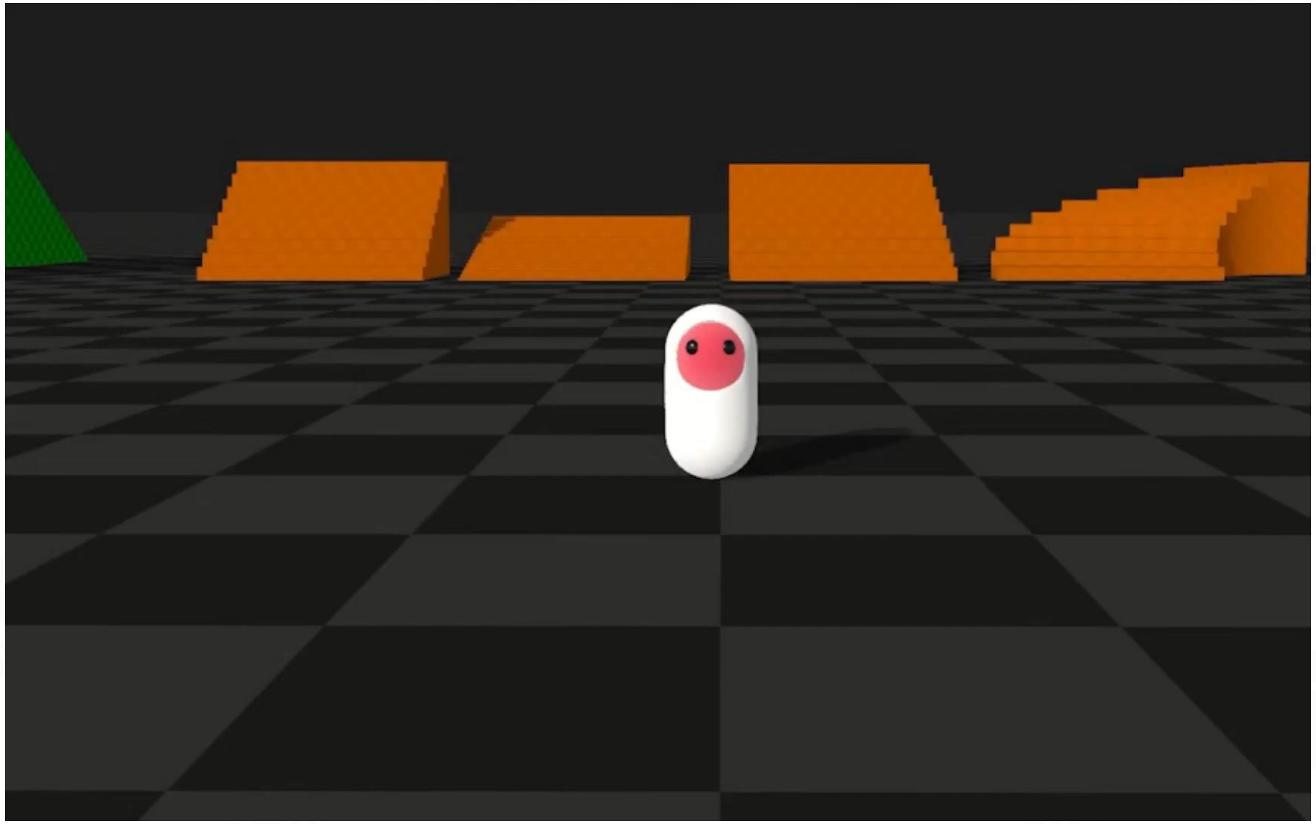


Auto Stepping and its Problem

- Sweep with step offset
- Virtual gap

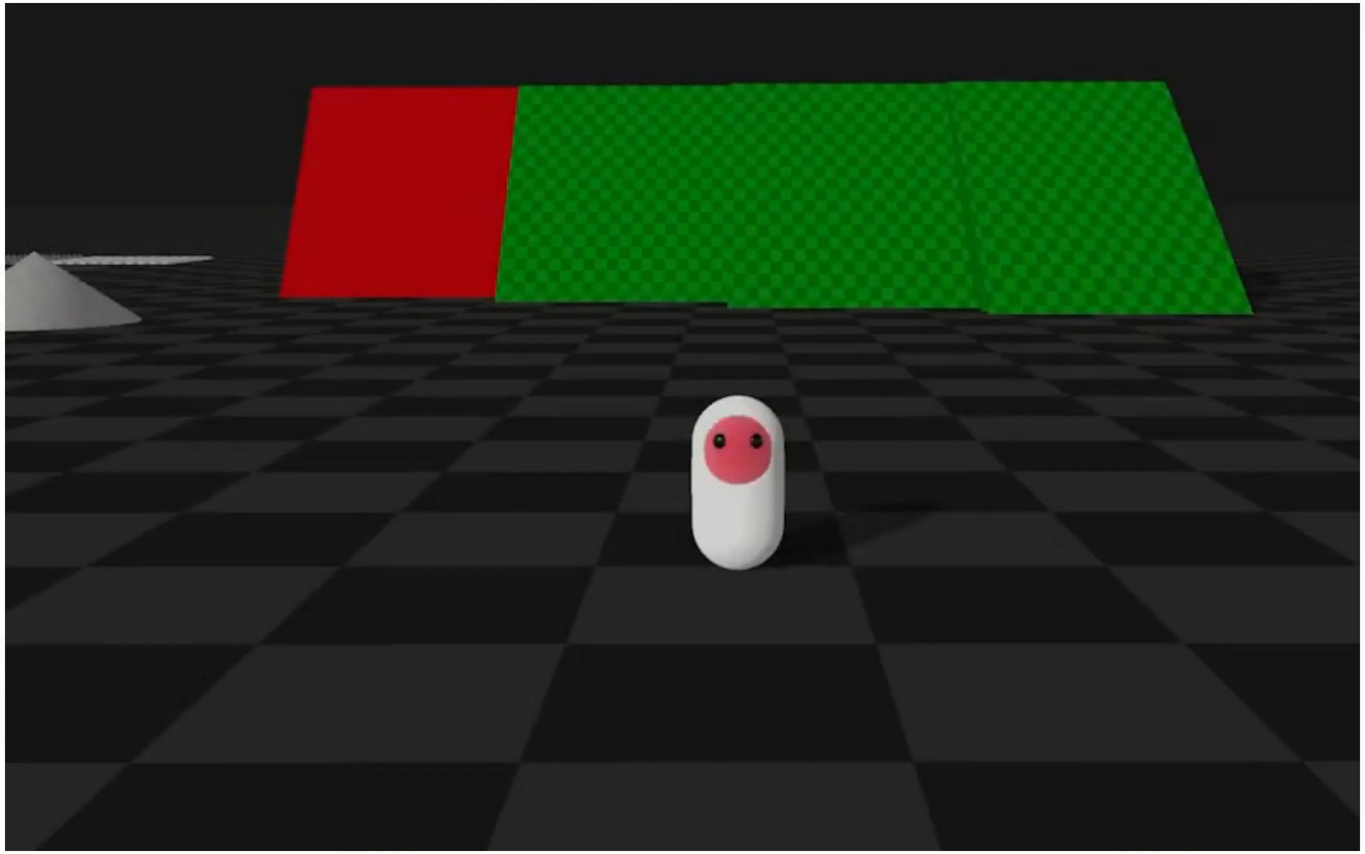






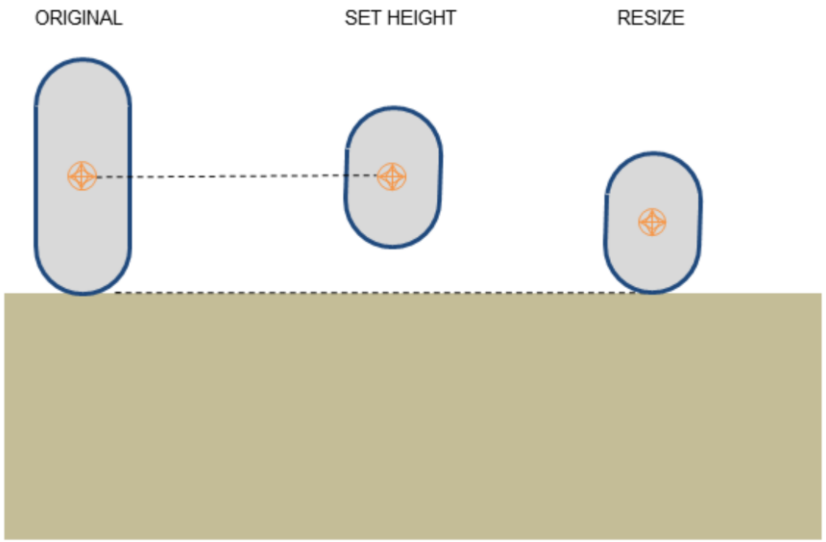
Slope Limits and Force Sliding Down

- Max climb slopes
- Slide down on steep slopes



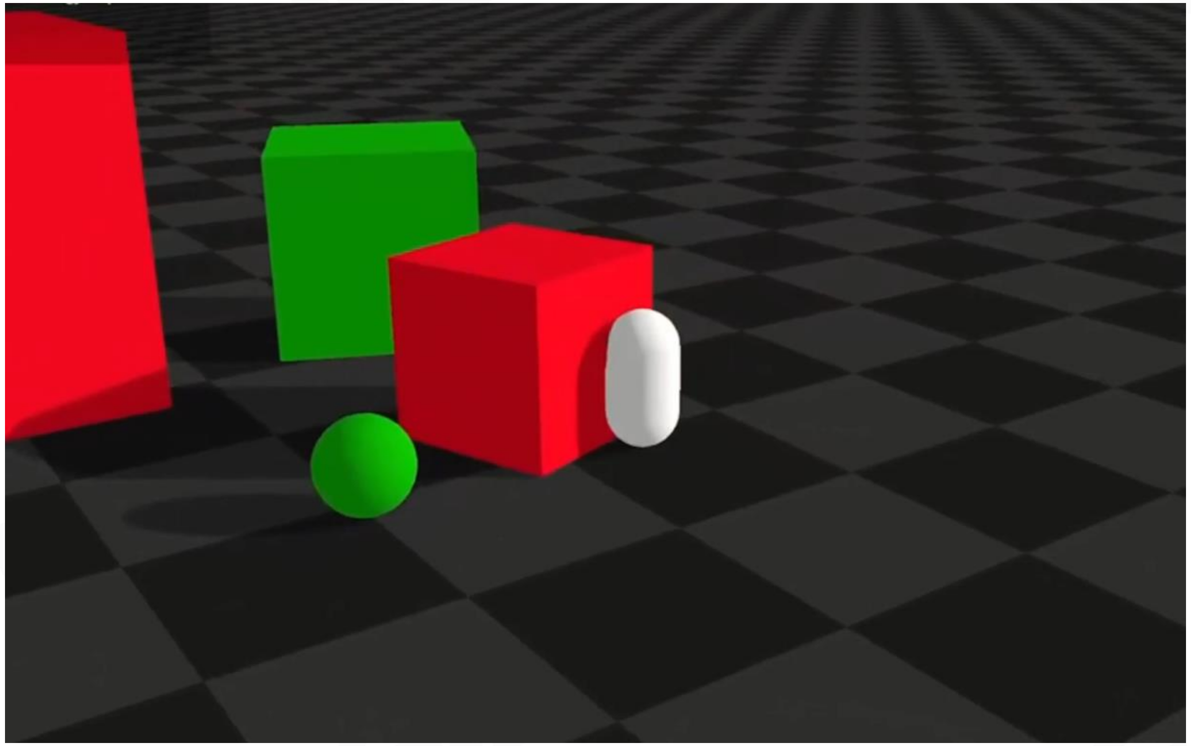
Controller Volume Update

- Change the controller volume size at runtime, e.g. crouching
- **Overlap test before update to avoid insertion inside objects**

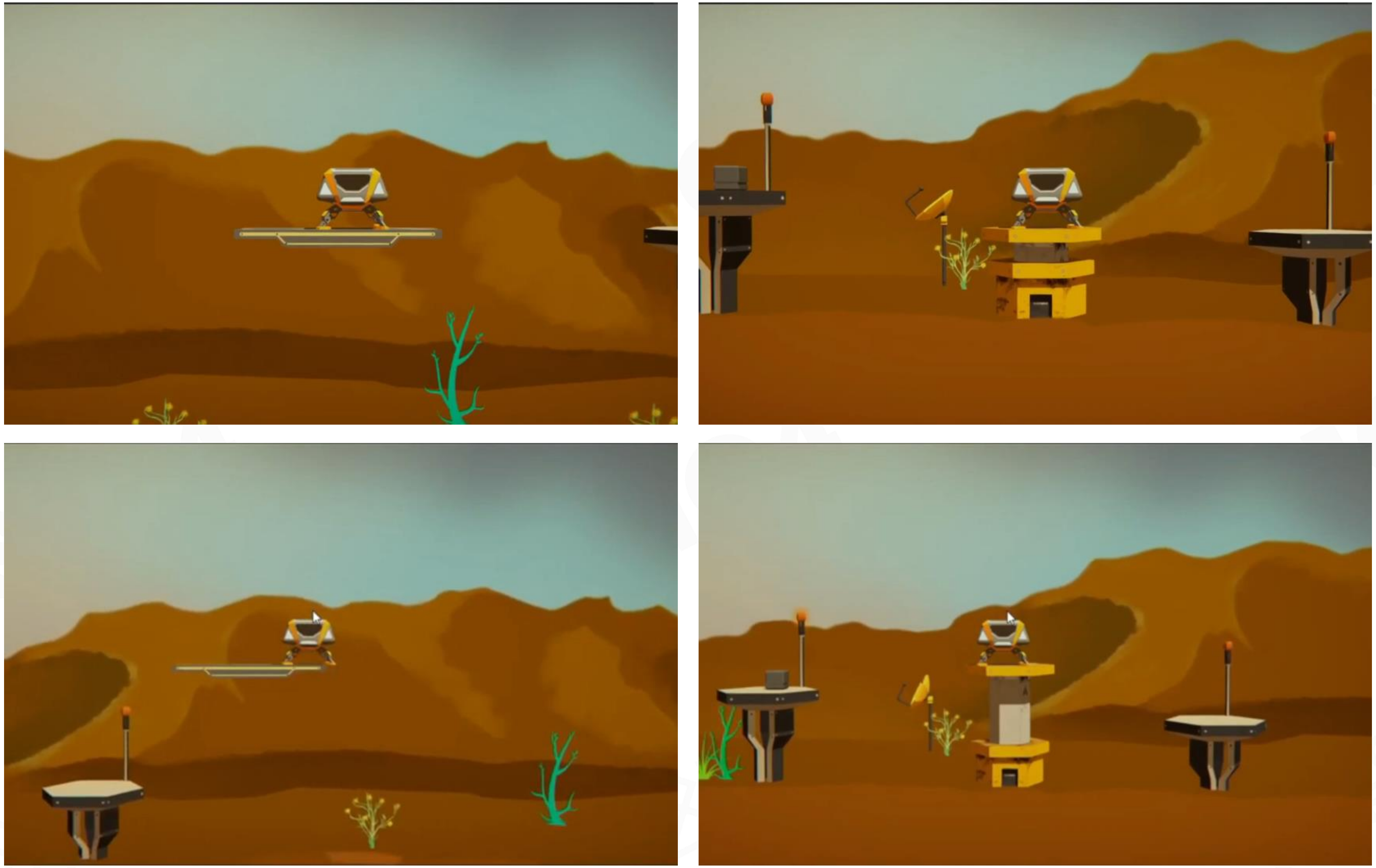


Controller Push Objects

- Hit Callback when character controller collides with dynamic actor
- Apply force to dynamic actor



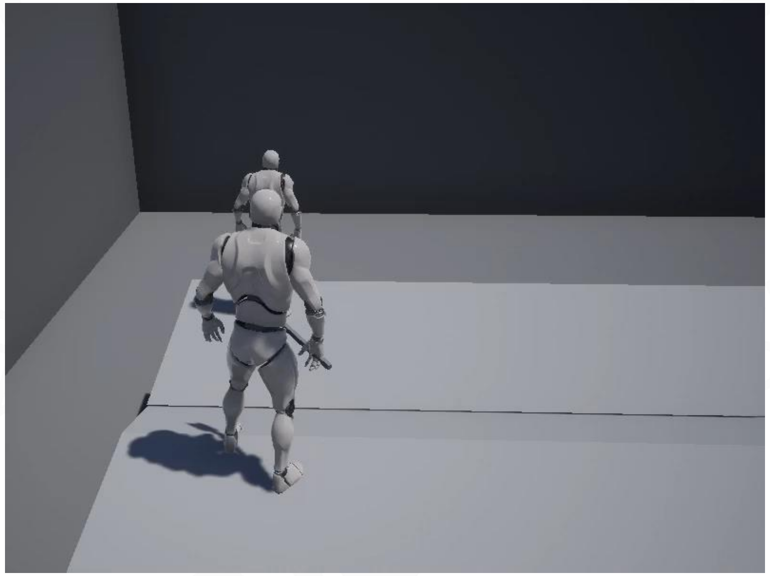
Standing on Moving Platform



Ragdoll

Why Should We Use Ragdoll

Die on the ground

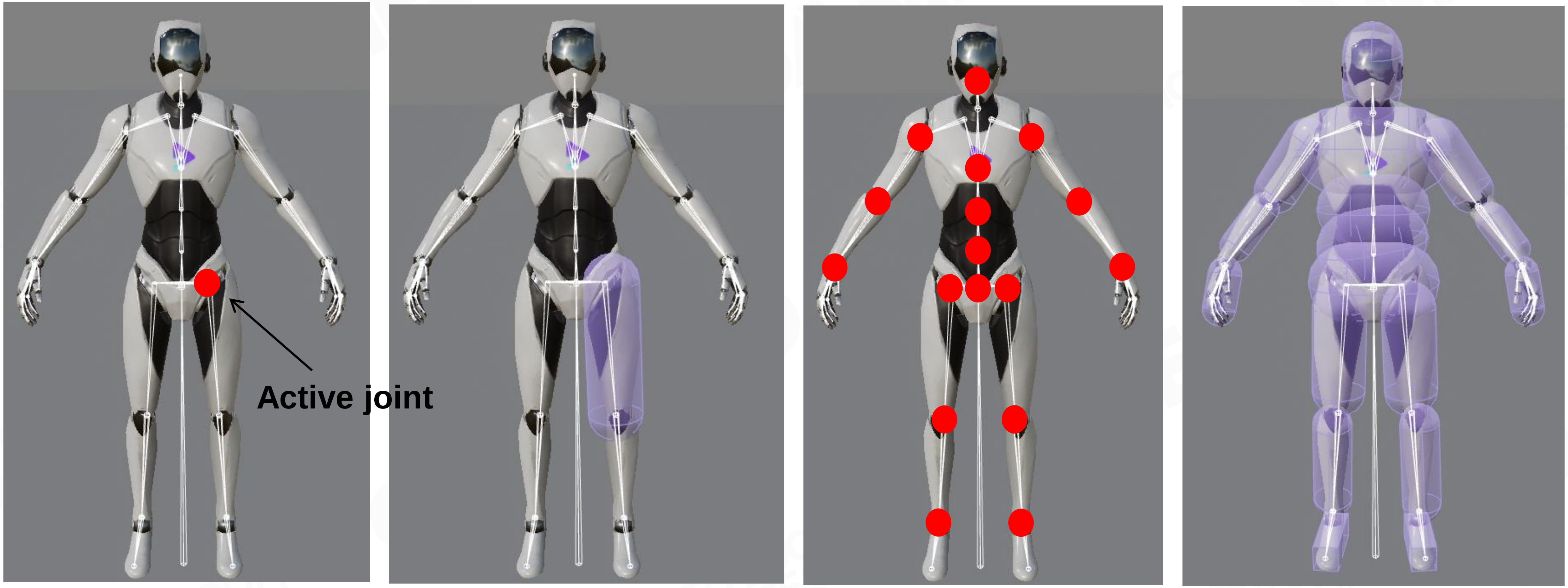


Die on the edge of a cliff

Enable physics

Map Skeleton to Rigid Bodies

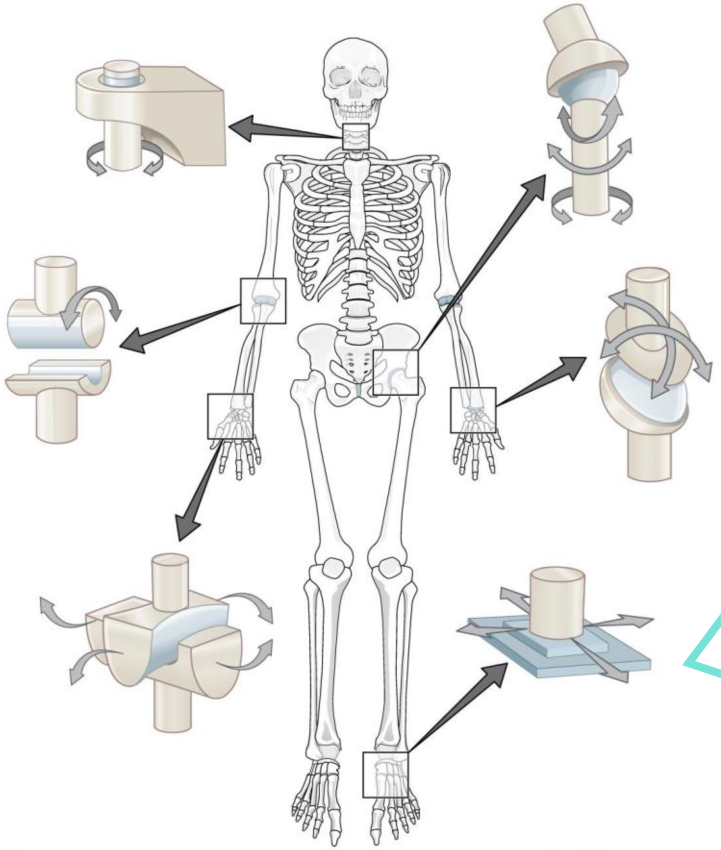
Bind key joints with rigid bodies



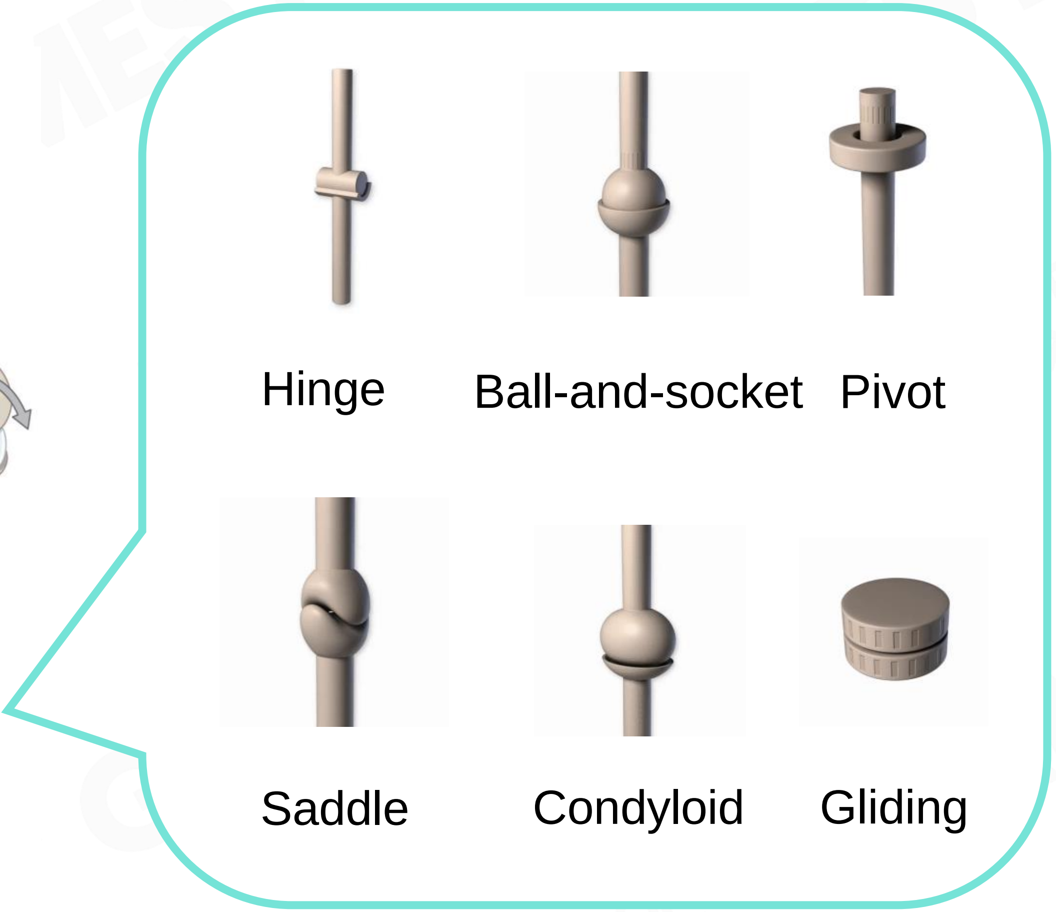
Human Joint Constraints

Various constraints

- Ball-and-socket
- Hinge
- Pivot
- Condyloid
- Saddle
- Gliding
- ...

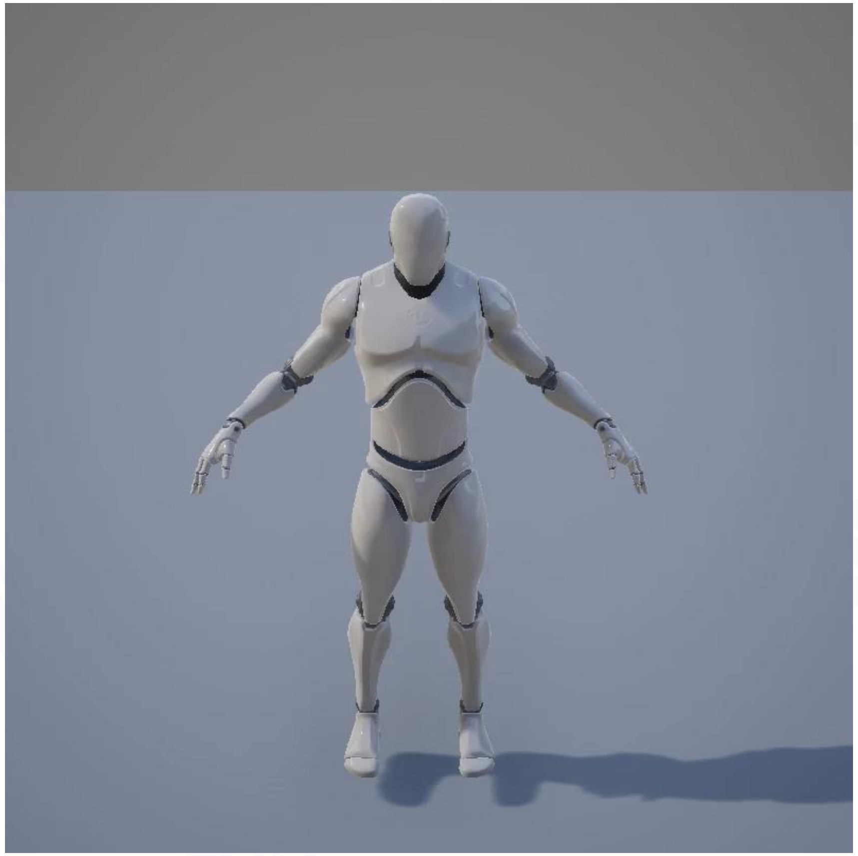


Constraints of Human Skeleton



Importance of Joint Constraints

The constraints should match the anatomical skeleton



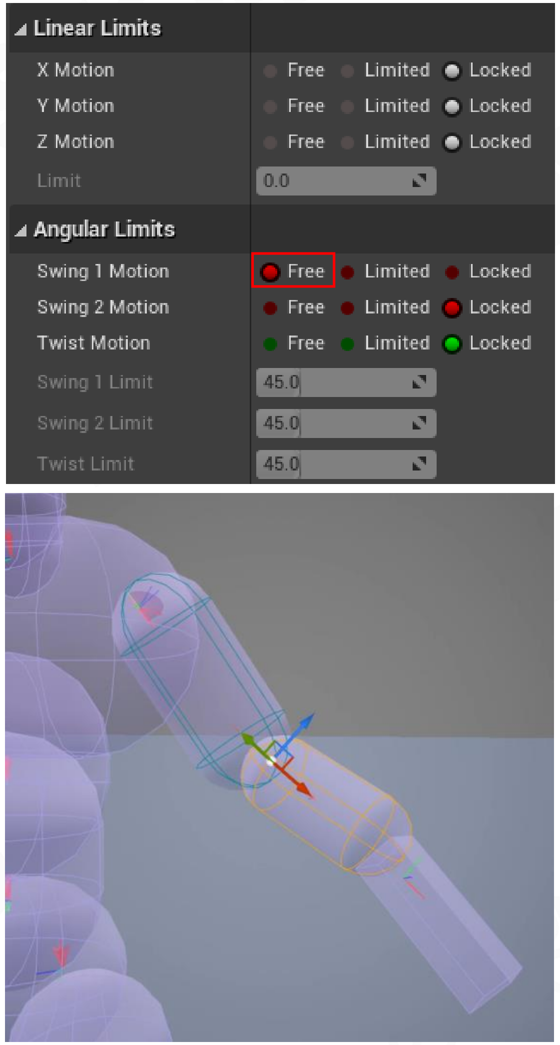
Result with correct constraints

Weird result with free hinges only

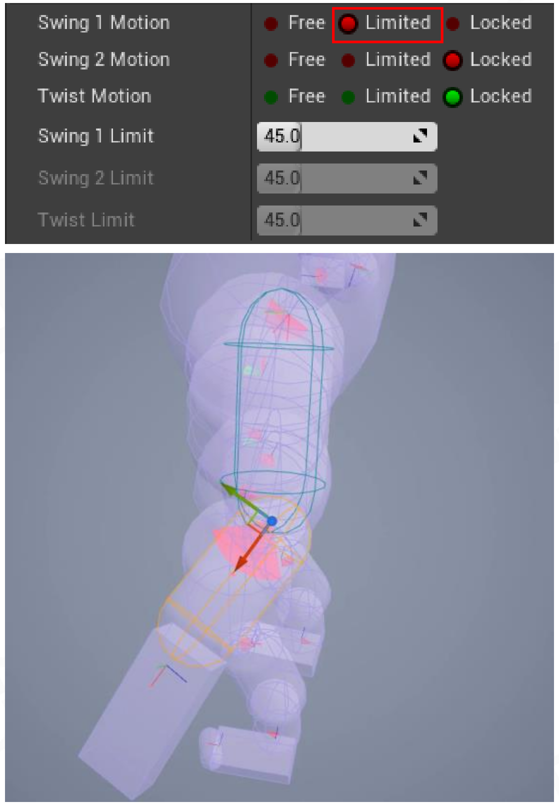
Constraints – Properties

Case: Hinge Constraint

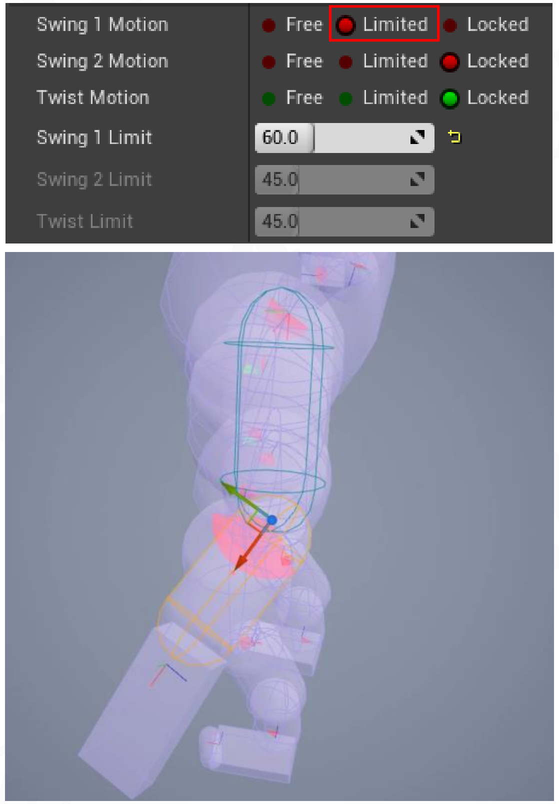
Hinge



Free Swing



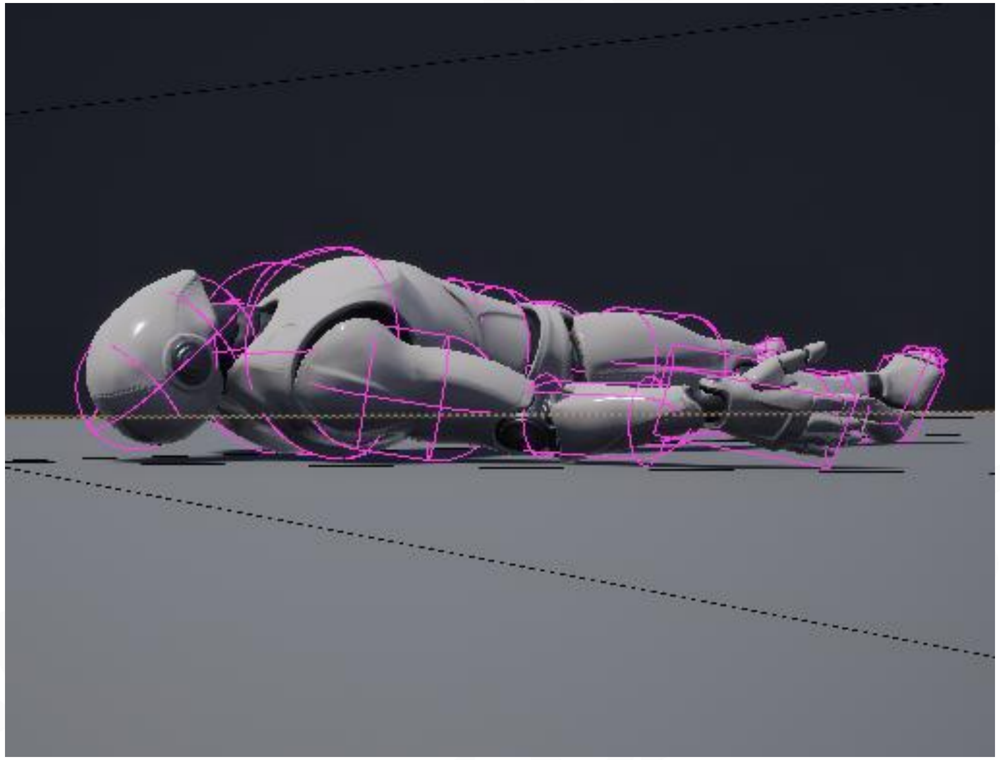
45° Limited Swing



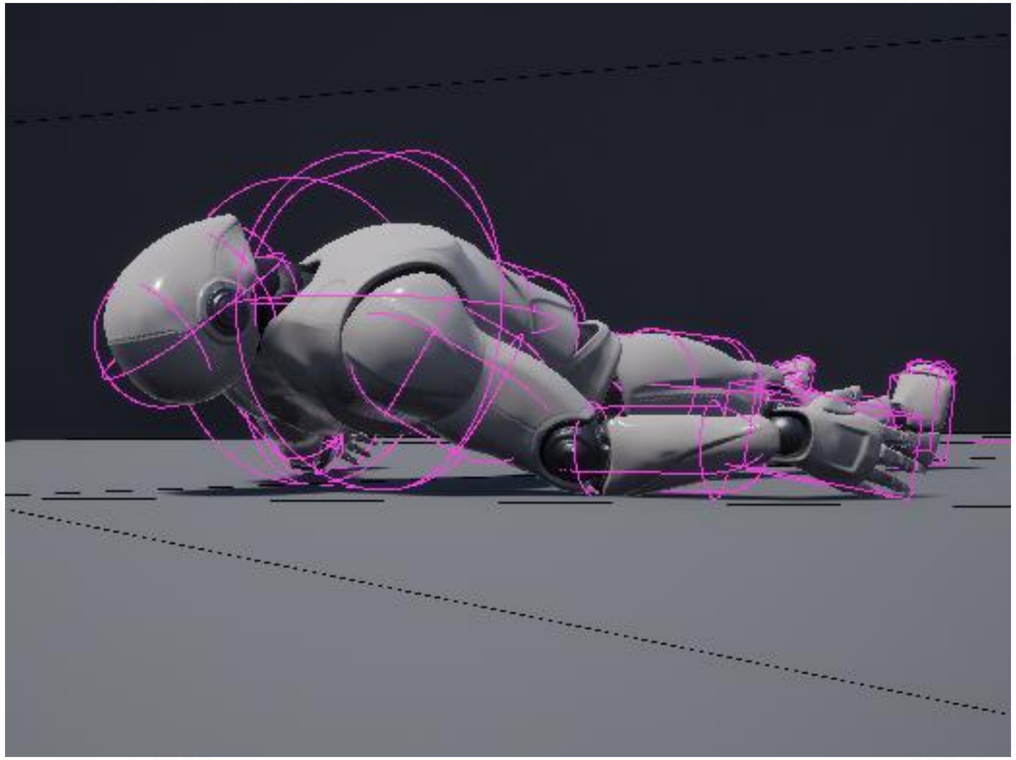
60° Limited Swing

Carefully Tweaked Constraints

The rigid bodies should fit the mesh as much as possible



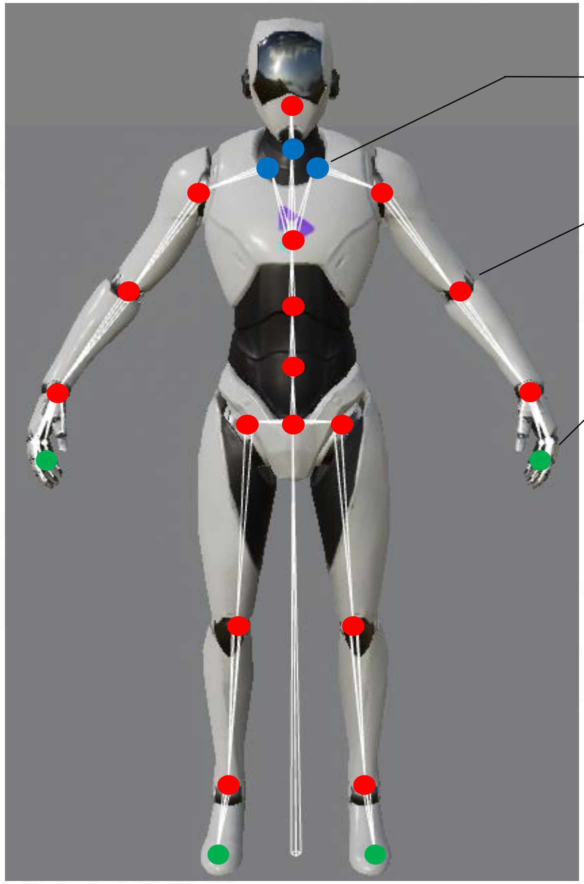
Correct result



Incorrect result if not unfitting

Animating Skeleton by Ragdoll

Update skeleton per frame



- Intermediate joints → Bind Pose
- Active joints → Rigid Body Pose
- Leaf joints → Animation Pose

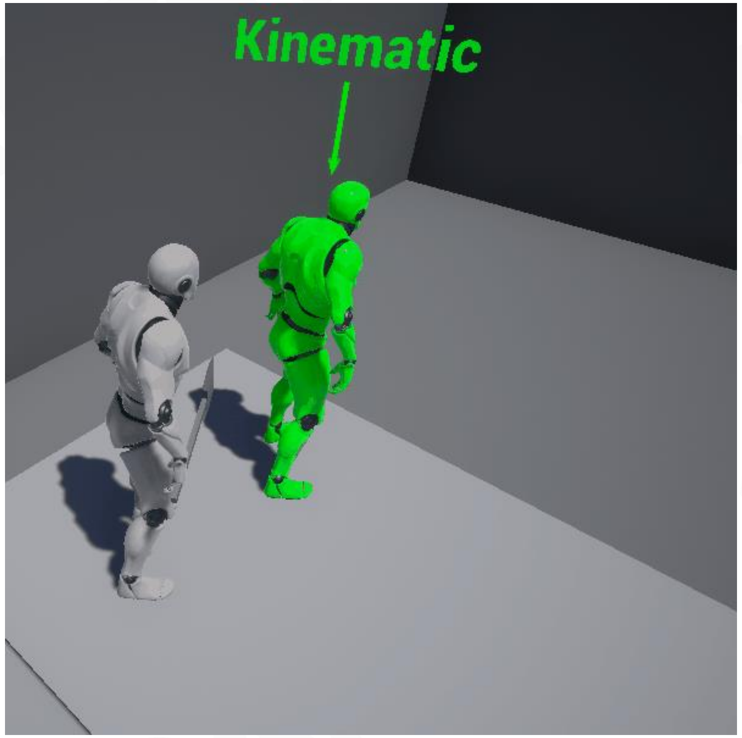
Blending between Animation and Ragdoll

Kinematic state ragdoll

- Rigid bodies are driven by animation

Dynamic state ragdoll

- Rigid bodies are simulated by physics

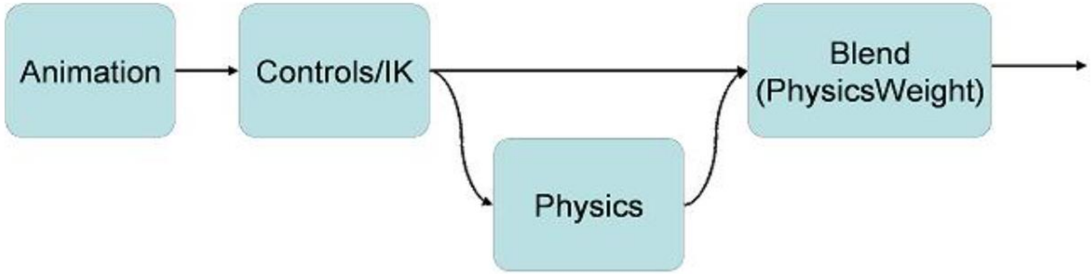


created | dead | destroyed → character state

ragdoll state: | ← kinematic → | ← dynamic → |

Powered Ragdoll – Physics-Animation Blending

Blend between the animation pose and the physics pose



Animation only

Ragdoll physics only

Physics-animation Blending

Clothing

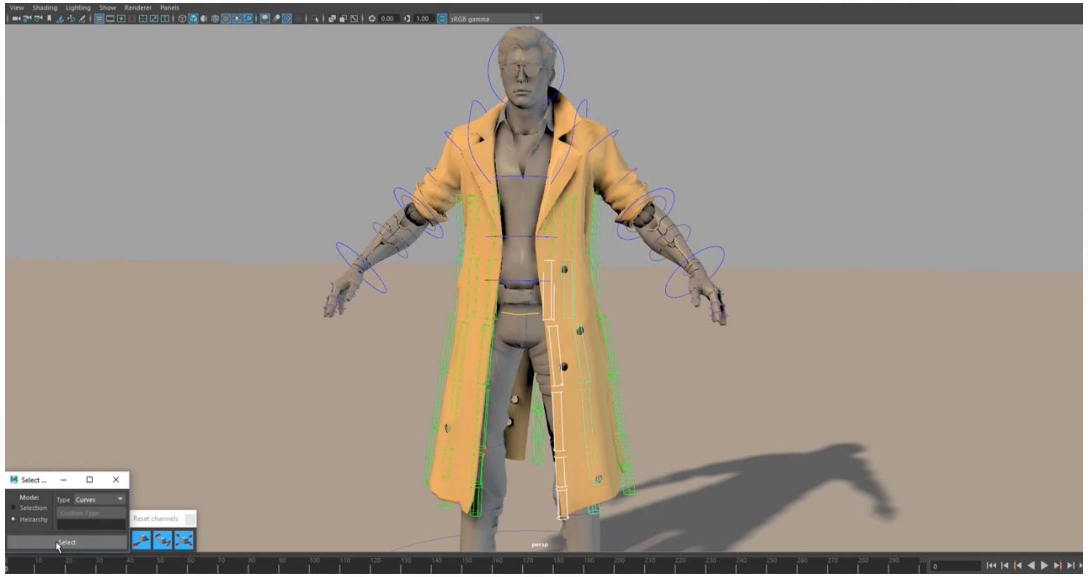
Animation-based Cloth Simulation

- Pipeline
  - Animators produce the animation of bones
  - Generate more animation data via DCC tools
  - Engine replays the animation when running
- Pros
  - Cheap
  - Controllable
- Cons
  - Not realistic
  - Could not interact with environment
  - The designation of clothes is limited

Popular among mobile games

Rigid Body-based Cloth Simulation

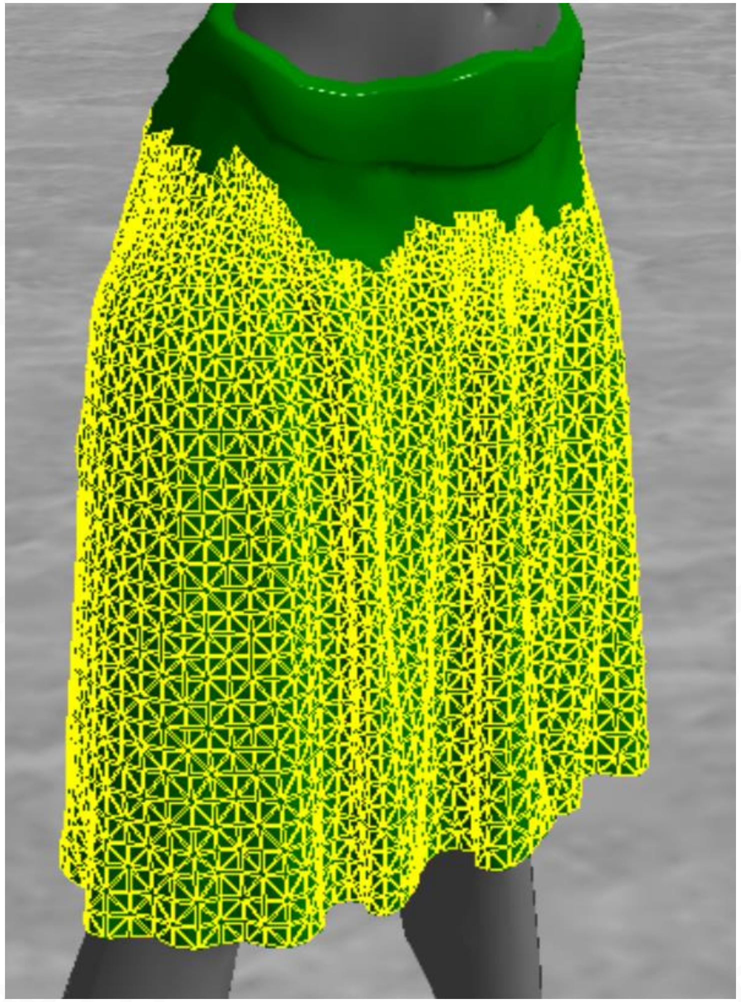
- Pipeline
  - The bones of cloth are bound with rigid bodies and constraints
  - The effect are solved by physics engine
- Pros
  - Cheap
  - Interactive
- Cons
  - Undetermined quality
  - Work load for animators
  - Not robust
  - Needs physics engine with high performance



Items like tails, special hair, and pendant

Mesh-based Cloth Simulation

Render Mesh vs. Physical Mesh

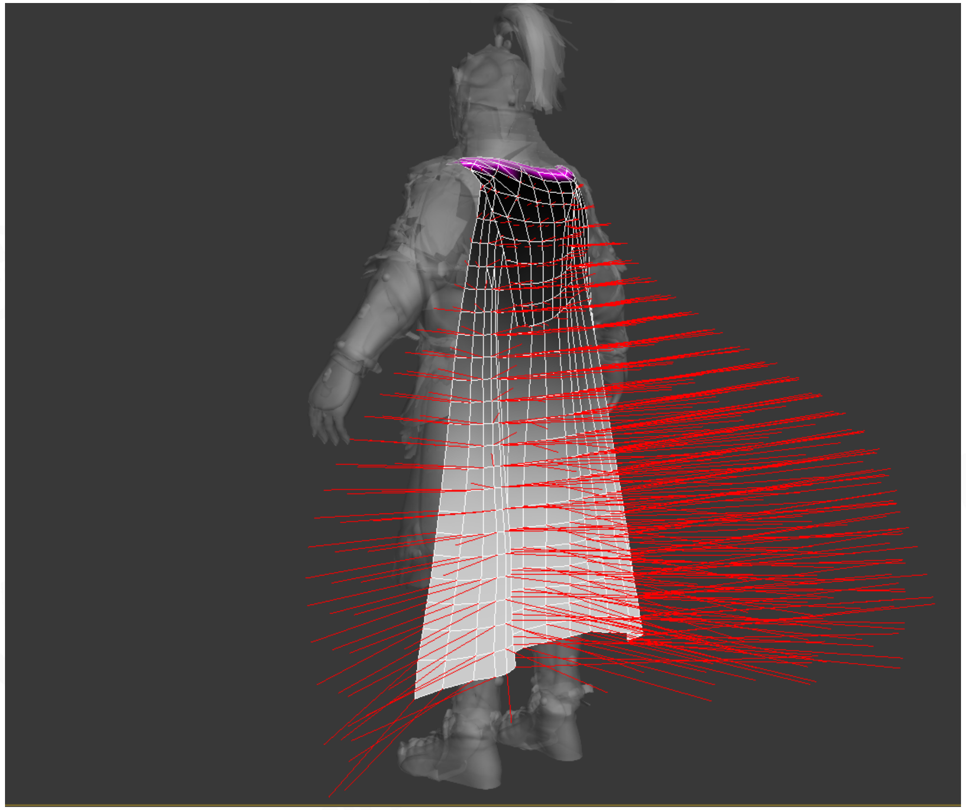


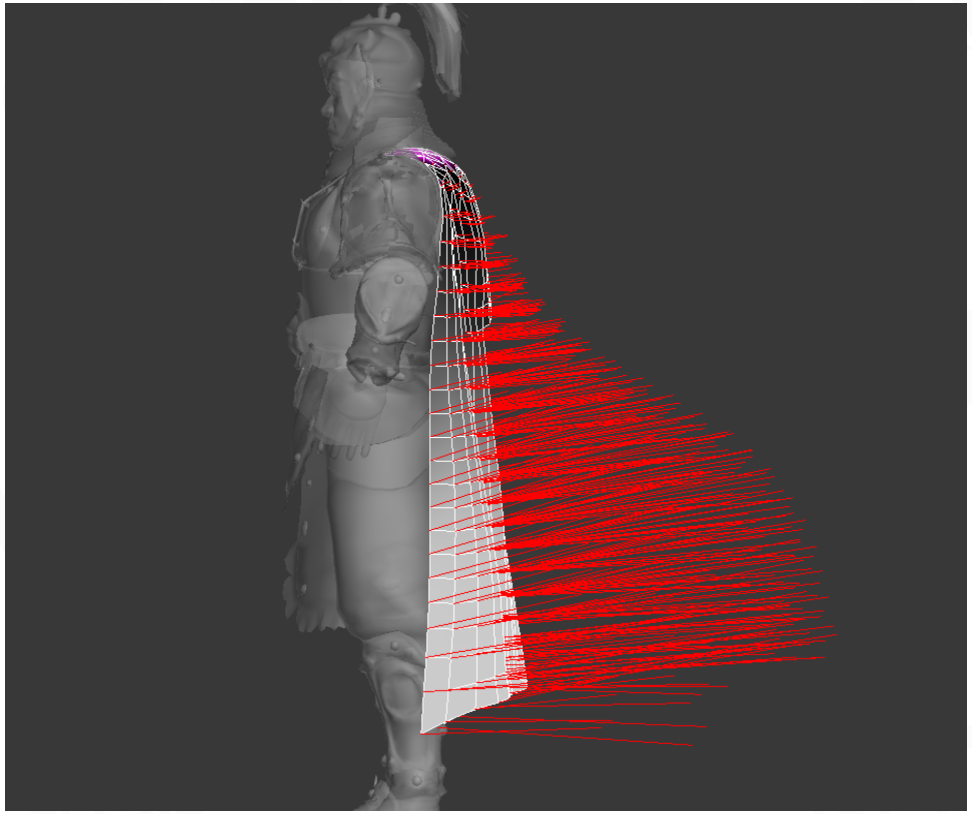
Render Mesh

Physical Mesh

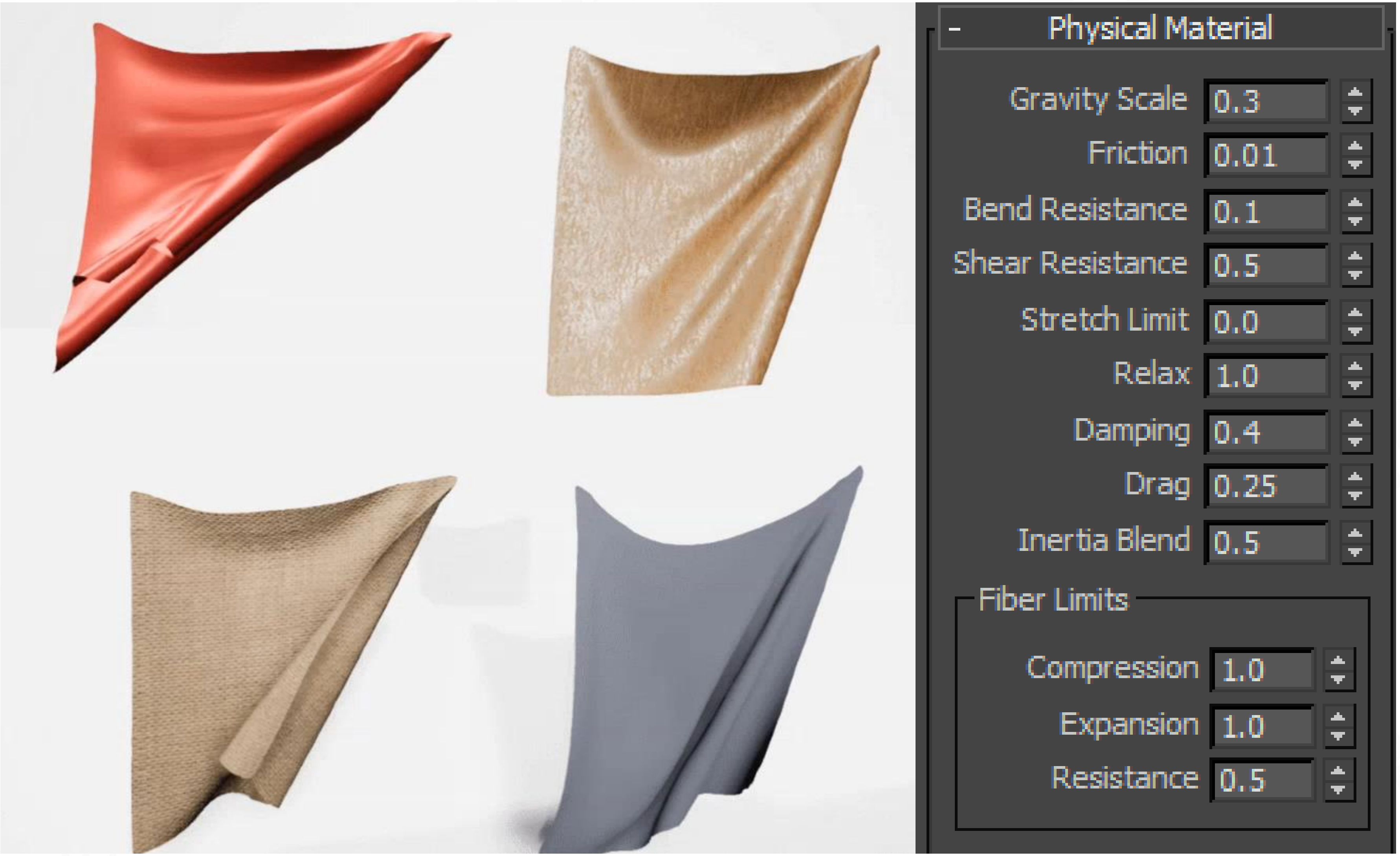
Paint Cloth Simulation Constraints

Add maximum radius constraints to each vertex



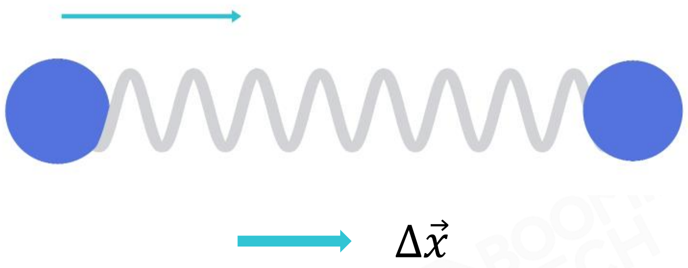


Set Cloth Physical Material

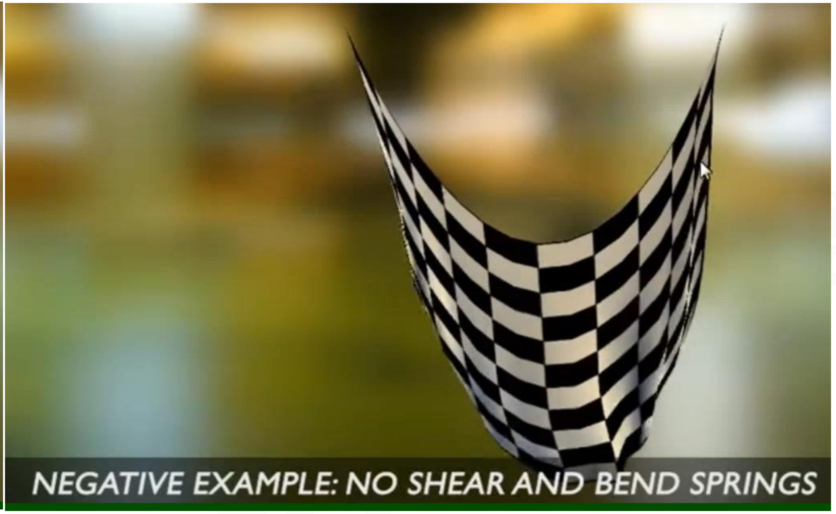


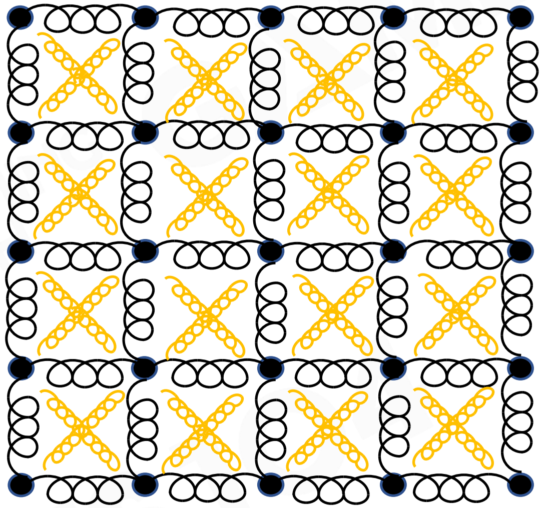
Cloth Solver – Mass-Spring system (1/3)

- Spring force
  - $\vec{F}^s = k_{\text{spring}} \Delta\vec{x}$
- Spring damping force
  - $\vec{F}^D = -k_{\text{damping}} \vec{v}$



Cloth Solver – Mass-Spring system (2/3)





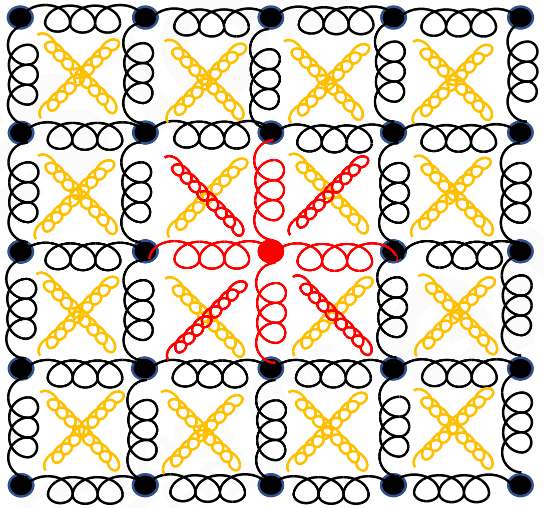
Cloth Solver – Mass-Spring system (3/3)

- For a vertex, we could apply force analysis on it

$$\vec{F}_{net}^{vertex}(t) = M\vec{g} + \vec{F}_{wind}(t) + \vec{F}_{air\ resistance}(t) + \sum_{Springs\in v}\left(k_{spring}\Delta\vec{x}(t) - k_{damping}\vec{v}(t)\right) = M\vec{a}(t)$$

- Then, we just need to use integrator to calculate the next position. In the cloth simulation, Verlet is a good choice.

$$\vec{x}(t+\Delta t) = 2\vec{x}(t) - \vec{x}(t-\Delta t) + \vec{a}(t)(\Delta t)^2$$



Verlet Integration

- Recap Semi-Euler method

$$\begin{cases} \vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\,\Delta t \\ \vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t+\Delta t)\Delta t \end{cases}$$

Verlet Integration

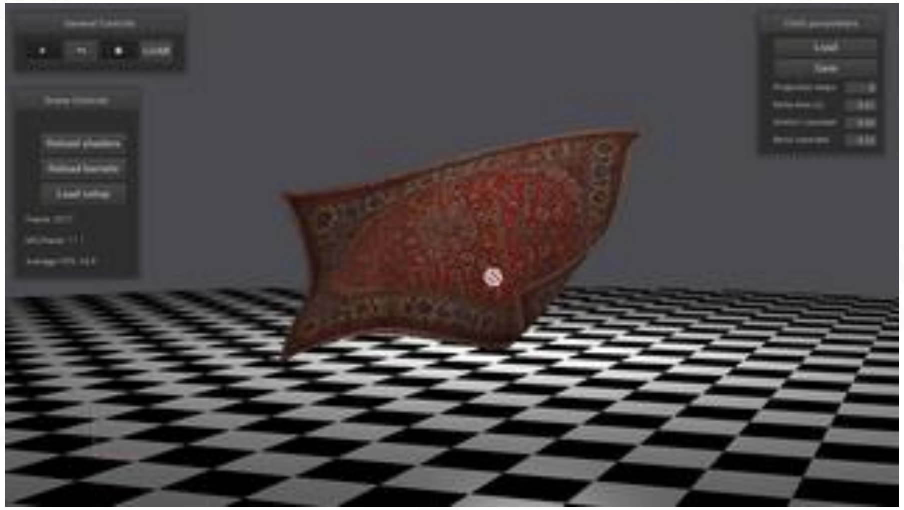
- Recap Semi-Euler method

$$\begin{cases} \vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\,\Delta t \\ \vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t+\Delta t)\Delta t \\ \vec{x}(t) = \vec{x}(t-\Delta t) + \vec{v}(t)\Delta t \end{cases} \Rightarrow \begin{cases} \vec{x}(t+\Delta t) = \vec{x}(t) + (\vec{v}(t) + \vec{a}(t)\,\Delta t)\Delta t \\ \vec{x}(t) = \vec{x}(t-\Delta t) + \vec{v}(t)\Delta t \end{cases}$$

$$\Rightarrow \vec{x}(t+\Delta t) = 2\vec{x}(t) - \vec{x}(t-\Delta t) + \vec{a}(t)(\Delta t)^2$$

Verlet integration does not need to consider about velocity when calculate, so it is faster

Cloth Solver – Position Based Dynamics



Basically, the simulation needs

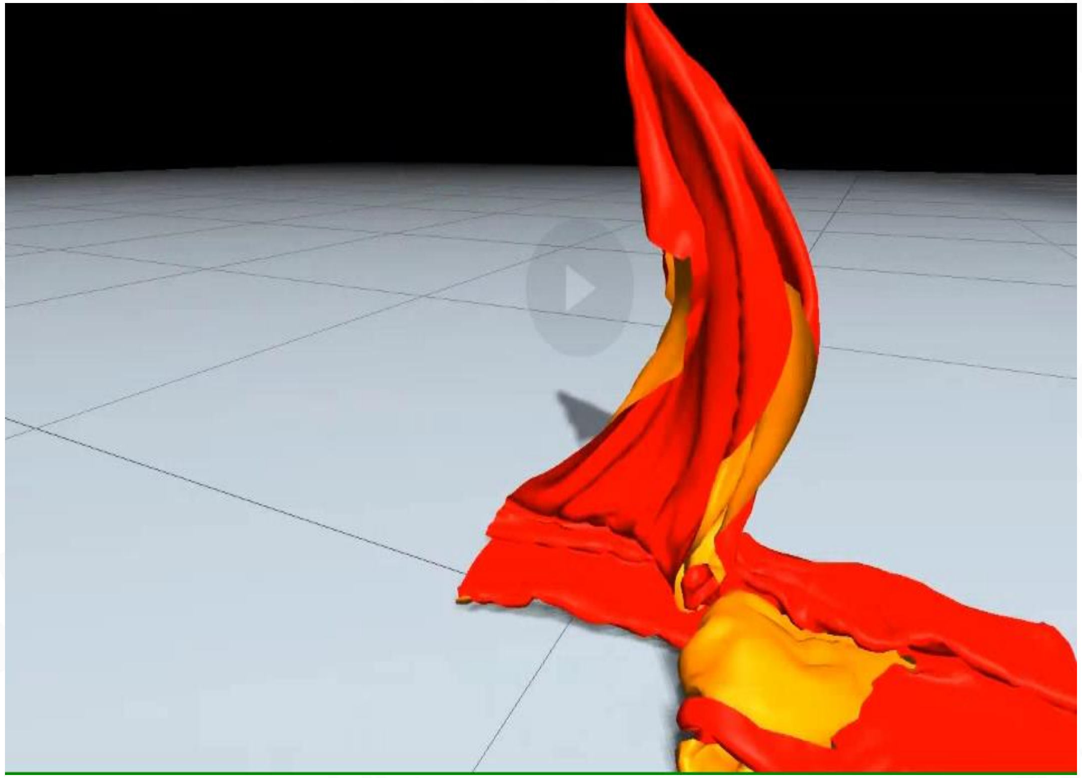
Constrains → Force → Velocity → Position

Luckily, we have Position Based dynamics (PBD)

Constrains → Position

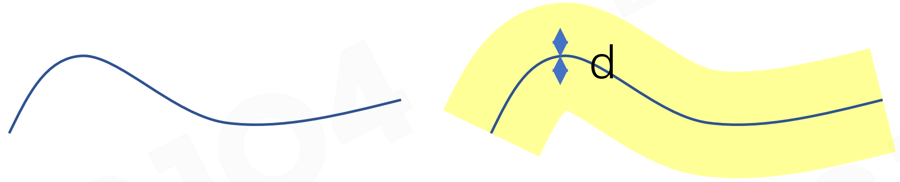
Self Collision

- As a kind of flexible material, cloth can fold and collide with itself
- This is pretty tricky in real-time game physics simulation

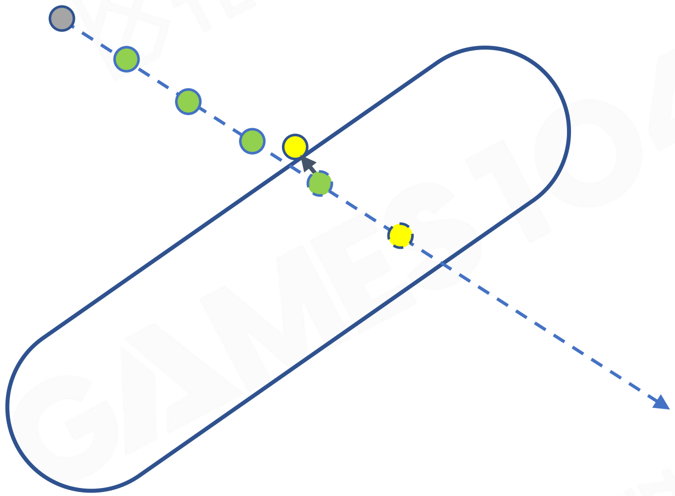


Common Solutions for Self Collision (1/2)

- Make the cloth thicker

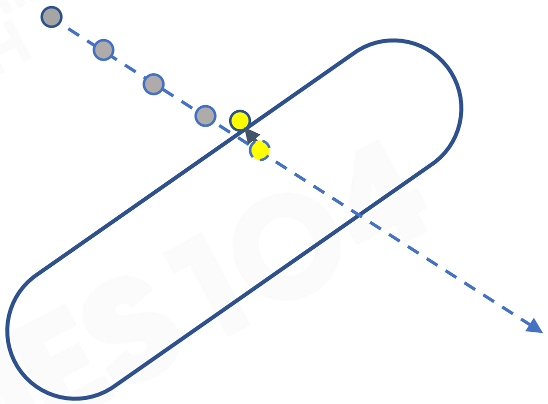


- Use many substeps in one physics simulation step

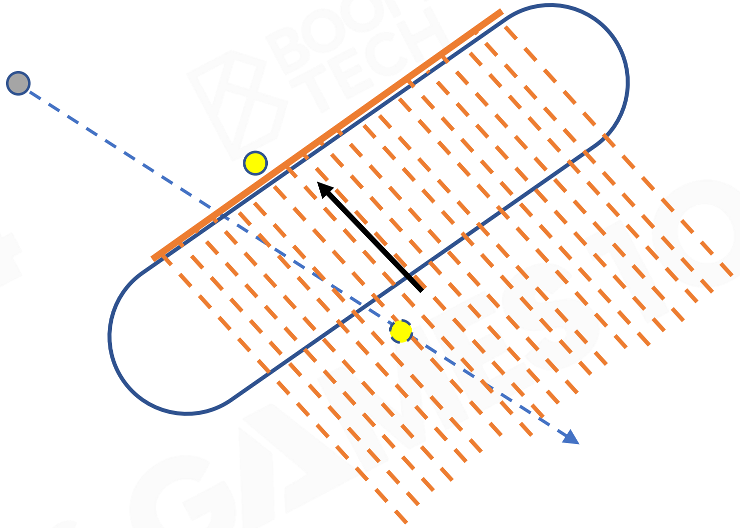


Common Solutions for Self Collision (2/2)

- Enforce maximal velocity



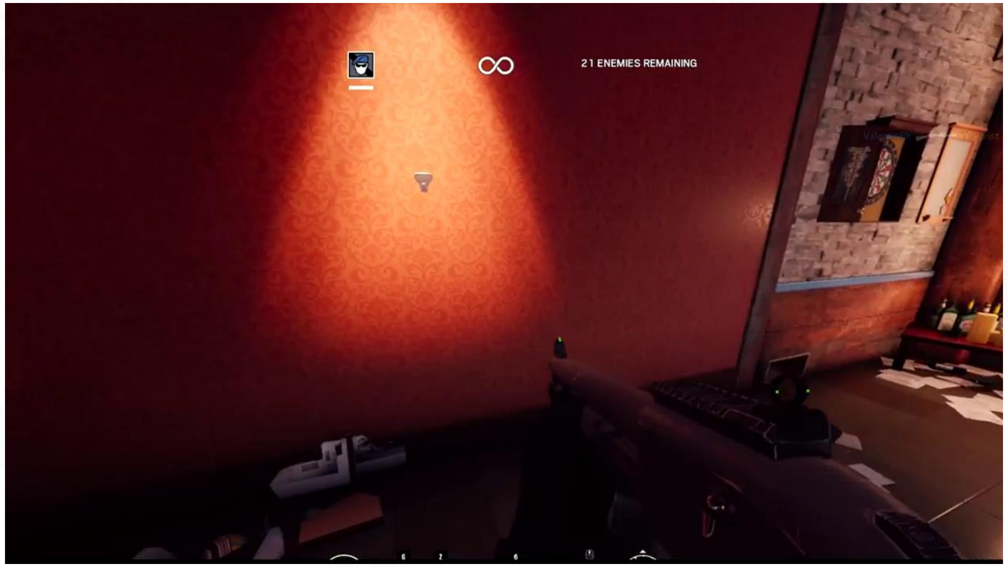
- Introduce contact constraints and friction constraints

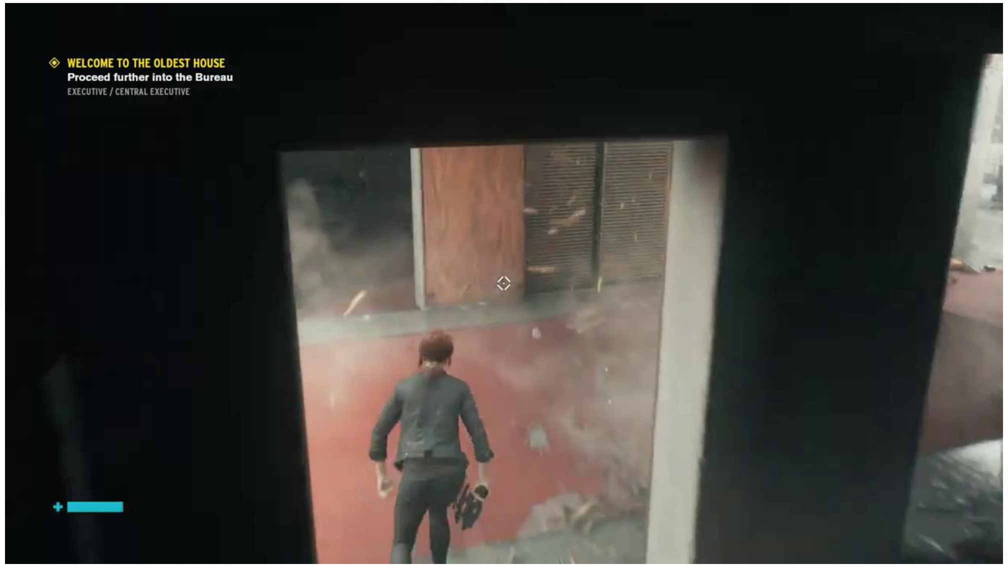


Destruction

Destruction is Important

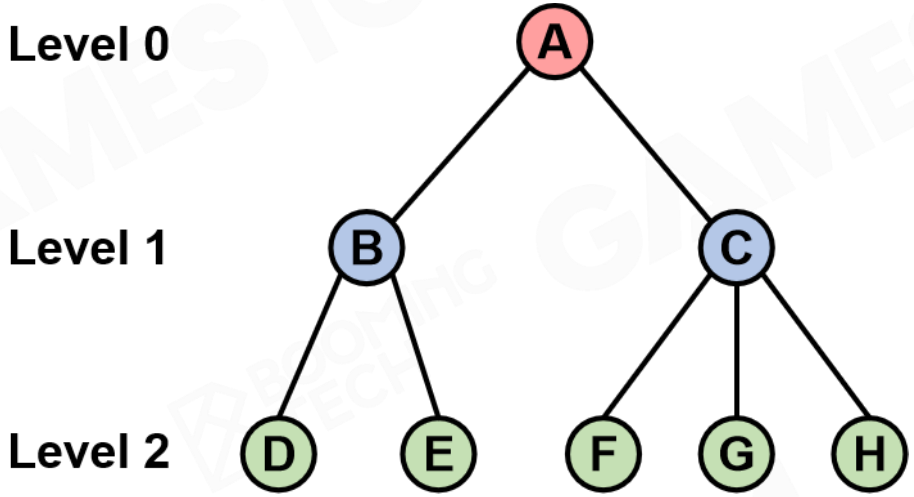
- Not only a visual effect
- Making the game world much more vivid and immersive
- A key mechanism in many games

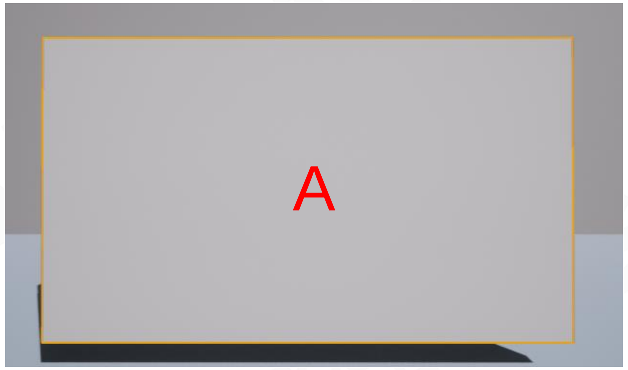


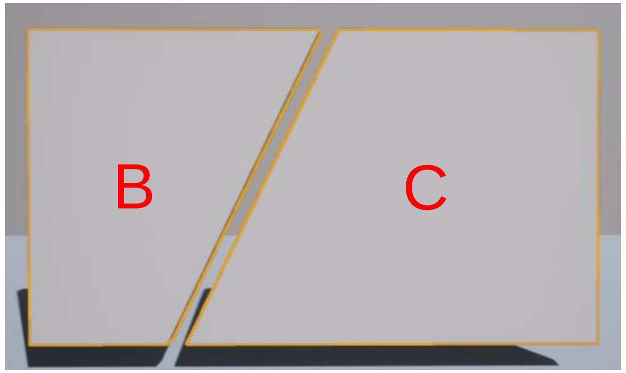


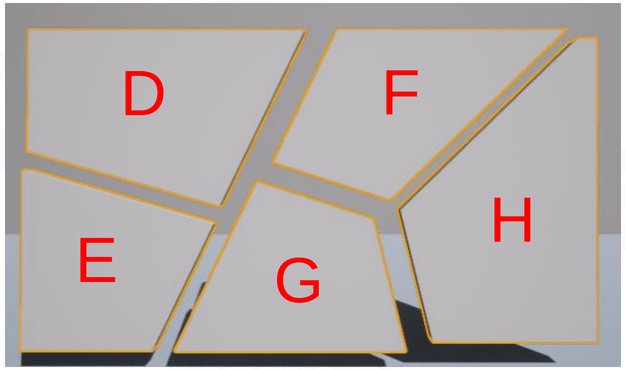
Chunk Hierarchy

- Organize the fractured chunks level by level
- Different damage threshold for each level





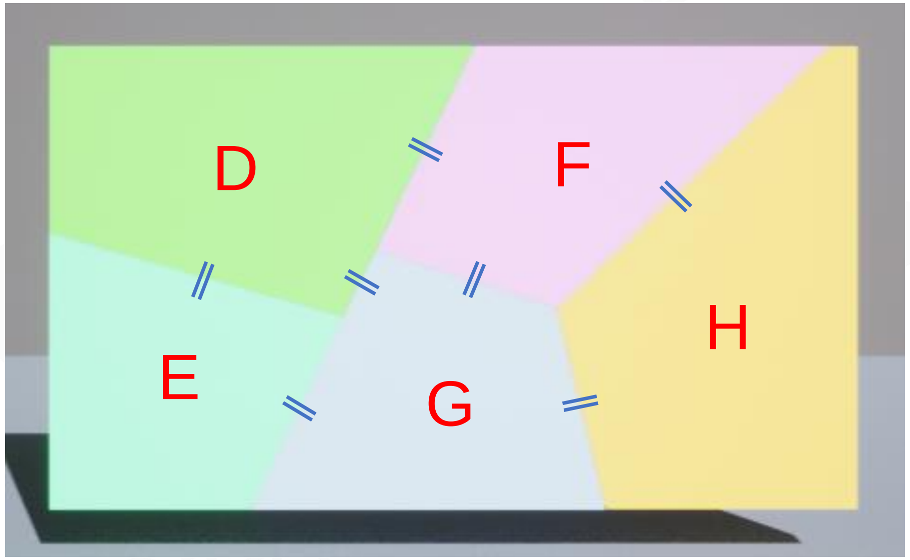


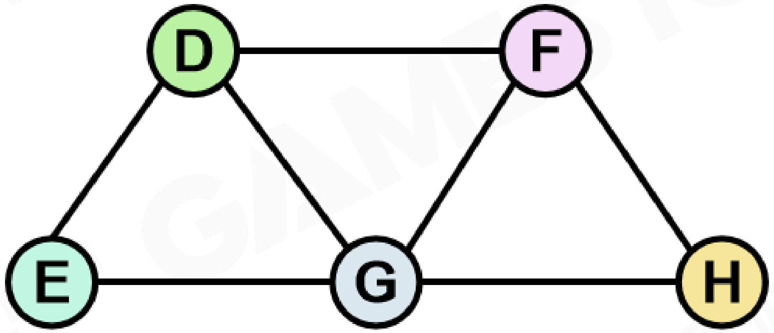


Connectivity Graph

Construct connectivity graph for chunks at the deepest level

- One node per chunk
- One edge if two chunks share a face
- Update at runtime

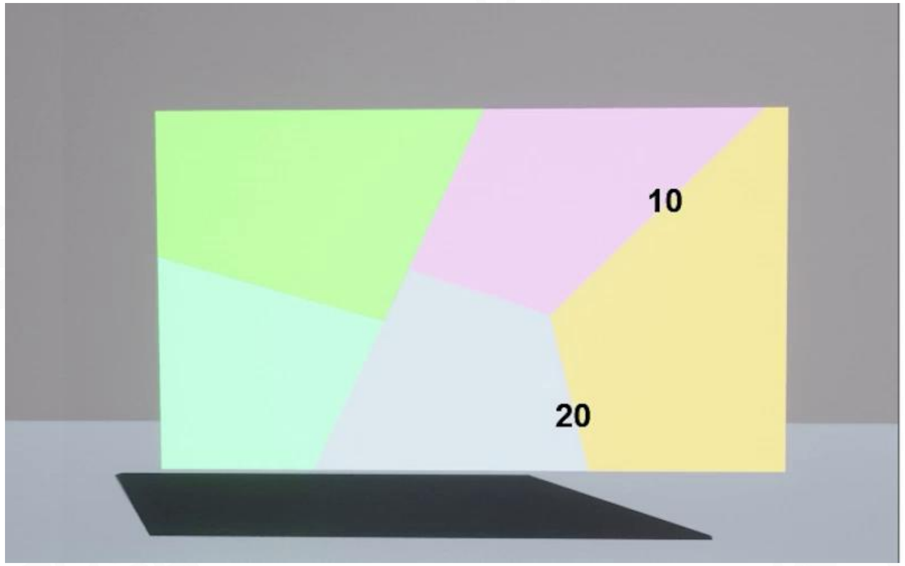


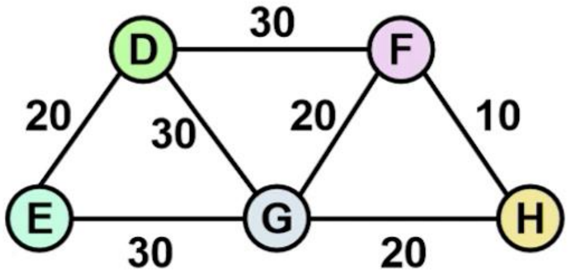


Connectivity Value

The value on each edge in the connectivity graph

- How much damage needed to break the edge
- Update after each damage
- Break the edge when the value goes to 0





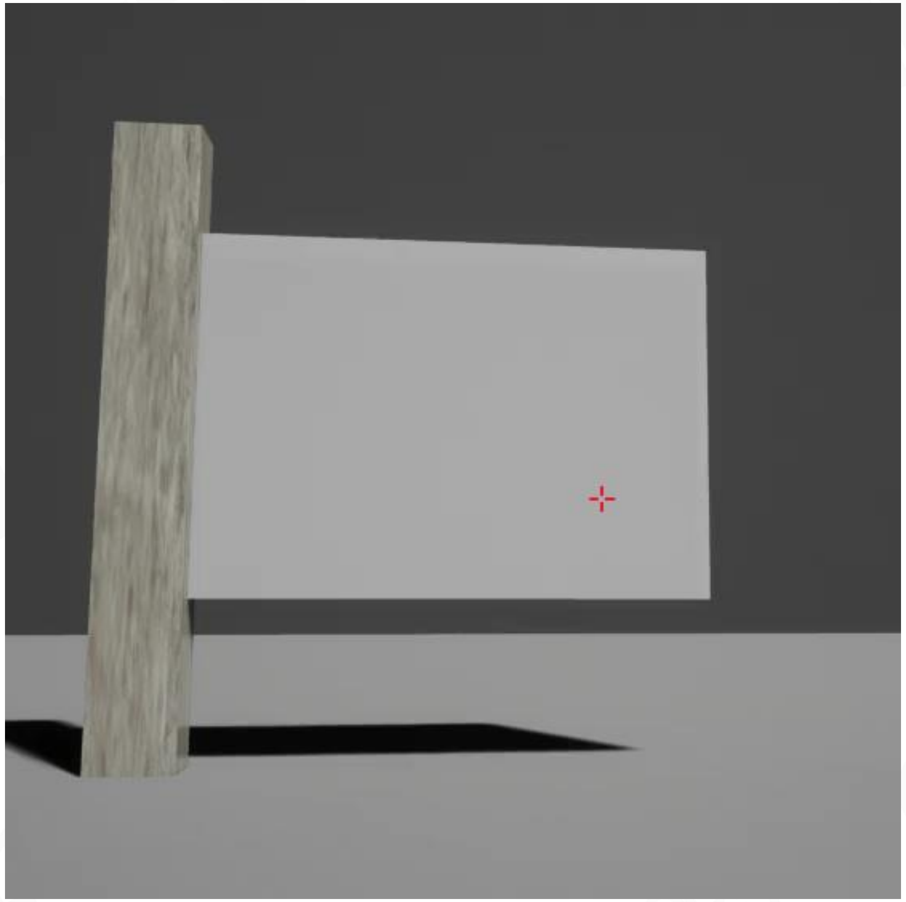
Damage Calculation (1/2)

Calculate damage from impulse at the impact point

- I : the applied impulse (e.g. by collision)
- H : the material hardness of the rigid body

The damage at the impact point is

$$D = \frac{I}{H}$$



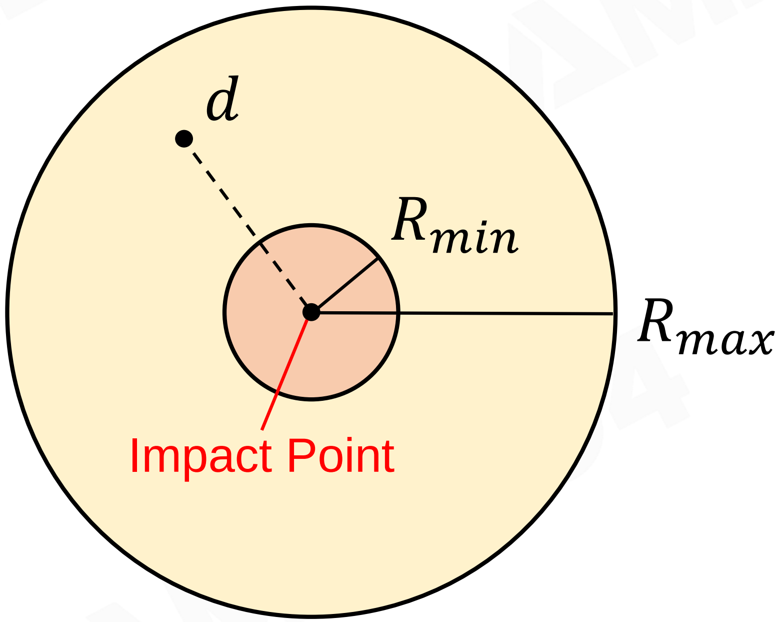
Damage Calculation (2/2)

Damage distribution

- D : the damage at the impact point
- R_{min} : the minimum damage radius
- R_{max} : the maximum damage radius
- k : the damage fall off exponent

The damage D_d at distance d is

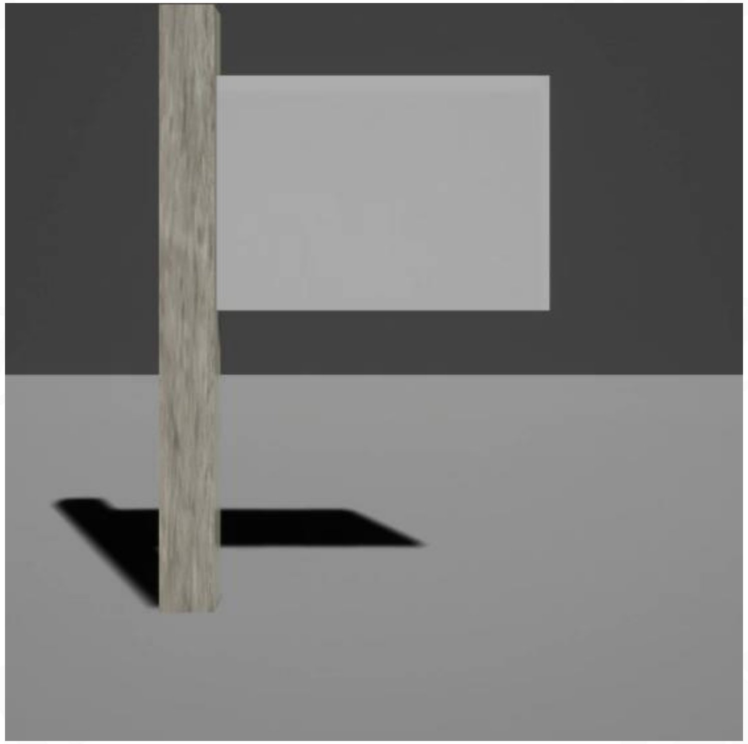
$$D_d = \begin{cases} D & d \leq R_{min} \\ D \cdot \left(\dfrac{R_{max} - d}{R_{max} - R_{min}}\right)^K & R_{min} < d < R_{max} \\ 0 & d \geq R_{max} \end{cases}$$



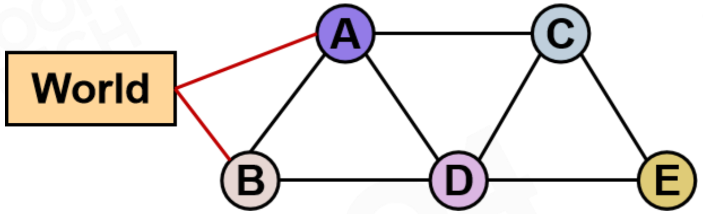
Damage Range

Destruction with/without Support Graph

Not connect with world



Connect with world



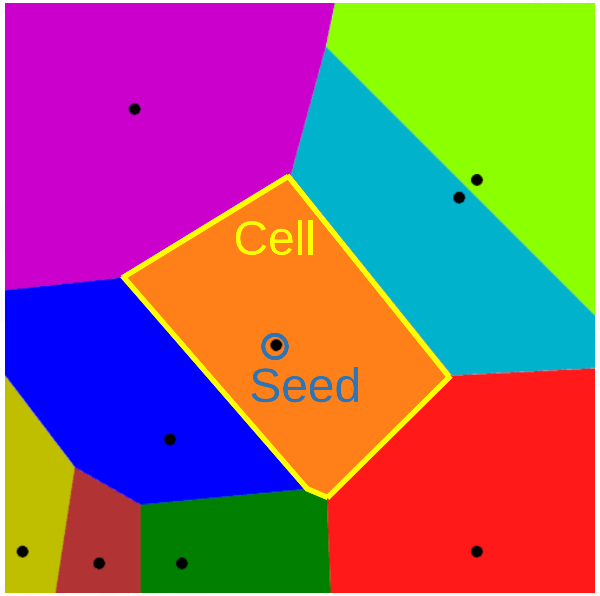
Support Graph

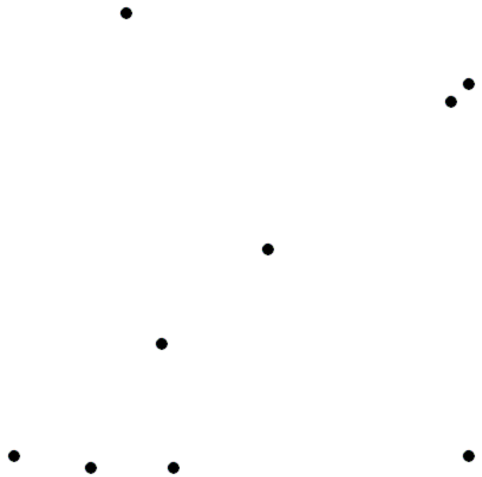
Build Chunks by Voronoi Diagram

A partition of a plane into regions close to each of **seeds**

Voronoi Cell

- the region for each seed
- Points in the cell are closer to the seed than to any other

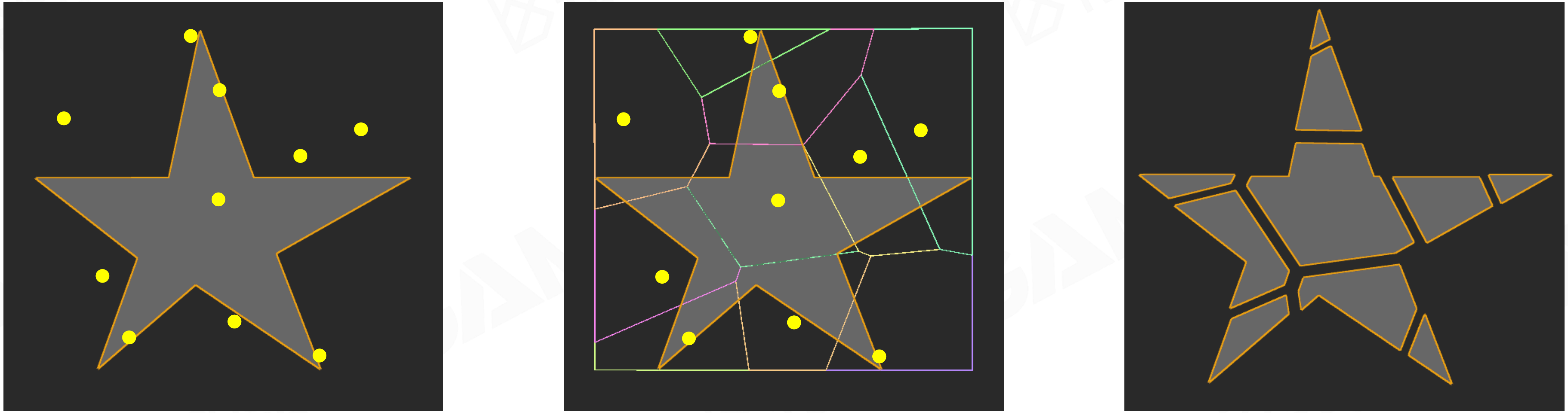




Fracturing with Voronoi Diagram - 2D Mesh

Pick **N** random points within the bounding rect of the mesh

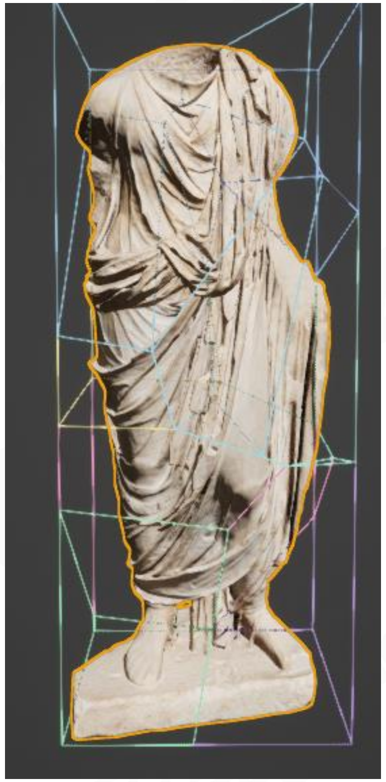
- Construct the Voronoi diagram
- Intersect each Voronoi cell with the mesh to get all fractured chunks



Fracturing with Voronoi Diagram - 3D Mesh (1/2)

Similar to the 2D situation, but not trivial

- New triangles need to be generated for all fracture surfaces



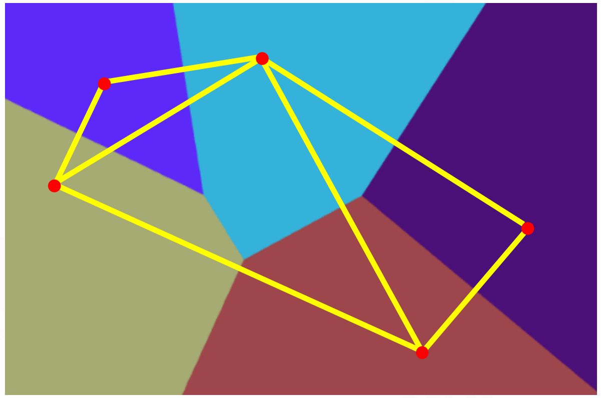
Fracturing with Voronoi Diagram - 3D Mesh (2/2)

Generate triangles for all fracture surfaces

- Usually by Delaunay Triangulation(is dual to Voronoi diagram)
- New texture and texture coordinates

New texture for fracture surfaces

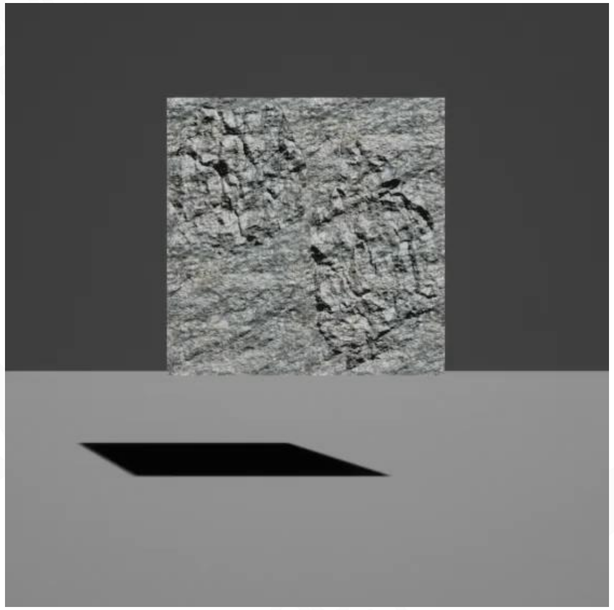
A fracture surface



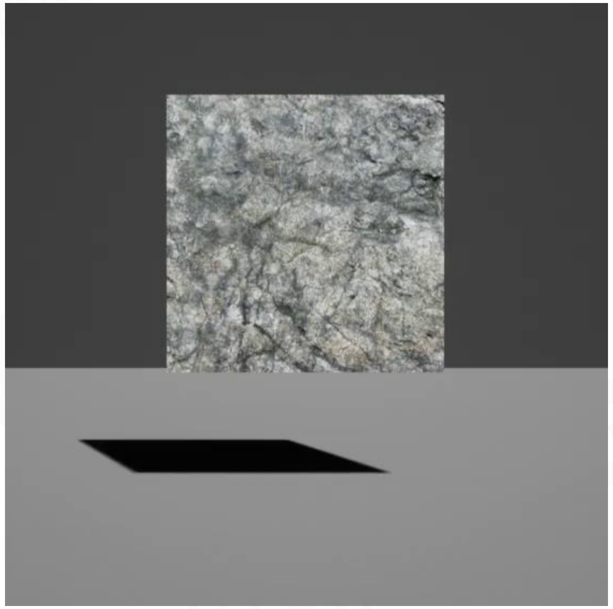
Delaunay Triangulation and its dual Voronoi Diagram

Different Fracture Patterns with Voronoi Diagram

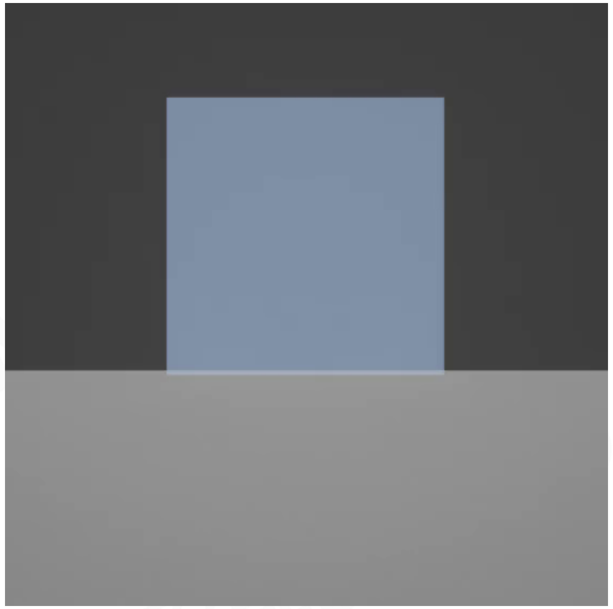
- Uniform random pattern
- Clustered pattern
- Radial pattern
- etc.



Uniform Random Pattern



Clustered Pattern

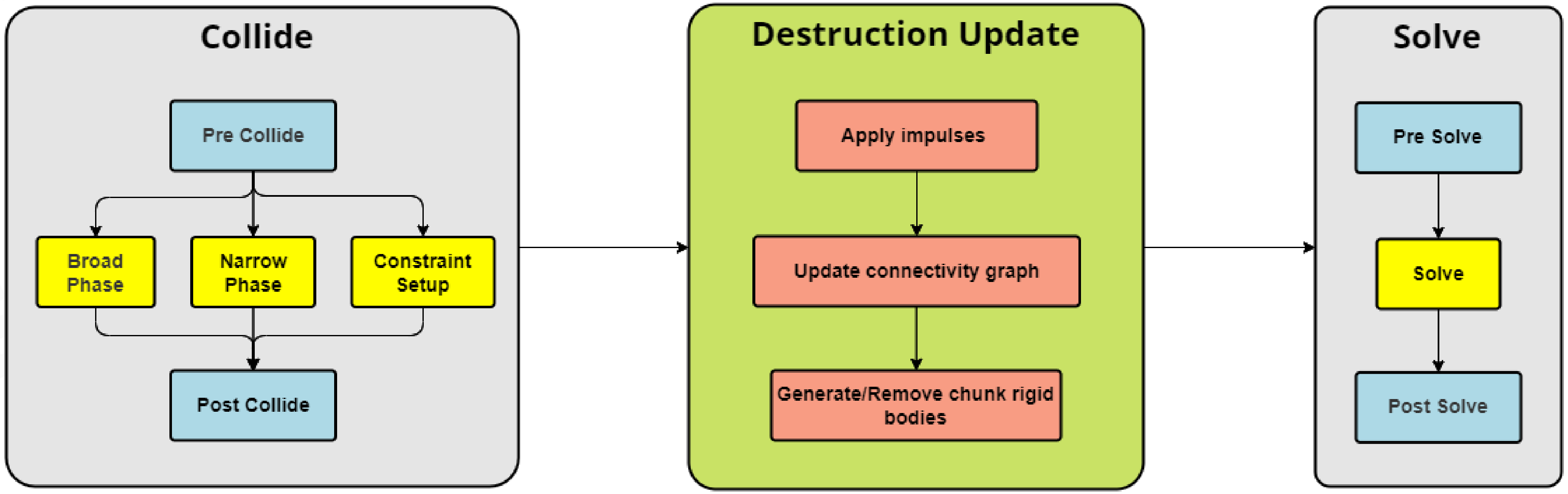


Radial Pattern

Destruction in Physics System (2/2)

Handle destruction after collision

- New rigid bodies may be generated after destruction



Make it more realistic

Fracture is not enough

- Sound effects
- Particle effects
- Navigation updated

Make path by destruction

Fracture with Particle

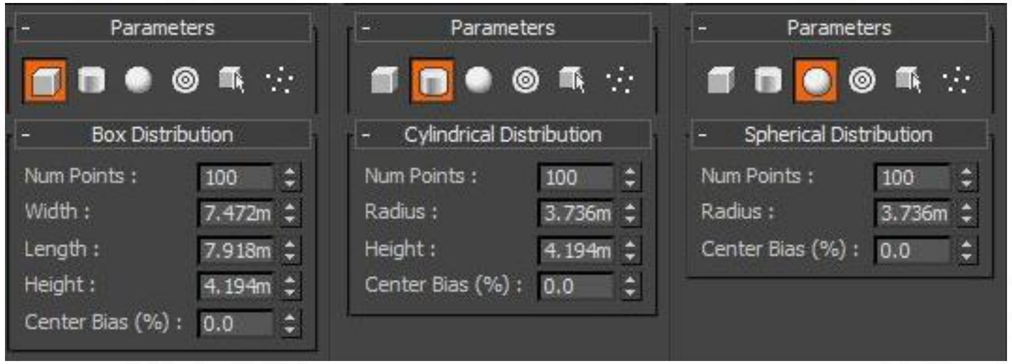
Issues with Destruction

Add destruction with caution

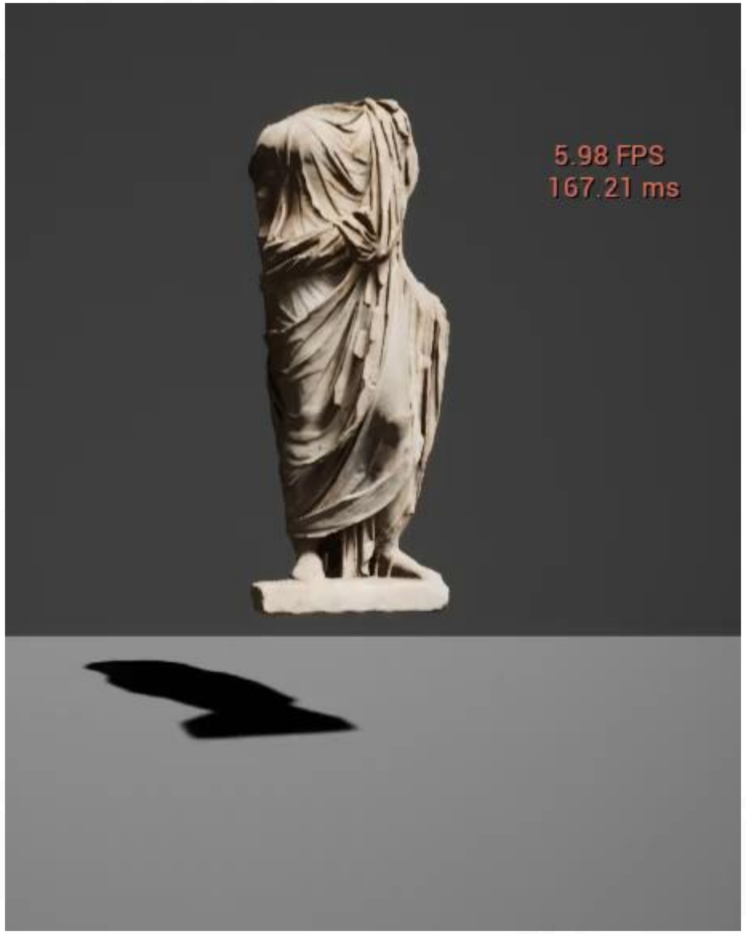
- Numerous debris may cause larger performance overhead

Empirical when artists design the destruction effect

- Many parameters to be tuned, e.g. fracture parameters
- Produce performance highly depends on the authoring tool



Some mesh fracture parameters



Performance overhead with numerous debris

Popular Destruction Implementations (1/2)

NVIDIA APEX Destruction

- Widely used in games(supported in UE4)
- Official destruction authoring tool(PhysX Lab)
- Deprecated in 2017

NVIDIA Blast

- Successor of APEX
- Better performance, scalability and flexibility

Blast in NVIDIA Omniverse

Popular Destruction Implementations (2/2)

Havok Destruction

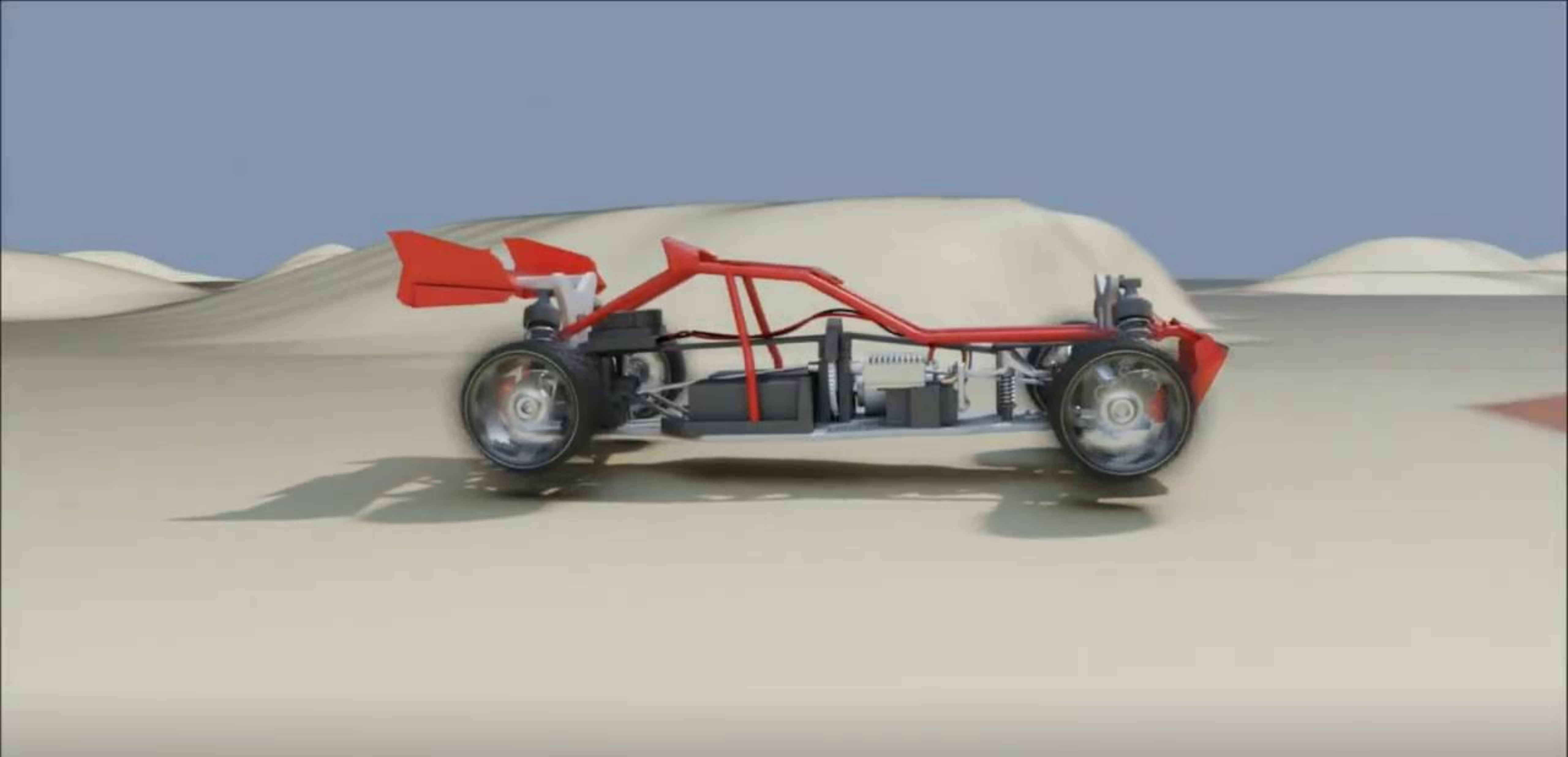
- Widely used in games(supported in Unity)
- Good performance and various features
- High license fee

Chaos Destruction(Epic Games.)

- Complete tool chain supported
- Still beta in UE5

Havok Destruction in *Battlefield Hardline*

Chaos Destruction Demo



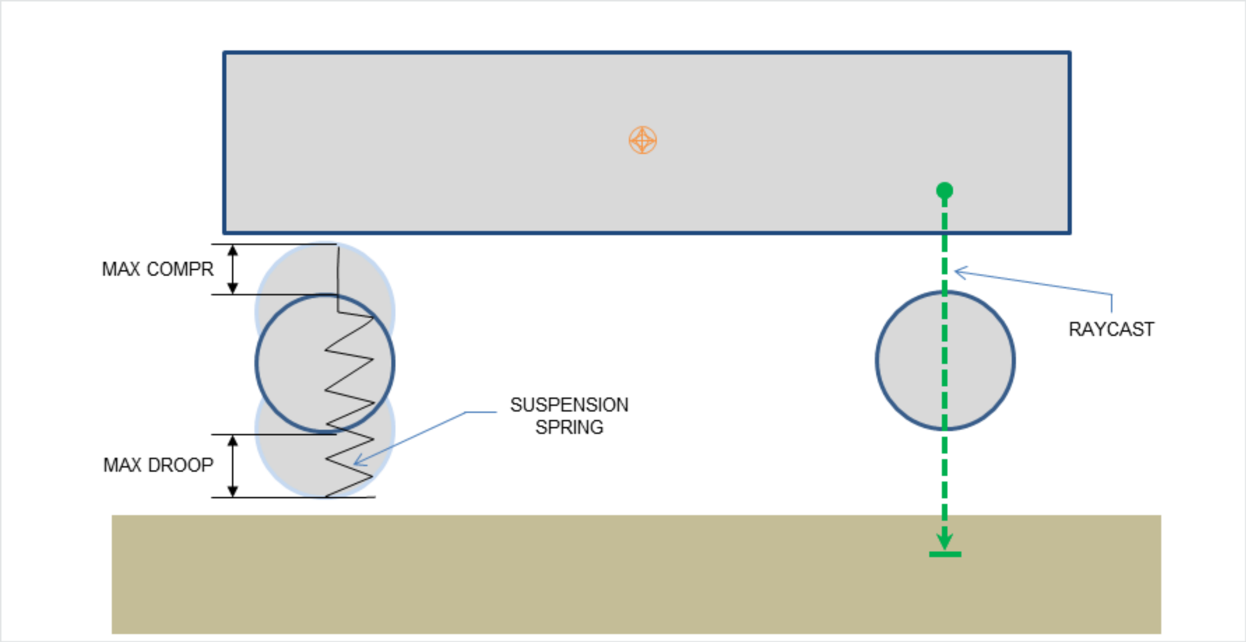
Vehicle

Vehicle Implementation Spectrum

Stylized → Realistic

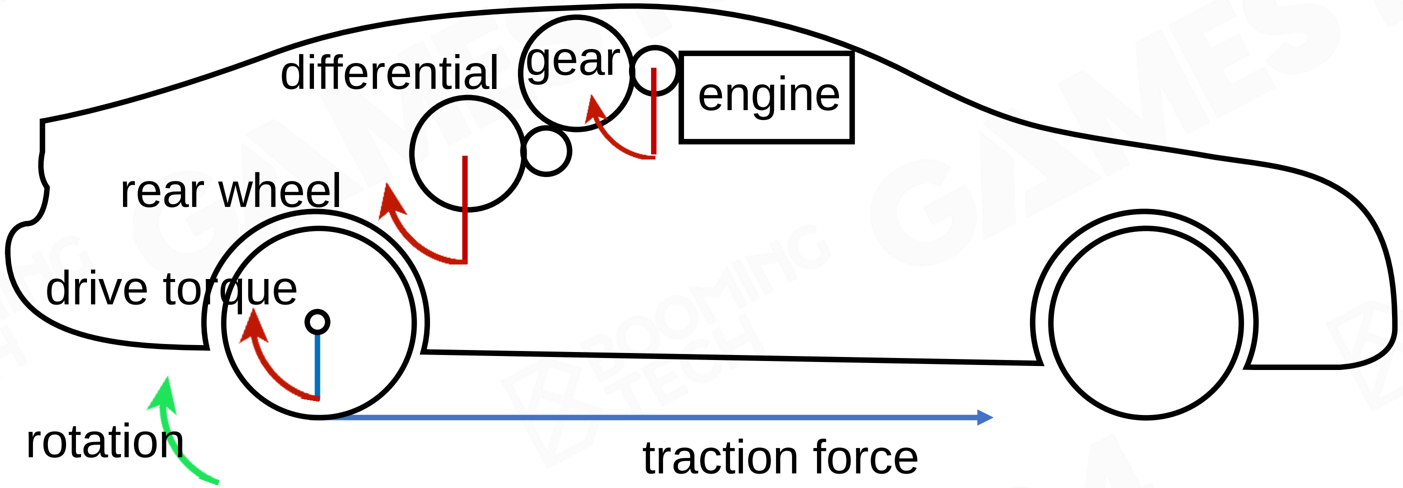
Vehicle Mechanism Modeling

- A rigid body actor
  - Shapes for the chassis and wheels
  - Scene query for the suspension simulation



Traction Force

- Get torque from a curve
  - $T = T_{engine} X_g X_d n$
- Calculate traction
  - $\vec{F}_{traction} = \frac{T}{R_w} \vec{u}$

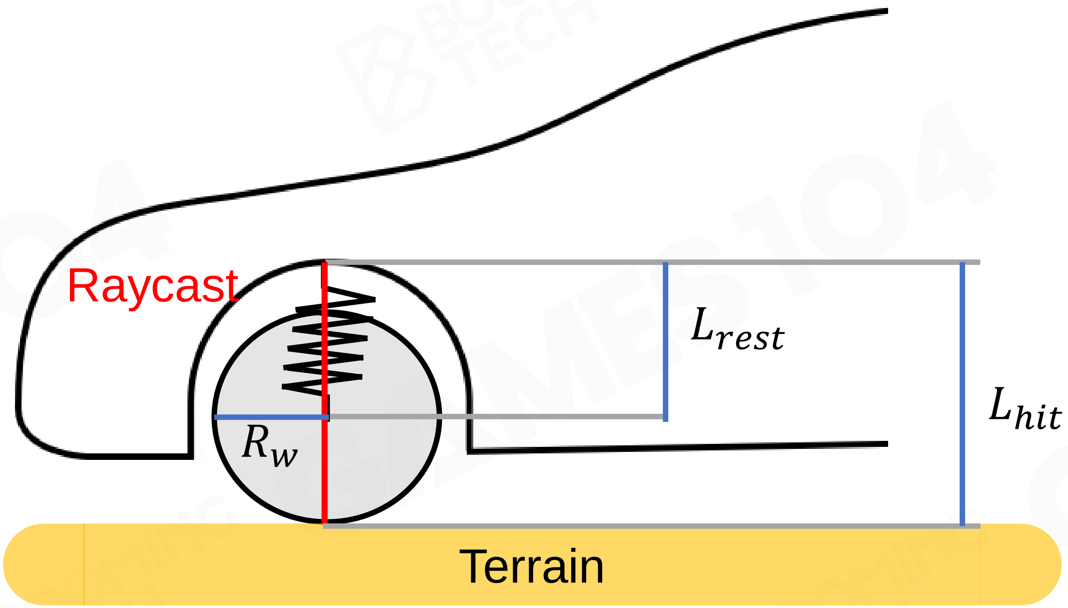


$\vec{u}$:a unit vector which reflects vehicle heading
T :wheel torque
T_{engine} :engine torque represented by curves
X_g :the gear ratio
X_d :the differential ratio
n :transmission efficiency and
R_w :wheel radius

Suspension Force

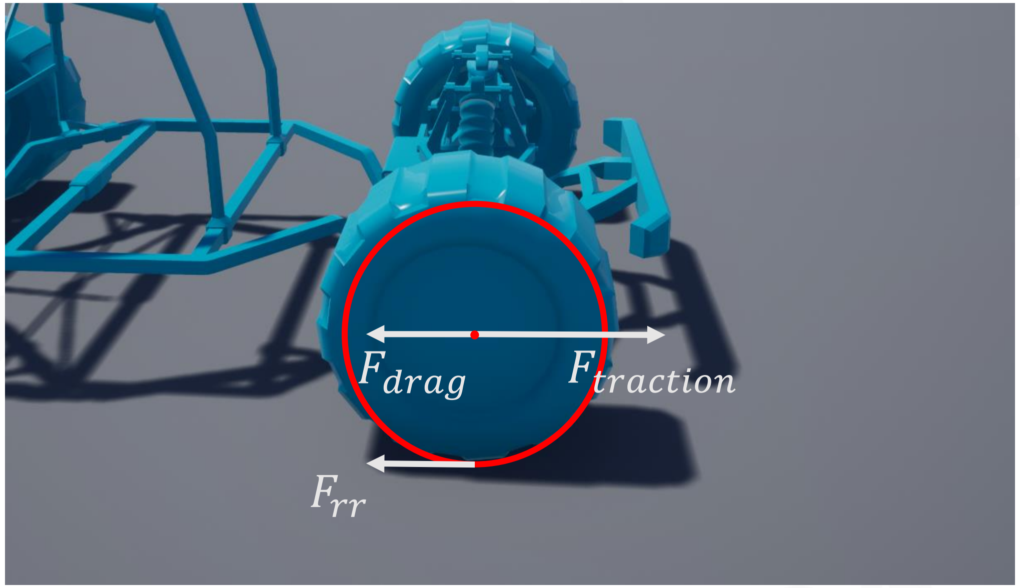
- Apply on attachment points of chassis and suspension
- Calculated independently for each wheel
  - $\left|\vec{F}_{suspension}\right| = k(L_{rest} - (L_{hit} - R_w))$

k	:spring stiffness
R_w	:wheel radius
L_{rest}	:length of spring rest
L_{hit}	:distance of raycast hit



Tire Forces

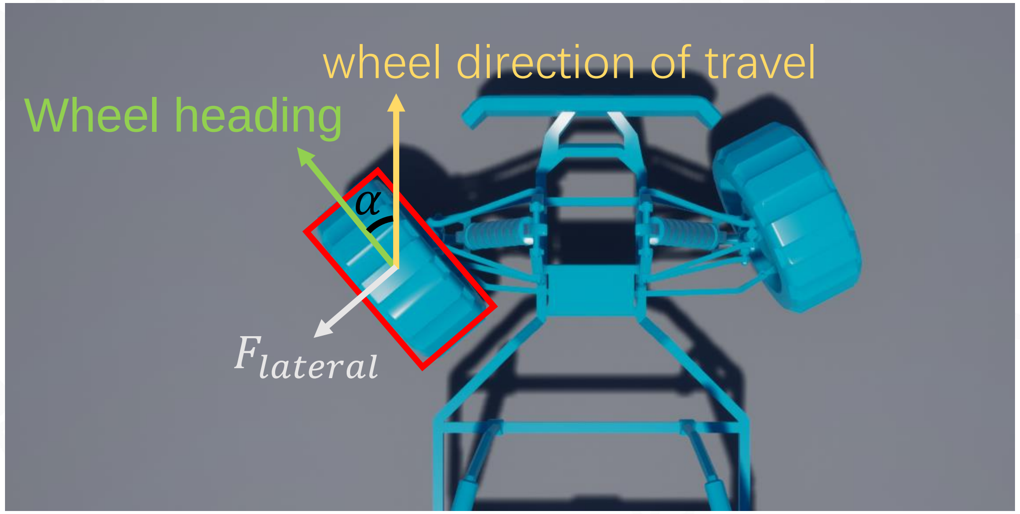
- Longitudinal force
  - $F_{long} = F_{traction} + F_{drag} + F_{rr}$
- Lateral force
  - $F_{lateral} = C_c * \alpha$



F_{rr}: rolling resistance

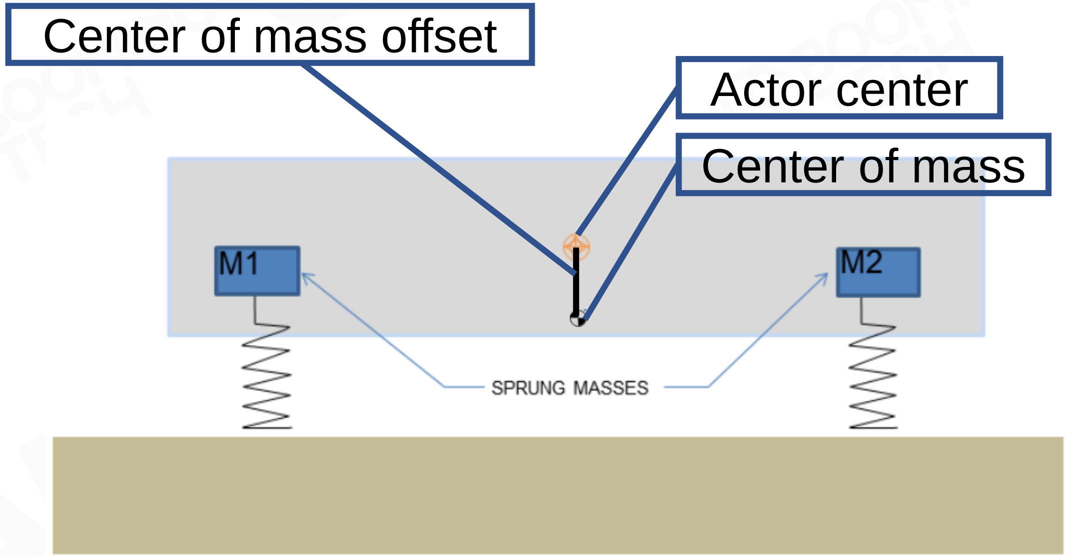
C_c : cornering stiffness

α : slip angle



Center of Mass (1/3)

- Affects handling, acceleration, and traction
- Should be a tunable value



$M_1,\ M_2$: the sprung masses

$\vec{x}_1,\ \vec{x}_2$: the sprung mass coordinates in actor space

M : the rigid body mass

$\vec{x}_{cm}$: the rigid body center of mass offset

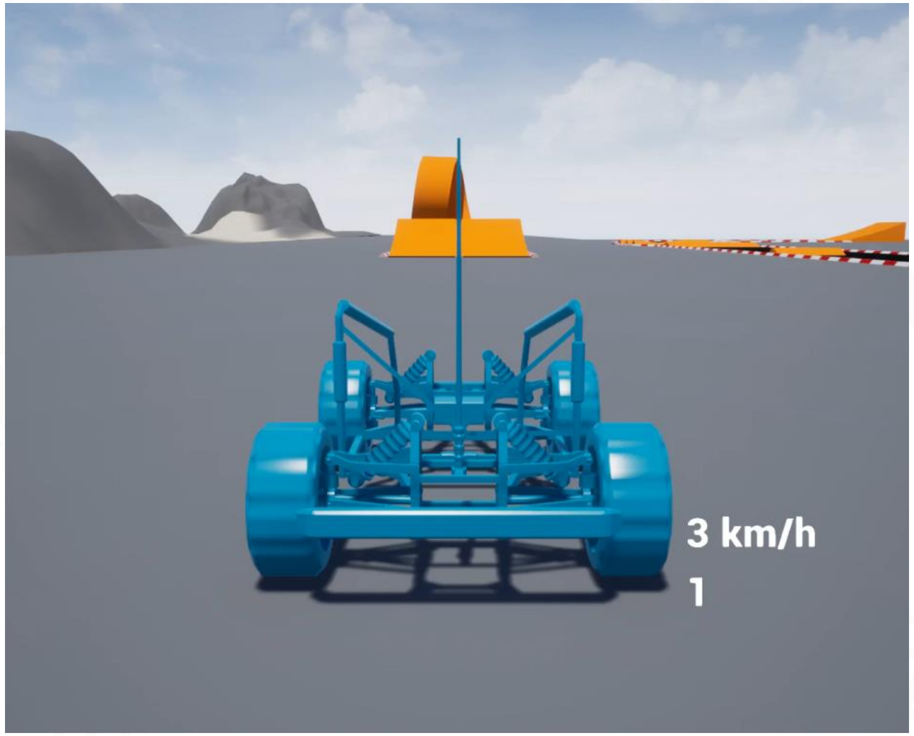
$$M = M_1 + M_2$$

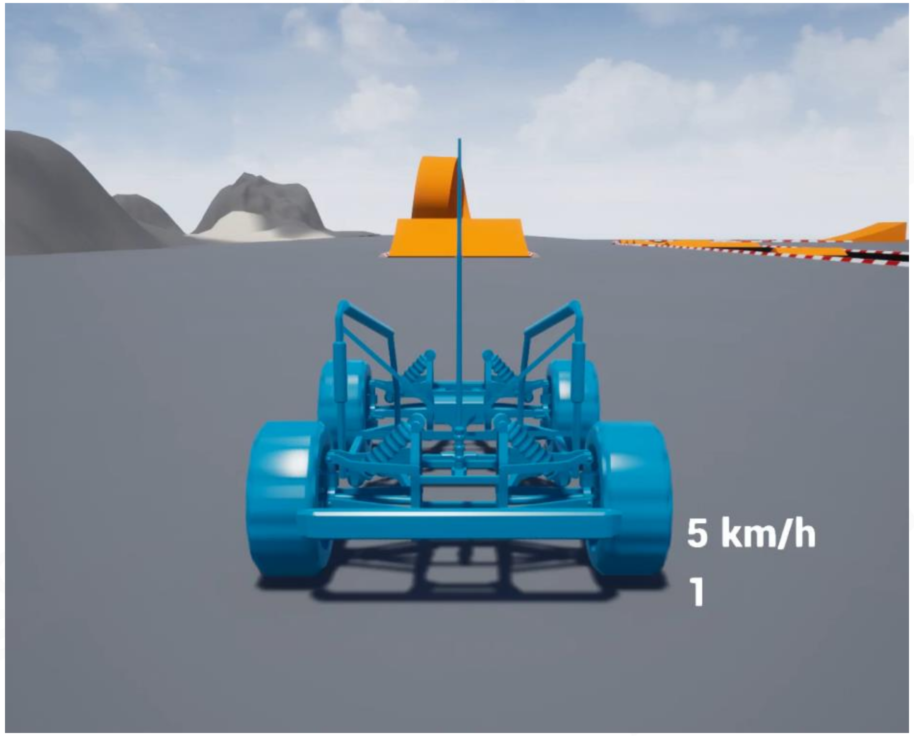
$$\vec{x}_{cm} = \frac{M_1\vec{x}_1 + M_2\vec{x}_2}{M}$$

Center of Mass (2/3)

Affects the stability of the vehicle in the air

- front-heavy -> dive
- rear-heavy -> stabilize

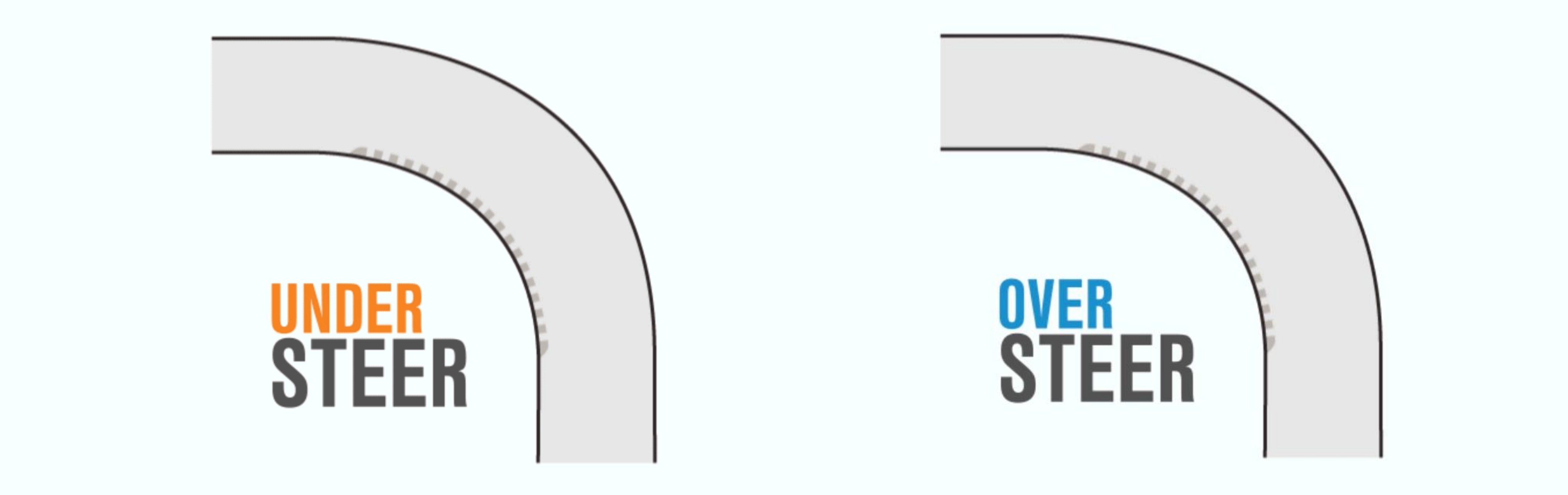




Center of Mass (3/3)

Affects vehicle steering control

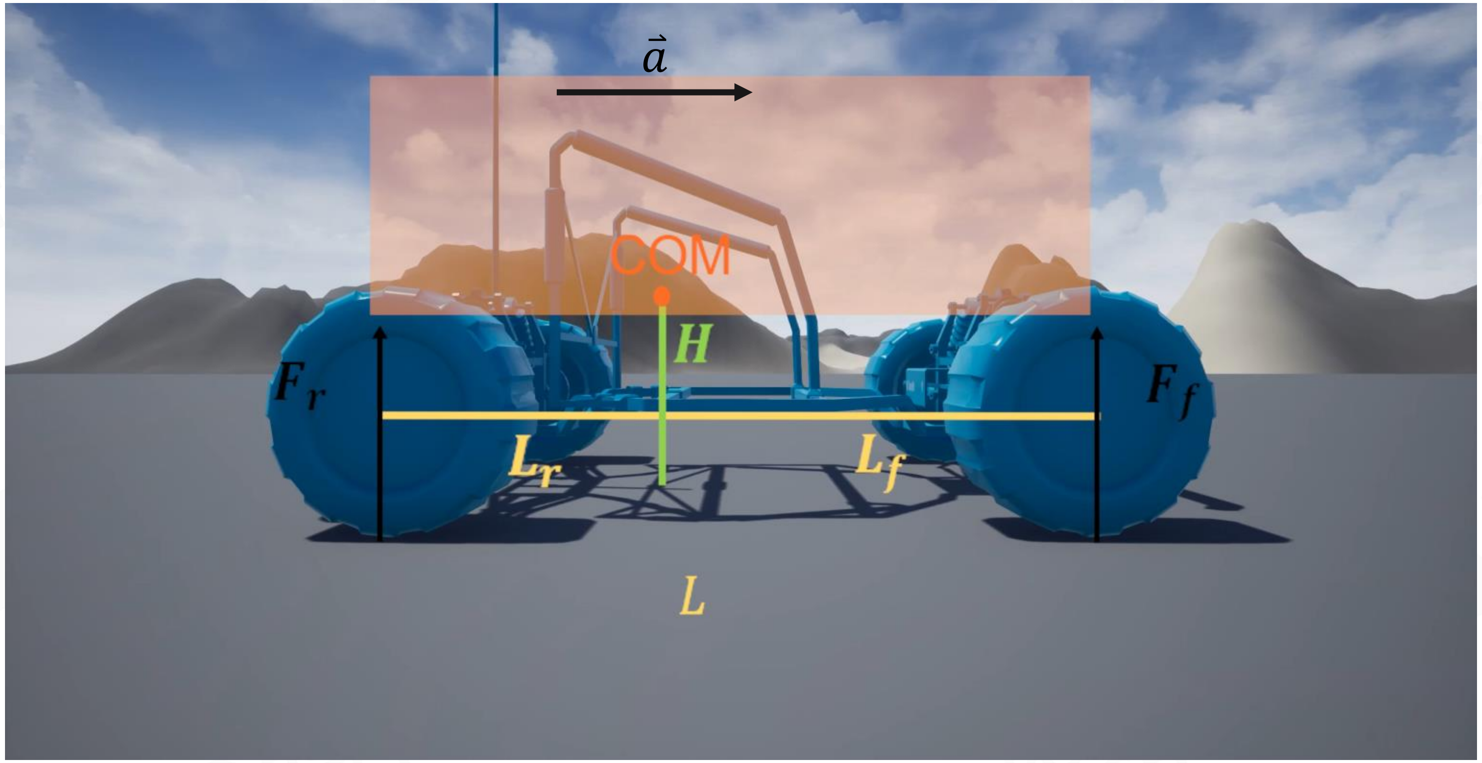
- front-heavy -> **understeering**
- rear-heavy -> **oversteering**



Weight Transfer

Affects the maximum traction force per wheel

- $\vec{F}_f = \frac{L_f}{L} M\vec{g} \mp \frac{H}{L} M\vec{a}$
- $\vec{F}_r = \frac{L_r}{L} M\vec{g} \pm \frac{H}{L} M\vec{a}$
- $\vec{F}_{traction} = \mu \vec{F}_{suspension}$



M: mass of vehicle
μ : friction coefficient of the tire

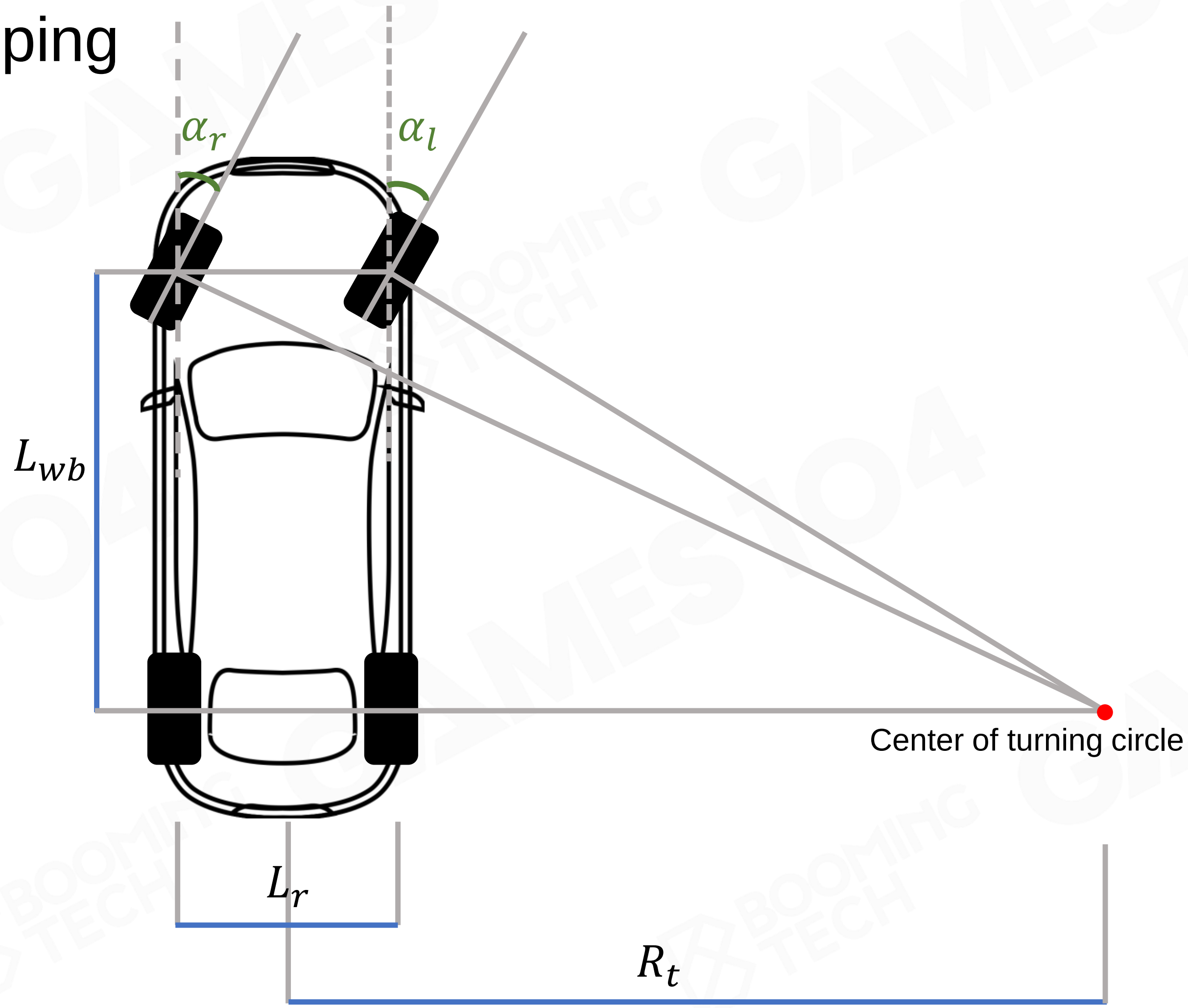
Steering angles (1/2)

- Same steering angle causes slipping
- Ackermann steering
  - different steering angles

$$\alpha_l = \tan^{-1}\frac{L_{wb}}{R_t + \frac{L_r}{2}}$$

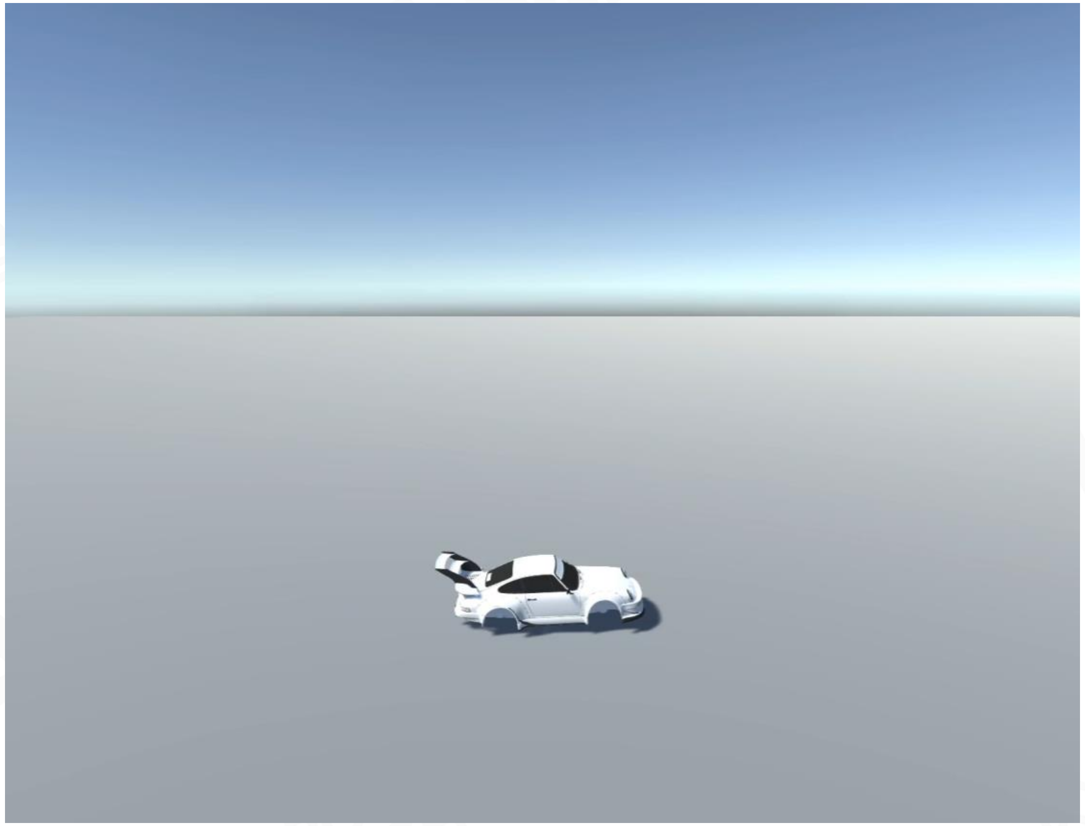
$$\alpha_r = \tan^{-1}\frac{L_{wb}}{R_t - \frac{L_r}{2}}$$

L_{wb}:length of wheel base
L_r :length of rear track
R_t :turn radius



Steering angles (2/2)

Without Ackermann steering



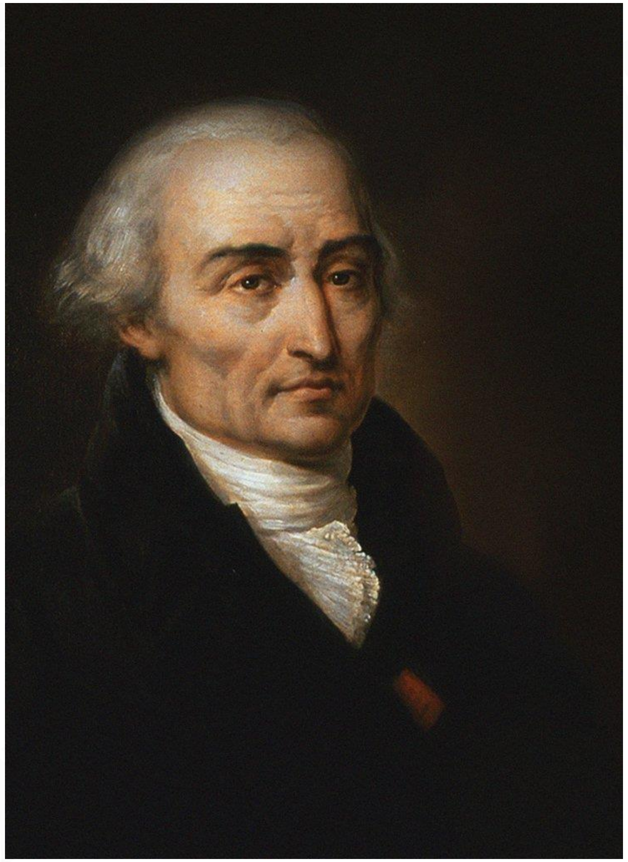
With Ackermann steering

Advanced Wheel Contact

Advanced: PBD/XPBD

Recap: Solving Constraints

- Modelling constraints based on Lagrangian mechanics
  - Collision constriaints
    - Non-penetration
    - Friction
    - Restitution
  - Cloth constraints
    - Stretching
    - Bending



Joseph-Louis Lagrange

(1736 - 1813)

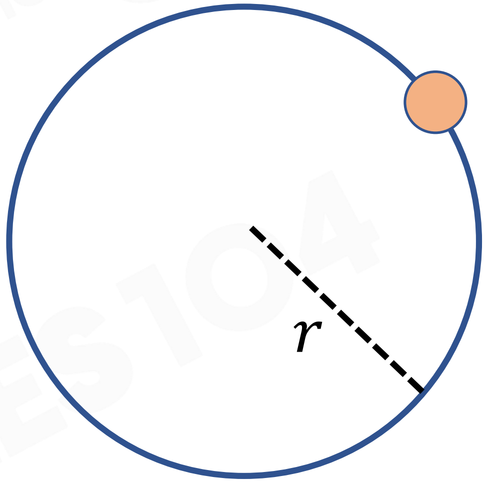
Circling Constraint

Position constraint $C(\mathbf{x}) = 0$

$$C(\mathbf{x}) = ||\mathbf{x}|| - r = 0$$

Velocity constraint $\frac{\mathrm{d}}{\mathrm{d}t}C(\mathbf{x}) = \frac{\mathrm{d}C}{\mathrm{d}\mathbf{x}}\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = 0$

$\frac{\mathrm{d}C}{\mathrm{d}\mathbf{x}}$: $\mathbf{J}$ $\quad$ $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$: $\mathbf{v}$



$\mathbf{J}$

Jacobian

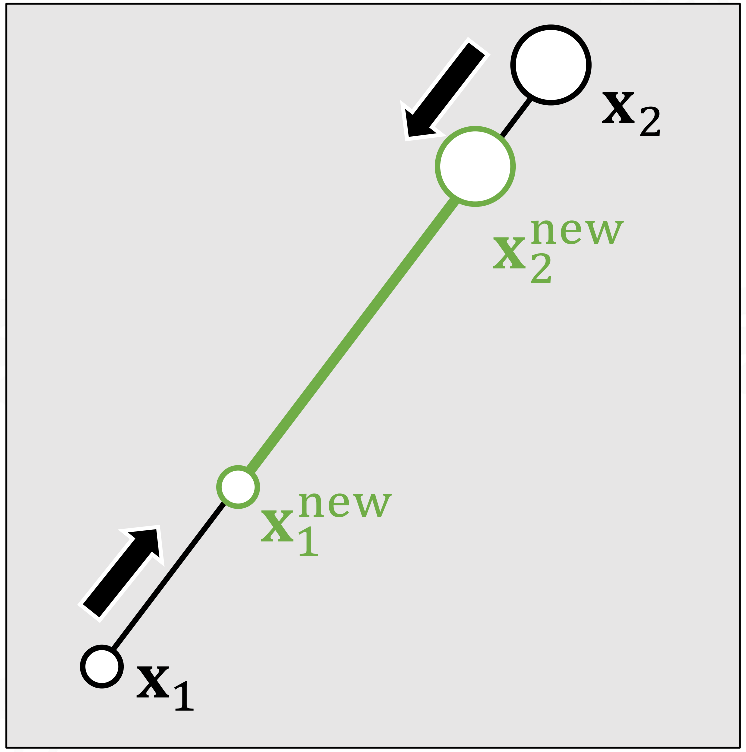
- Row Vector
- $\mathbf{J}^{\mathrm{T}}$ is perpendicular to $\mathbf{v}$
- Transforms velocity to velocity constraint

$$\mathbf{J}^{\mathrm{T}} \cdot \mathbf{v} = 0$$

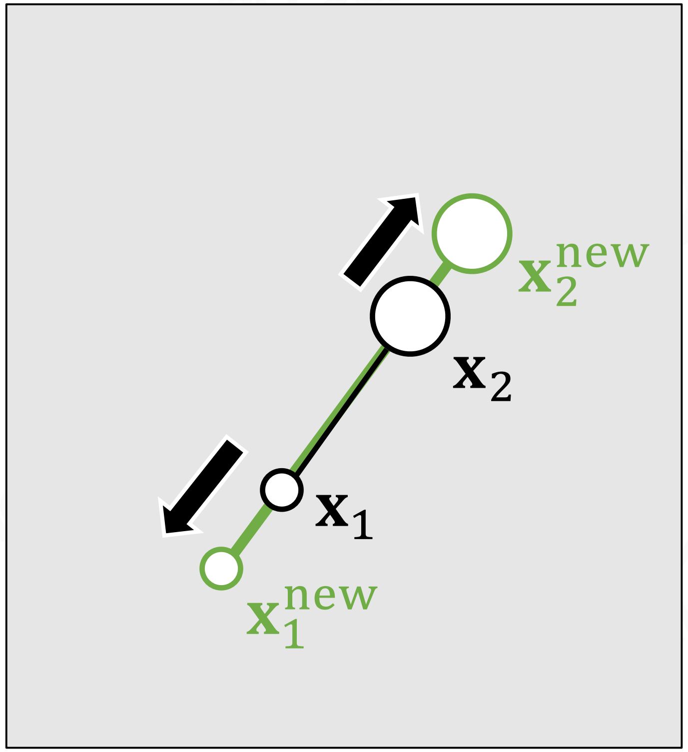
String Constraints

$$C_{stretch}(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\| - d$$

Stretched Case



Compressed Case

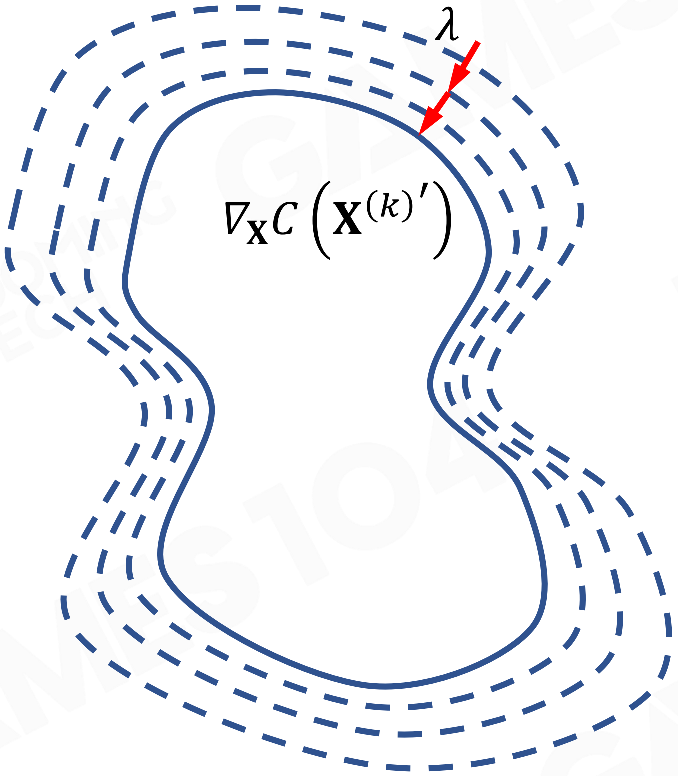


PBD – Constraints Projection

$$\mathbf{X}^{(k)'} = \begin{bmatrix} \mathbf{x}_1^{(k)'} \\ \vdots \\ \mathbf{x}_n^{(k)'} \end{bmatrix}$$

$k = 0$: Integrated positions

$k > 0$: position with correction from last iteration



$$C\left(\mathbf{X}^{(k)'} + \Delta\mathbf{X}\right) \approx C\left(\mathbf{X}^{(k)'}\right) + \nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right) \cdot \Delta\mathbf{X} = 0$$

$$\Delta\mathbf{X} = \lambda \nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right)$$

PBD – Constraints Projection

$$C\left(\mathbf{X}^{(k)'} + \Delta\mathbf{X}\right) \approx C\left(\mathbf{X}^{(k)'}\right) + \nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right) \cdot \Delta\mathbf{X} = 0$$

$$\Delta\mathbf{X} = \lambda \nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right)$$

$$C\left(\mathbf{X}^{(k)'}\right) + \nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right) \cdot \lambda \nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right) = 0$$

$$\lambda = -\frac{C\left(\mathbf{X}^{(k)'}\right)}{\left\|\nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right)\right\|^2} \qquad \Delta\mathbf{X} = -\frac{C\left(\mathbf{X}^{(k)'}\right)}{\left\|\nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right)\right\|^2} \nabla_{\mathbf{X}} C\left(\mathbf{X}^{(k)'}\right)$$

Position Based Dynamics – Workflow (1/6)

forall vertices i
  initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
endfor
loop
  forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
  dampVelocities$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
  forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  forall vertices i **do** generateCollisionConstraints$(\mathbf{x}_i \rightarrow \mathbf{p}_i)$
  loop solverIterations **times**
    projectConstraints$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N)$
  endloop
  forall vertices i
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  endfor
  velocityUpdate$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
endloop

Time step

Position Based Dynamics – Workflow (2/6)

forall vertices i
    initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
endfor
loop
    forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
    dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
    forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
    forall vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
    loop solverIterations **times**
        projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
    endloop
    forall vertices i
        $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
        $\mathbf{x}_i \leftarrow \mathbf{p}_i$
    endfor
    velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
endloop

Semi-implicit integration

Position Based Dynamics – Workflow (3/6)

forall vertices i
  initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
endfor
loop
  forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
  dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
  forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  forall vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
  loop solverIterations **times**
    projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
  endloop
  forall vertices i
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  endfor
  velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
endloop

- For collisions in this time step, generate constraints
- Structural constraints are initialized when starting the simulation

Position Based Dynamics – Workflow (4/6)

forall vertices i
  initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
endfor
loop
  forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
  dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
  forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  forall vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
  loop solverIterations **times**
    projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
  endloop
  forall vertices i
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  endfor
  velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
endloop

Solver interation

Position Based Dynamics – Workflow (5/6)

forall vertices i
  initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
endfor
loop
  forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
  dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
  forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  forall vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
  loop solverIterations **times**
    projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
  endloop
  forall vertices i
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  endfor
  velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
endloop

Update states after solver iterations

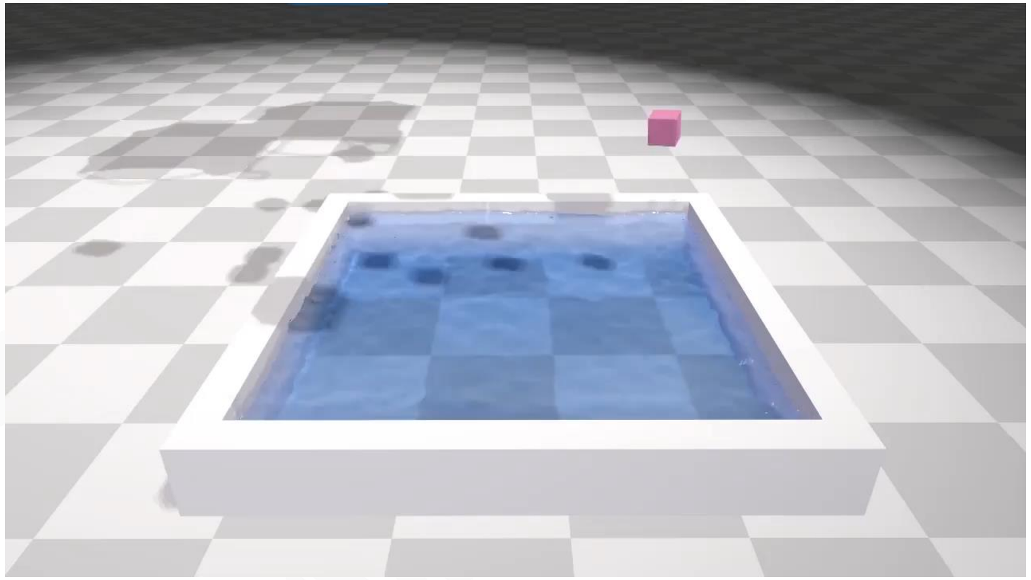
Position Based Dynamics – Workflow (6/6)

forall vertices i
    initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
endfor
loop
    forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
    dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
    forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
    forall vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
    loop solverIterations **times**
        projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
    endloop
    forall vertices i
        $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
        $\mathbf{x}_i \leftarrow \mathbf{p}_i$
    endfor
    velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
endloop

the velocities of colliding vertices are modified according to friction and restitution coefficients

Advantages of PBD

- Projecting constraints to position corrections
- Fast, stable for most cases
- Hard to control constraint satisfaction
  - Cannot prioritize collision constraints over others
- Commonly used for cloth simulation in games
- NVIDIA FleX

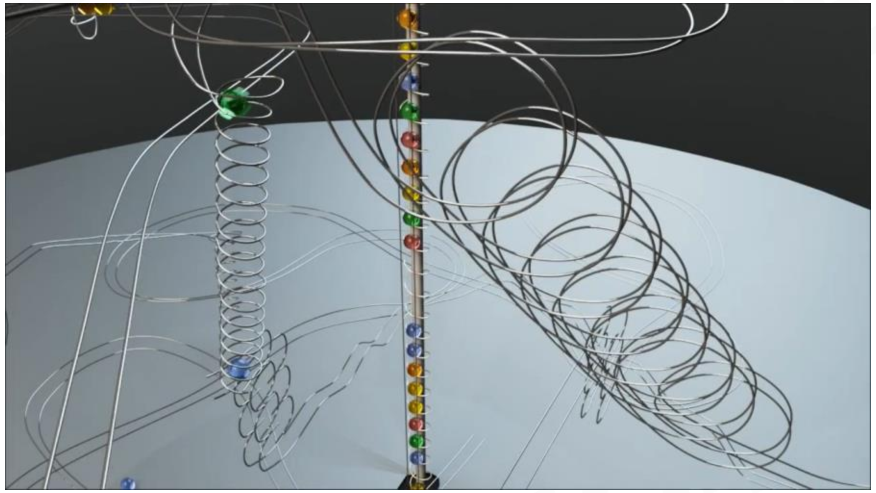


Extended Position Based Dynamics (XPBD)

- Use compliances as inverse of constraint stiffness to handle infinite stiffness constraints (rigidbody)
- Reintroduce rigidbody orientation to XPBD for rigidbody simulation application
- Unreal Chaos physics engine

$$U(\mathbf{X}) = \frac{1}{2}\boldsymbol{C}(\mathbf{X})^{\mathrm{T}}\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\mathbf{X})$$

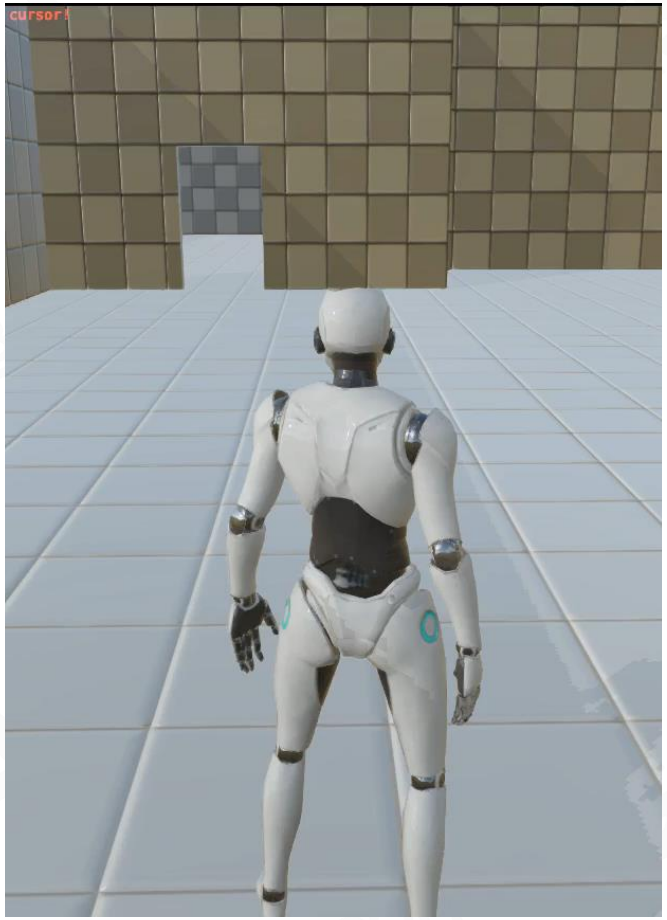
Block diagonal compliance matrix



Homework 3 (available since 1 June)

- You are supposed to...
  - Implement pose blend function in animation system
  - Implement a simple logic of ASM
  - Implement a simple character controller based on physics scene queries
  - Add more features to the character controller that you like (advanced)
  - Write a report document that contains screenshots of your results

- Download at
  - Course-site: https://games104.boomingtech.com/sc/course-list
  - Github: https://github.com/BoomingTech/Pilot/tree/games104/homework03-animation-physics



References

Character Controller & Ragdoll

- Character Controllers Chapter in PhysX User's Guide: https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/CharacterControllers.html
- Diablo 3 Ragdolls: How to smack a demon, Erin Catto, GDC 2012: https://box2d.org/files/ErinCatto_Ragdolls_GDC2012.pdf
- Physics Animation in 'Uncharted 4: A Thief's End', Michal Mach, Naughty Dog, GDC 2017: https://gdcvault.com/play/1024087/Physics-Animation-in-Uncharted-4
- Physics Driven Ragdolls and Animation at EA: From Sports to Star Wars, Jalpesh Sachania, Frostbite EA, GDC 2018: https://gdcvault.com/play/1025210/Physics-Driven-Ragdolls-and-Animation
- Physical Animation in 'Star Wars Jedi: Fallen Order', Bartlomiej Waszak, Respawn Entertainment, GDC Summit 2020: https://gdcvault.com/play/1026848/Physical-Animation-in-Star-Wars

Clothing

- Blowing from the West: Simulating Wind in 'Ghost of Tsushima', Bill Rockenbeck, Sucker Punch Productions, GDC 2021: https://www.gdcvault.com/play/1027124/Blowing-from-the-West-Simulating
- Cloth Self Collision with Predictive Contacts, Chris Lewin, Electronic Arts, GDC 2018: https://www.gdcvault.com/play/1025083/Cloth-Self-Collision-with-Predictive
- Machine Learning: Physics Simulation, Kolmogorov Complexity, and Squishy Bunnies, Daniel Holden, Ubisoft Montreal, GDC 2020: https://www.gdcvault.com/play/1026713/Machine-Learning-Physics-Simulation-Kolmogorov
- Matt's Webcorner - Cloth - Stanford Computer Graphics, Stanford 2014 Course, https://graphics.stanford.edu/~mdfisher/cloth.html
- 从零开始学图形学：弹簧质点系统——Euler Method和Verlet Integration, 启思, 知乎专栏 https://zhuanlan.zhihu.com/p/355170943

Destruction

- 游戏破坏系统简介，网易游戏雷火事业群，知乎 https://zhuanlan.zhihu.com/p/346846195
- Destructible Environments in 'Control': Lessons in Procedural Destruction, Johannes Richter, Remedy, GDC Summer 2020, https://www.gdcvault.com/play/1026820/Destructible-Environments-in-Control-Lessons
- The Art of Destruction in 'Rainbow Six: Siege', Julien L'Heureux, Ubisoft, GDC 2016, https://www.gdcvault.com/play/1023307/The-Art-of-Destruction-in
- NVIDIA Blast official site, https://developer.nvidia.com/blast
- Voronoi Diagram, https://cs.brown.edu/courses/cs252/misc/resources/lectures/pdf/notes09.pdf
- Delaunay Triangulations, https://members.loria.fr/monique.teillaud/collab/Astonishing/2017_workshop_slides/Olivier_Devillers.pdf
- Unreal Engine Chaos, https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Physics/ChaosPhysics/Overview/

Vehicle

- Vehicle Chapter in PhysX User's Guide: https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/CharacterControllers.html
- Vehicle in Unreal Engine User's Guide: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Vehicles/VehicleUserGuide/
- Car Physics for Games, Marco Monster: https://asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html
- Replicating Chaos: Vehicle Replication in Watch Dogs 2, Matt Delbosc, Ubisoft Toronto, GDC 2017: https://www.gdcvault.com/play/1026956/Replicating-Chaos-Vehicle-Replication-in

PBD

- Positon Based Dynamics, M. Müller et al., 3rd Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS“, 2006: https://matthias-research.github.io/pages/publications/posBasedDyn.pdf
- XPBD: Position-Based Simulation of Compliant Constrained Dynamics, M. Macklin et al., MIG '16: Proceedings of the 9th International Conference on Motion in Games, 2016: http://mmacklin.com/xpbd.pdf
- Detailed Rigid Body Simulation using Extended Position Based Dynamics, M. Müller et al., Symposium on Computer Animation, 2020: https://www.researchgate.net/publication/344464310_Detailed_Rigid_Body_Simulation_using_Extended_Position_Based_Dynamics
- Position Based Dynamics: A fast yet physically plausible method for deformable body simulation, Tiantian Liu, GAMES Webinar 2019-88: https://slides.games-cn.org/pdf/Games201988%E5%88%98%E5%A4%A9%E6%B7%BB.pdf

Lecture 11 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