05 - Physics and Collision Systems

CS 3160 - Game Programming

Max Gilson

Physics and Collision Systems

- The math behind creating a realistic physics system is “easy”
  - We know Newtonian physics F=ma, differential calculus, and integral calculus
- Calculating a perfectly realistic physics system is too computationally intensive for real time applications (games)
  - The challenge of a physics system is simulating realistic physics while reducing computational load as much as possible
  - We attempt to achieve a believable approximation of physics

Physics and Collision Systems

- It is possible to calculate all the physics in this scene entirely using perfect Newtonian physics
  - Lighting
  - Smoke
  - Fire
  - Air
  - Destructible elements
- It would take years to simulate without any approximations
  - Even more complicated in a massive open world!

Lighting

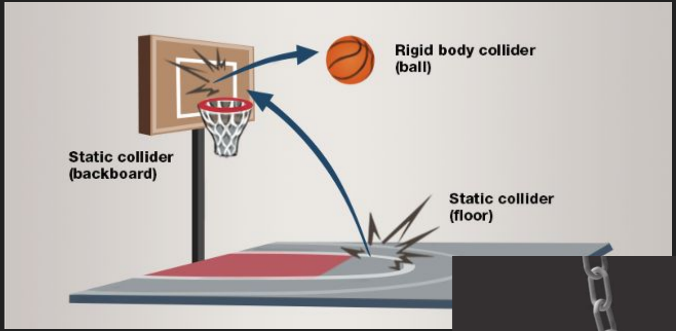
- Lights are physics too!
    - Assume a single light bulb emits 302 quintillion photons per second
    - Assume your runs at CPU has 5 billion clock cycles per second
    - You see the issue
- In games, lighting is a graphics issue, not a physics issue

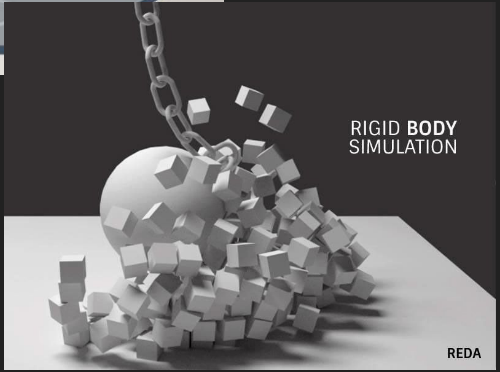
Uses for Physics

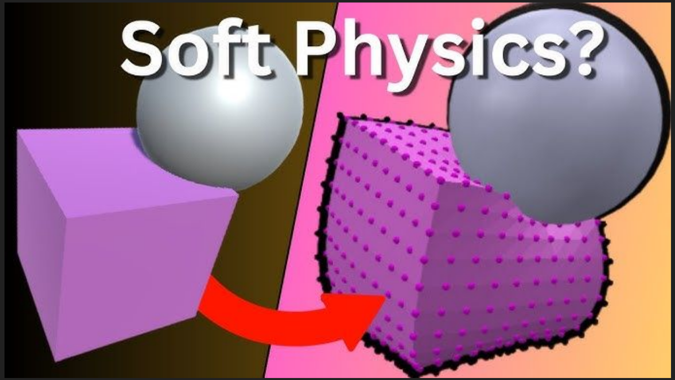
- Physics are not just to add realism to a game
- Physics give the player feedback and interaction
  - If an explosion occurs, you want the objects in the environment to react accordingly
  - If the player can wall jump, collision detection is necessary
- Physics can result in emergent gameplay
  - Emergent gameplay is the unplanned behavior that results from the interaction of a game's systems

Types of Physics

- Rigidbody physics allows for collisions between objects without deforming the objects
  - Provided by most game engines by default
- Softbody physics allows for the deformation of objects
  - Everything from marshmallows to car crashes
  - Can be implemented in Unity but is not available by default

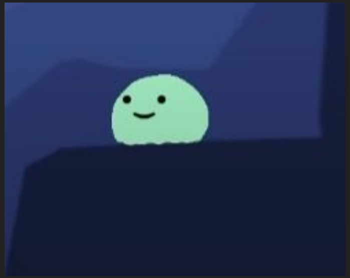






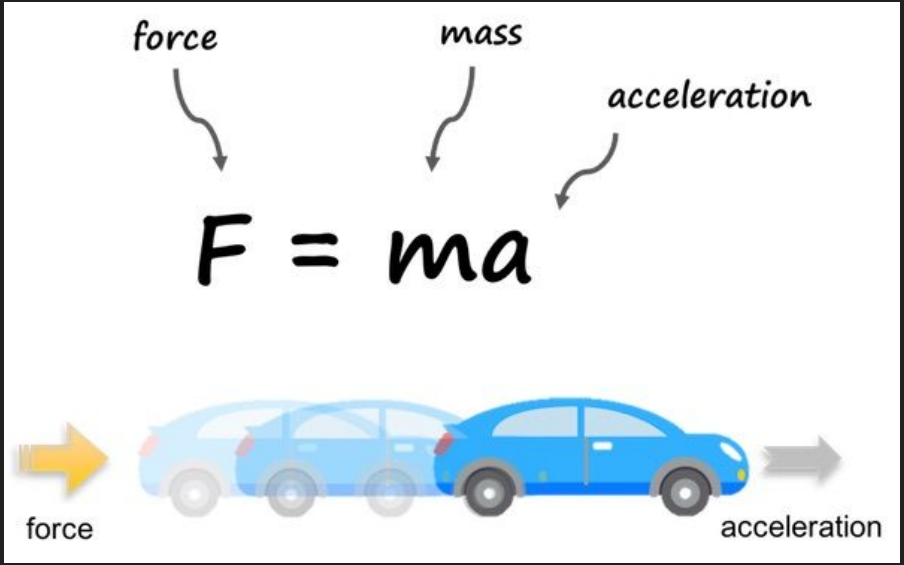
Softbody Physics in Unity

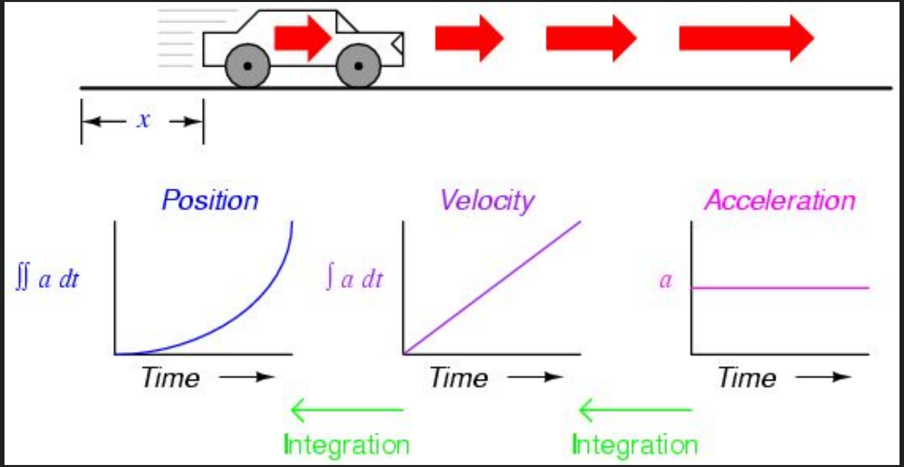
- You can implement softbody physics in Unity with some work:
    - 2D: https://www.youtube.com/watch?v=3avaX00MhYc
    - 2D: https://www.youtube.com/watch?v=y1D4DiZhSlo
    - 3D: https://www.youtube.com/watch?v=1rYCqsEj3dU



Rigidbody Physics

- We will focus on Rigidbody physics since Softbody physics are less common and harder to implement
- Force = mass x acceleration
  - When in 3D, force and acceleration are 3D vectors
- Recall the game loop and the physics loop: if our physics loop runs at a fixed frequency of 50 Hz, then delta time (dt) is 0.02 seconds:
  - Position = initial position + velocity * dt
  - Velocity = initial velocity + acceleration * dt
  - Acceleration = force / mass
- This works well for single bodies



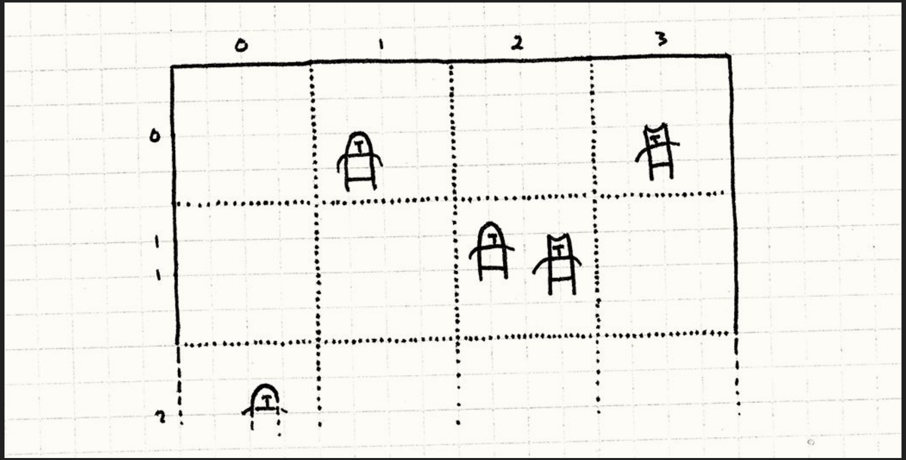


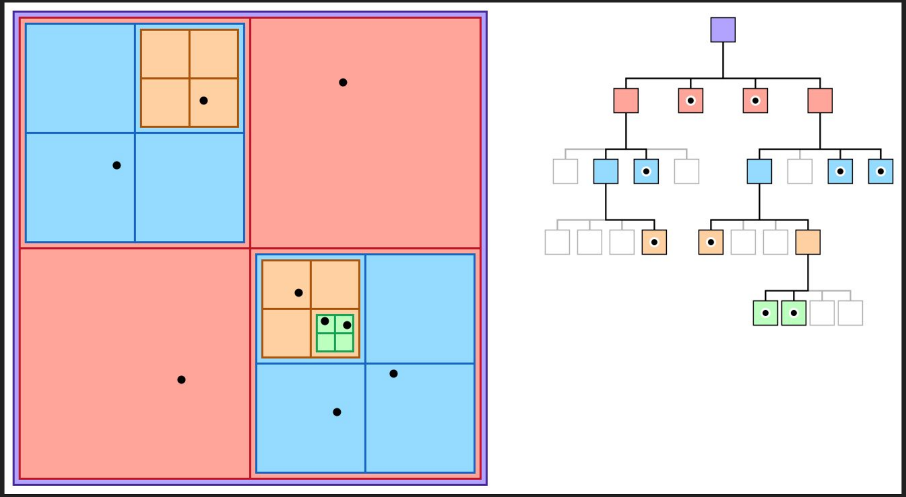
Rigidbody Physics Collisions

- The previous equations work fine but we need to consider collisions
  - This is the hard part of physics simulations
- How do you know if two things are colliding with each other?
  - Imagine two circles
  - The naive approach is to just compare the distance of two objects
    - This is way too slow
- Broad phase: quickly eliminate pairs of objects that definitely won’t collide
- Narrow phase: accurately test for collisions between objects identified in the broad phase

Simple Broad Phase

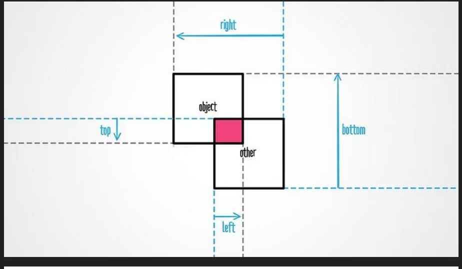
- Spatial partitioning is an easy way to implement a broad phase for a 2D collision system
- One spatial partitioning algorithm uses a hash table which has O(1) lookup
  - Each key of the hash table is a coordinate block (x, y)
  - Each value of the hash table is a set of physics objects
  - Compare distances between all objects in the set to see if they collide
  - If two objects exist at different coordinate blocks, they are not checked for collisions!
- Better methods use quadtrees or Kd-trees which have coordinates that can scale in size to improve performance

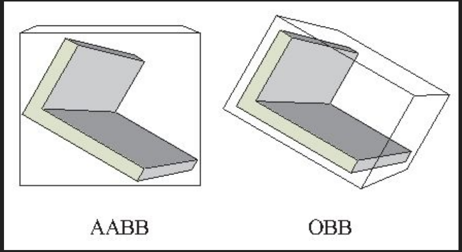


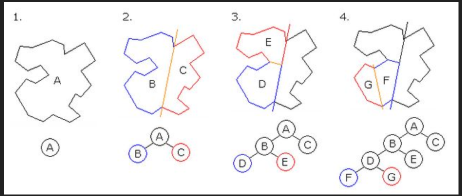


More Broad Phase

- There are many algorithms for broad phase collision detection
- 2D:
  - Axis-Aligned Bounding Box (AABB)
  - Quadtrees
- 3D:
  - Oriented Bounding Boxes (OBB)
  - Octrees
  - Binary Space Partitioning (BSP) Trees





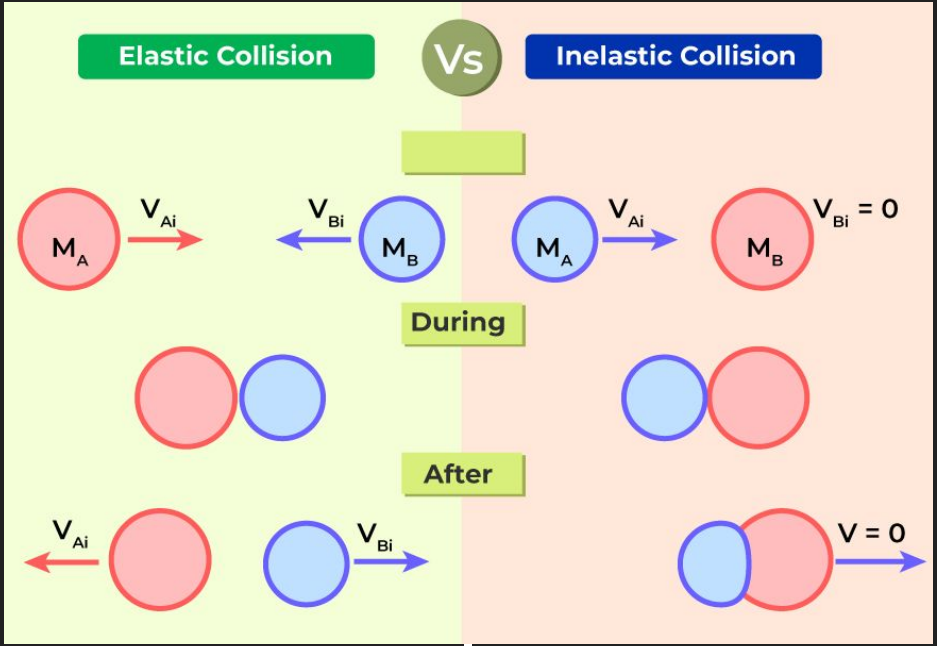


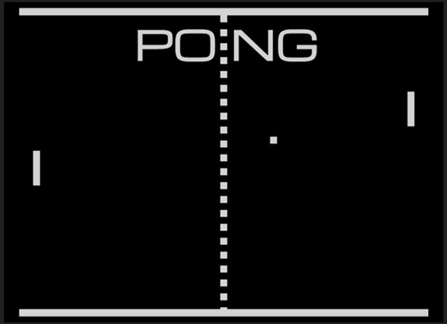
Narrow Phase

- Once you have isolated the physics objects that might actually collide, the narrow phase starts
- Performing a deeper check might be necessary for complex objects (non-spheres/non-circles)
- If two objects are colliding, how should this be resolved?
  1. Calculate how far the objects collided into each other
  2. Move objects apart
  3. Calculate translational and rotational velocities resulting from impact

Collision Response

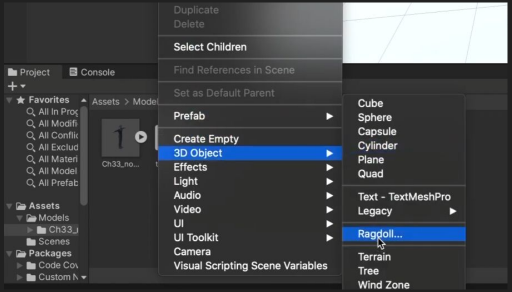
- Once a collision has occurred, the physics system must calculate what happens next
- Elastic collision
  - Energy is perfectly conserved
  - Pong
- Inelastic collision
  - Energy is lost due to friction, sound, heat, etc.
  - Any collision that does not involve a perfect bounce
    - Racing games slowing cars down when scraping each other

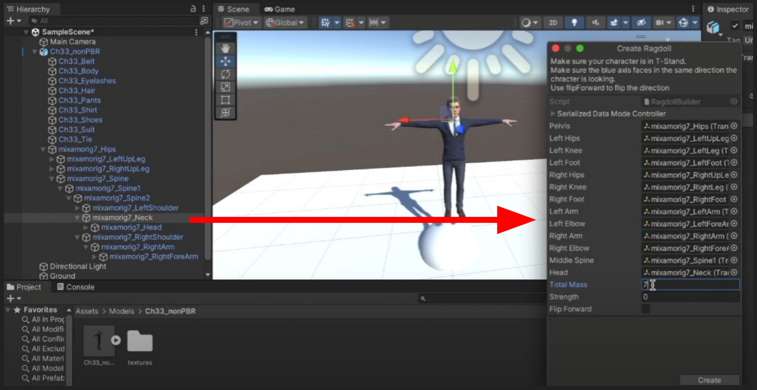




Ragdolls

- Ragdolls are multiple physics objects or rigidbodies tied together
  - This can be used to simulate the flailing of limbs
- Active ragdolls allow the player to control the movement of the ragdoll
  - Challenging to make look realistic





Create Ragdoll

Make sure your character is in T-Stand.
Make sure the blue axis faces in the same direction the chracter is looking.
Use flipForward to flip the direction

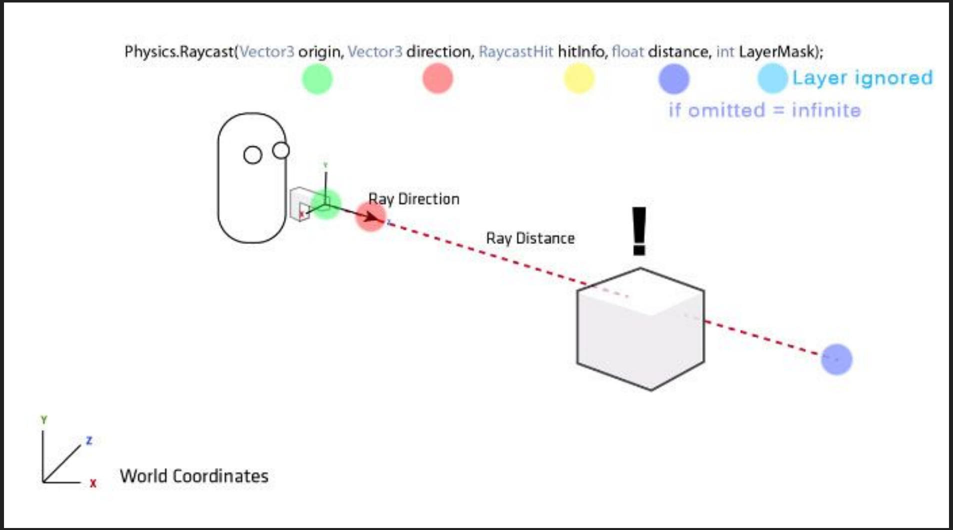
Script	RagdollBuilder
Pelvis	mixamorig7_Hips (Tran
Left Hips	mixamorig7_LeftUpLeg
Left Knee	mixamorig7_LeftLeg (T
Left Foot	mixamorig7_LeftFoot (T
Right Hips	mixamorig7_RightUpLe
Right Knee	mixamorig7_RightLeg (
Right Foot	mixamorig7_RightFoot
Left Arm	mixamorig7_LeftArm (T
Left Elbow	mixamorig7_LeftForeAr
Right Arm	mixamorig7_RightArm (
Right Elbow	mixamorig7_RightFore/
Middle Spine	mixamorig7_Spine1 (Tr
Head	mixamorig7_Neck (Tra
Total Mass	7
Strength	0
Flip Forward	

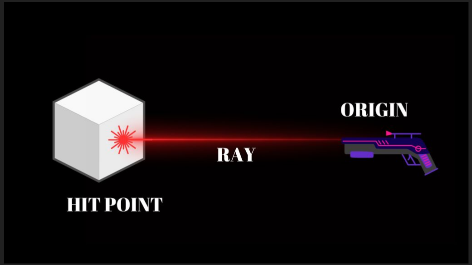
Determinism

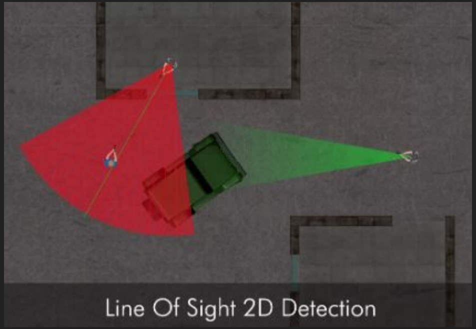
- If something is deterministic, then given the same input and initial state, a system will always produce the same output
    - No randomness or variation
- Very few physics simulations are deterministic
    - Making a deterministic physics engine is very hard and performance heavy
- Deterministic modes may be available but have performance or feature trade offs
    - Example: a physics simulation might be multithreaded behind the scenes and the multithreading causes it to be non-deterministic
        - Is determinism worth sacrificing multithreading performance?

Raycasts

- Raycasts are a way to detect a collision along a line in a specific direction from an origin
    - Shooting without a physics projectile
    - Line of sight for enemy detecting player


Physics.Raycast(Vector3 origin, Vector3 direction, RaycastHit hitInfo, float distance, int LayerMask);
Layer ignored
if omitted = infinite
Ray Direction
Ray Distance
!
Y
Z
X
World Coordinates





Raycasts in Unity

- You can perform raycasts in Unity by providing an origin, a direction, and a distance
- Physics.RayCast() returns true if the ray hit an object or false if not
- The RaycastHit type gives you the result of the raycast, if any

Weapon.cs:

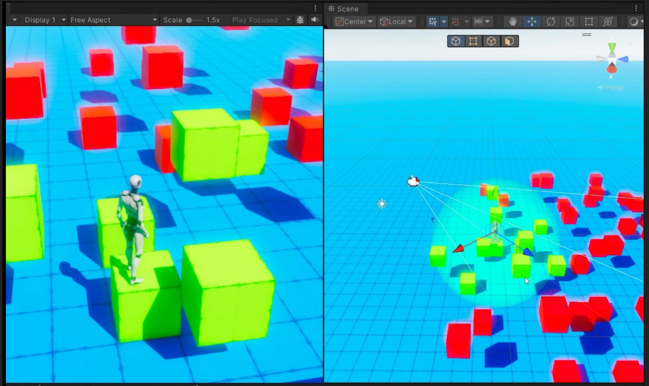
```csharp
using UnityEngine;

public class Weapon : MonoBehaviour
{
    private bool firePressed = false;
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            firePressed = true;
        }
    }

    void FixedUpdate()
    {
        if (firePressed)
        {
            RaycastHit hit;
            var origin = transform.position;
            var direction = transform.forward;
            var distance = 10f;
            var layerMask = LayerMask.GetMask("Wall", "Character");
            if (Physics.Raycast(origin, direction, out hit, distance, layerMask))

            {
                Debug.DrawRay(origin, direction * hit.distance, Color.red);
                Debug.Log($"Raycast Hit: {hit.transform.name}");
            }
            else
            {
                Debug.DrawRay(origin, direction * distance, Color.white);
                Debug.Log("Did not Hit");
            }
        }
    }
}
```

Physics Overlap

- To check if something is within a sphere's radius without wanting to use a collider you can use a physics overlap in Unity
  - Separate GameObjects for each sphere are not required
- Could be used for explosions or highlight objects nearby the player to pick up

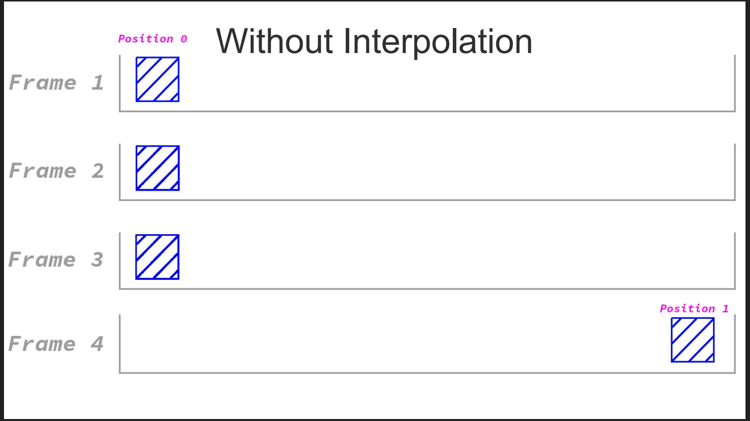
Explosion.cs:

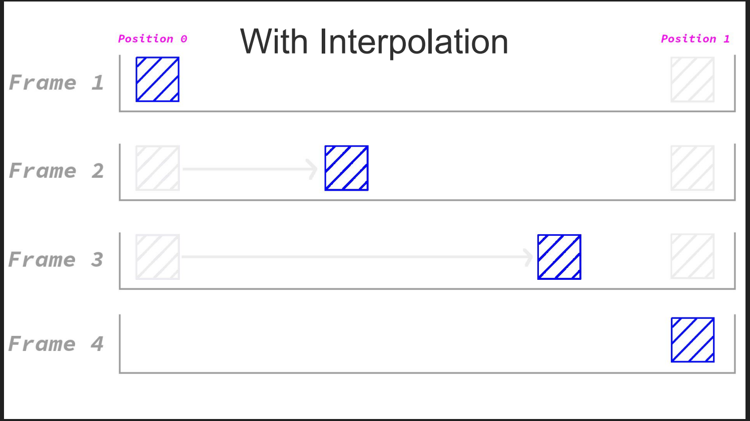
```
using UnityEngine;
using System.Collections;

public class Explosion : MonoBehaviour
{
    void ExplosionDamage(Vector3 center, float radius)
    {
        Collider[] hitColliders = Physics.OverlapSphere(center, radius);
        foreach (var hitCollider in hitColliders)
        {
            var hitGameObject = hitCollider.gameobject;
            // Apply damage to the gameobject here
        }
    }
}
```



Interpolation and Extrapolation

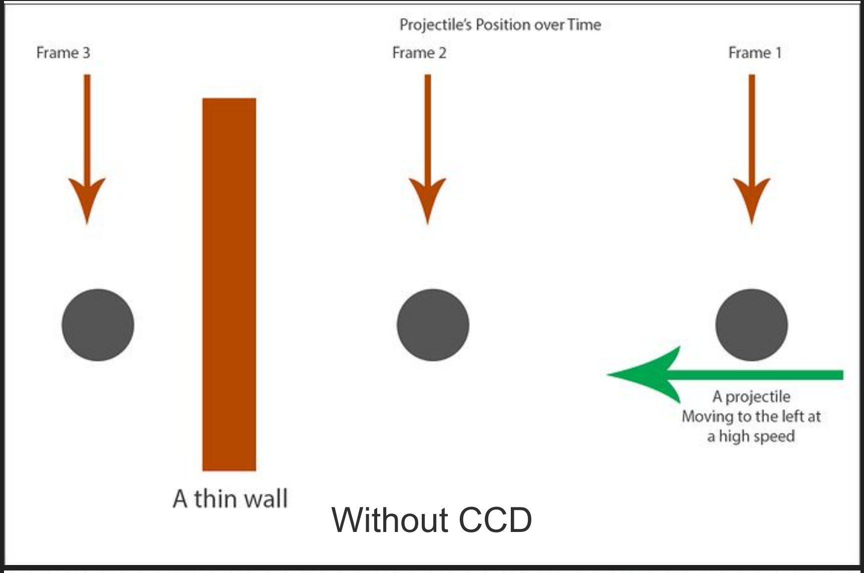
- If the physics timesteps are fixed, won't it look stuttery if the frame rate is really high?
    - Yes! You need to interpolate your physics objects
- Interpolation calculates the visual position between physics timesteps
    - Makes the motion smoother
    - Matches the frame rate of the game
    - Only updates visuals, does not update the physics system
- Interpolation lags behind by 1 physics timestep because you need to calculate the end position
- Extrapolation is another technique that does not wait for the end position, insteads estimates the visuals up to the estimated end position

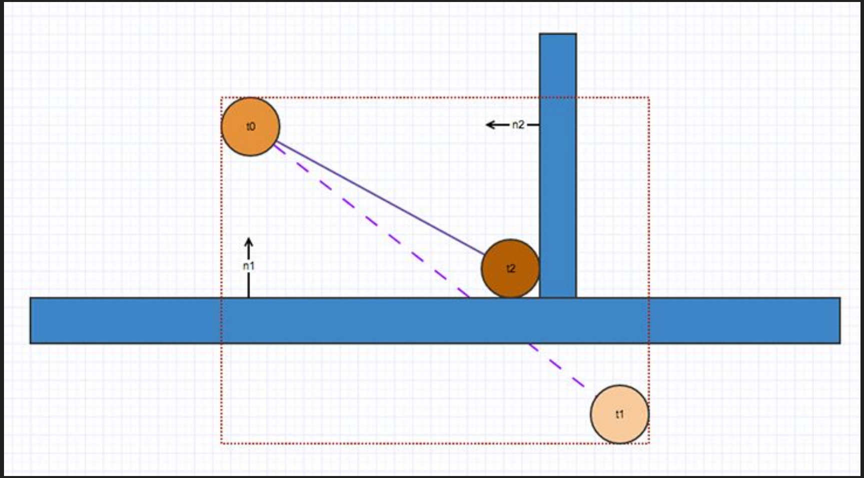




Continuous Collision Detection

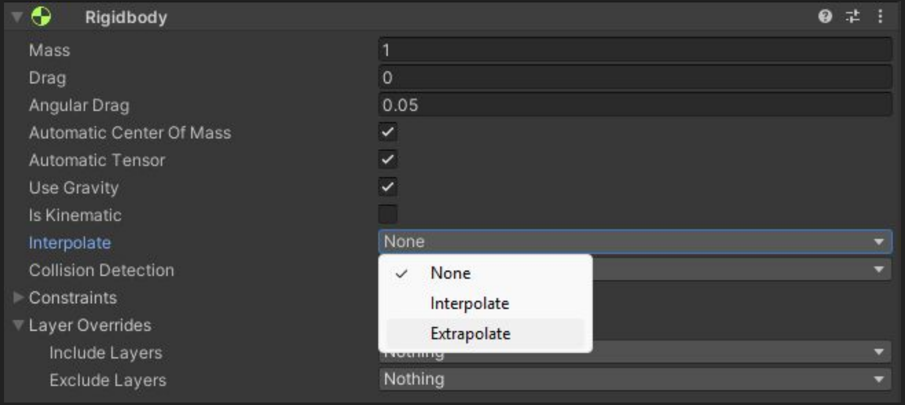
- Continuous Collision Detection (CCD) is a technique that improves physics simulations
  - Very bad for performance
  - Absolutely required for small and/or fast moving objects
- https://docs.unity3d.com/2023.1/Documentation/Manual/ContinuousCollisionDetection.html


Projectile's Position over Time
Frame 3
Frame 2
Frame 1
A projectile Moving to the left at a high speed
A thin wall
Without CCD



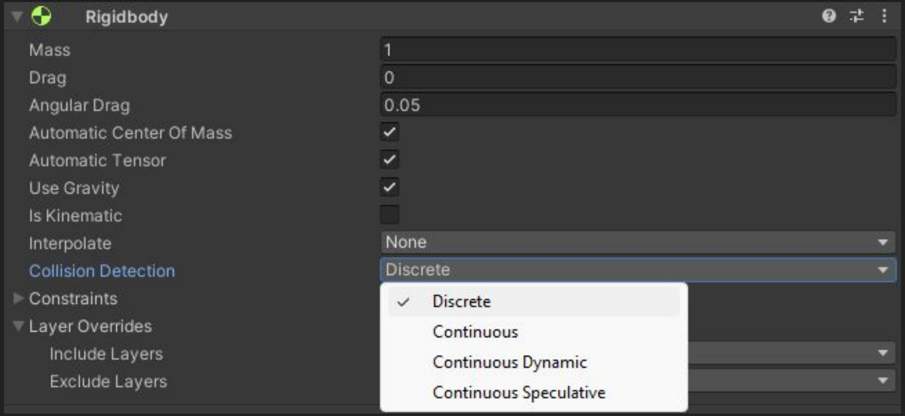
Interpolate, Extrapolate, and CCD in Unity

- Interpolating, extrapolating, and CCD are provided via a check box in Unity
- Be aware of the performance implications



Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Automatic Center Of Mass	✓
Automatic Tensor	✓
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	
Constraints	
Layer Overrides	
Include Layers	Nothing
Exclude Layers	Nothing



Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Automatic Center Of Mass	✓
Automatic Tensor	✓
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete
Constraints	
Layer Overrides	
Include Layers	
Exclude Layers	

Physics Libraries

- Most game developers or game programmers use off-the-shelf physics engines
    - Or use a game engine that has a physics engine built-in (i.e. Unity, Unreal, Godot)
- Writing a physics system from scratch takes a lot of time and technical skill
- Usually cheaper, easier, and less risky to go with a well known physics engine
    - https://www.havok.com/havok-powered/
- Regardless, as a game programmer you need to understand when/where/how to use the physics engine in your games!