Physics in Unity

Movement, collision and forces

Programming – Game Development Foundations

Last modified 10/12/15 by Richard Taylor

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Contents

- What is “physics” in a game?
- Collider and Rigidbody components
- Static vs. dynamic colliders
- Implementation considerations
- Interpolation

What is “physics” in a game?

- In a game, “physics” refers to the rules used when moving objects around.
  - Detecting and resolving collisions
  - Applying forces

- Game physics are not “realistic”.
  - They’re an approximation
  - Performance is more important than accuracy

What is “physics” in a game?

- Physics systems in games have become more advanced as CPU power has increased.

- Implementation of physics has changed over time.
  - Older games used custom physics/collision systems
  - Newer games often use generic libraries
    - PhysX, Havoc, Bullet, Box2D

Example: Super Mario Bros.

- In Super Mario Brothers there is a very simple physics system running
  - Mario has velocity
  - Mario experiences acceleration
  - There is gravity
  - Friction
  - Mario slides as well as the turtle shells

"NES Super Mario Bros" by Source. Licensed under Fair use via Wikipedia - https://en.wikipedia.org/wiki/File:NES_Super_Mario_Bros.png#/media/File:NES_Super_Mario_Bros.png

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Modern physics systems

- Modern games often use a general purpose physics system.

- In addition to collision detection and resolution, these often provide features such as:
  - Rigidbody dynamics
  - Joints
  - Cloth simulation
  - Ragdoll effects

Unity's Physics System

- Unity uses NVIDIA's PhysX library for 3D physics.
  - We will focus on this.

- Also has a 2D physics system.
  - Very similar in principle.

Unity's Physics System

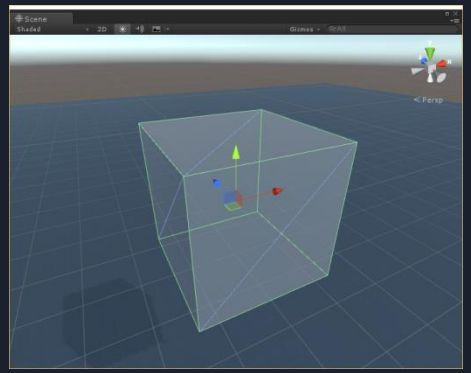
- There are two main components to be aware of:
  - Collider
  - Rigidbody

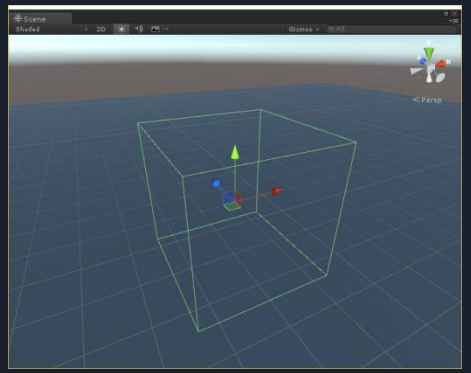
The Collider Component

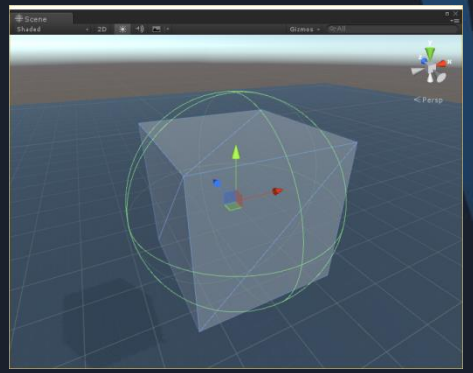
- Collider components define the shape of a body.

- They are separate from the visible part of a GameObject.
  - Collider does not have to match the visible shape.
  - You can have a Collider without a Renderer, or a Renderer without a Collider.

The Collider Component







The Collider Component

- Primitive colliders are relatively cheap to use.
  - Available in different shapes:
    - Box
    - Sphere
    - Capsule
  - Properties are available to refine their shapes.

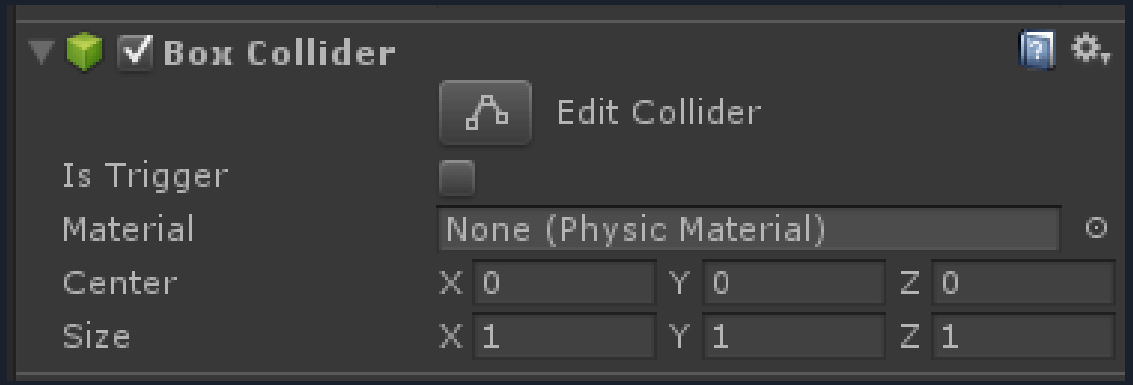
- MeshColliders are more expensive.
  - Consider using multiple primitive colliders instead of a MeshCollider.
  - Also have some restrictions. See the manual.

The Collider Component

- A Collider can be marked as a “trigger” by setting “isTrigger” to true.

- When a trigger is touched there is no “collision”. Instead, the OnTriggerEnter(...) function is called in attached scripts..
  - Great for pick ups, or triggering events.
  - Eg: You can use a trigger to open a door.

The Collider Component



The Rigidbody Component

- The Rigidbody component defines how a body moves, and allows forces to be applied to a body.
  - Properties which determine how forces behave.

- If a GameObject has a Collider and will be moved, then it should also have a Rigidbody.
  - We will come back to this later!

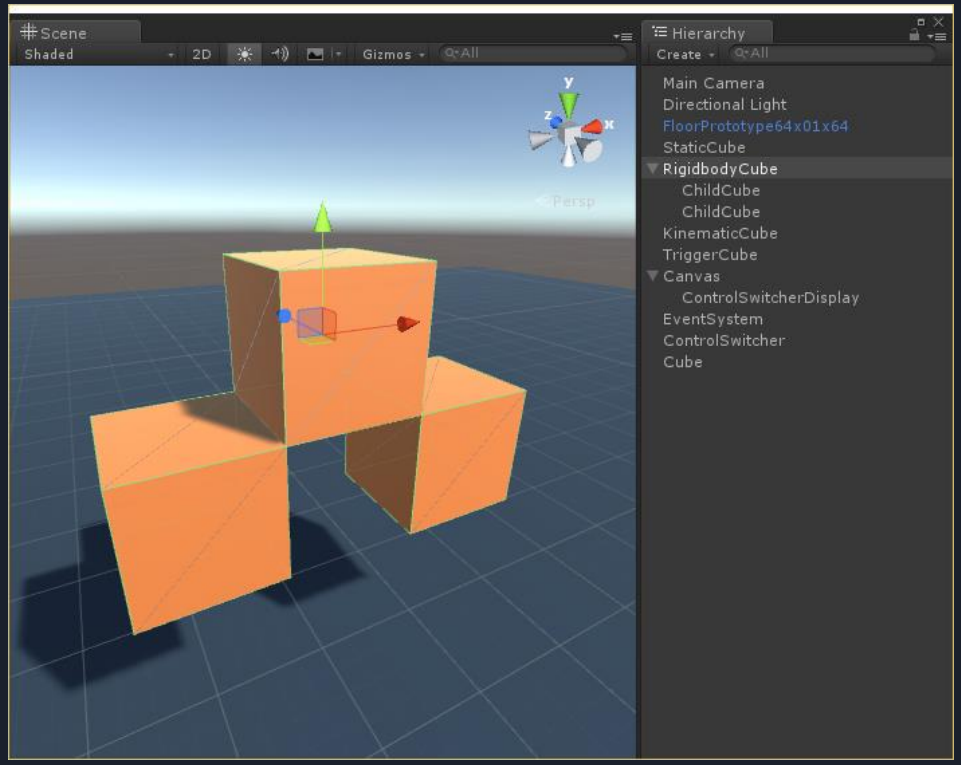
The Rigidbody Component

- By default a GameObject with a Rigidbody will:
  - Be subject to gravity
  - Physically respond to forces and collisions with other objects
- Gravity can be disabled by setting “useGravity” to false.
- Forces and collision response can de disabled by setting “isKinematic” to true.
  - We will come back to this!

The Rigidbody Component

- The shape of the Rigidbody is determined by its attached Colliders.
- A Rigidbody can have multiple Colliders.
  - Add them to child GameObjects.
  - Any collision messages will go to the GameObject with the Rigidbody.
- A Rigidbody should **not** be a child of another Rigidbody.

The Rigidbody Component



The Rigidbody Component

- Forces can be applied to a Rigidbody by colliding with it or through scripts.

- The result of a force applied to a Rigidbody is effected by:
  - The shape of the body (its colliders)
  - The physical properties of the Rigidbody (mass, drag)

The Rigidbody Component

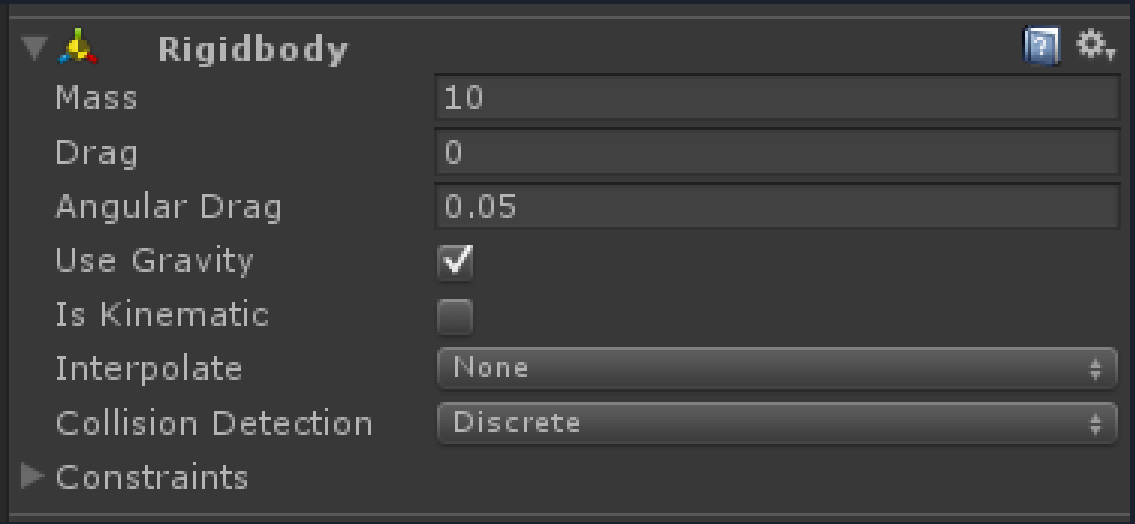
- The “mass” property defines how heavy the Rigidbody is.
  - Influences the effects of forces and collisions
  - Higher mass = larger forces required
- The “drag” and “angular drag” properties influence momentum over time.
  - Higher drag = bodies decelerate more quickly

Kinematic Rigidbodies

- If you set “isKinematic” to true on a Rididbody the physics system stops controlling that body.
  - It will **not** respond to forces or collisions from other bodies.
  - It **will** still push other non-kinematic bodies.
  - It **will** still interact with triggers.

- What is this used for?
  - Moving triggers
  - Movable solid objects which should ignore forces, such as doors

The Rigidbody Component



Static vs. Dynamic Colliders

- A Collider without a Rigidbody is considered to be a “static collider”.

- A Collider with a Rigidbody is considered to be a “dynamic collider”.
  - The Rigidbody may be attached to a parent GameObject.

Static vs. Dynamic Colliders

- Static colliders should not be moved.
  - This causes additional work inside the physics system.
- As long as they don’t move, static colliders are cheaper than dynamic colliders.
- Static colliders are generally used for the unchanging parts of a level
  - Ground, walls, trees

Static vs. Dynamic Colliders

- Dynamic colliders with “isKinematic” set to false:
  - Move via Rigidbody forces
  - Move via Rigidbody MovePosition(...) and MoveRotation(...)
  - Do not move via Transform - can result in instability

- Dynamic colliders with “isKinematic” set to true:
  - Can move via their Transform
  - Or move via Rigidbody MovePosition(...) and MoveRotation(...)
  - Do not move via forces on the Rigidbody – they don’t apply

Collision Rules

- There are a number of rules about how the physics system handles a particular collision.

- Thankfully, the Unity Manual has us covered.
  - Includes matrices of how various configurations interact with one another.

- http://docs.unity3d.com/Manual/CollidersOverview.html

Collision Rules

Collision detection occurs and messages are sent upon collision

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		Y				
Rigidbody Collider	Y	Y	Y			
Kinematic Rigidbody Collider		Y				
Static Trigger Collider						
Rigidbody Trigger Collider						
Kinematic Rigidbody Trigger Collider						

Collision Rules

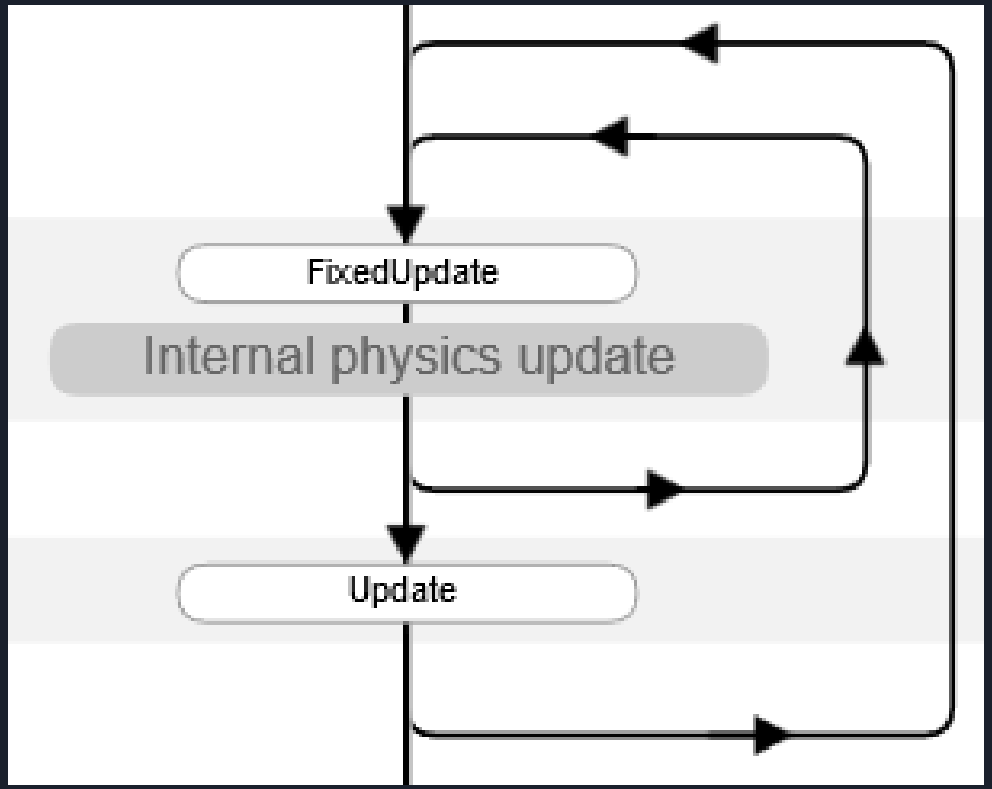
Trigger messages are sent upon collision

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider					Y	Y
Rigidbody Collider				Y	Y	Y
Kinematic Rigidbody Collider				Y	Y	Y
Static Trigger Collider		Y	Y		Y	Y
Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y
Kinematic Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y

Scripting for Rigidbody Components

- Unity's physics system runs separately from its scene update and rendering.
- The scene update (and rendering) runs at a variable rate.
  - Update() is called once per scene update and rendered frame
- The physics system runs at a fixed update rate
  - FixedUpdate() may be called multiple times for some frames, and not at all for others.
  - This is to increase physics stability.

Scripting for Rigidbody Components



Unity Manual, Execution Order of Event Functions (modified)

Scripting for Rigidbody Components

- Use FixedUpdate() for:
  - Applying “continuous” forces
  - Moving Rigidbody objects (non-kinematic)

- Unfortunately, FixedUpdate() isn’t 100% reliable for detecting button presses
  - It may not get called on the frame where the button is pressed
  - Use Update() to set a variable, then perform the action in FixedUpdate()

Scripting for Rigidbody Components

```
bool jumpInput = false;

void Update() {
    // If the button is pressed, set jumpInput to true.
    if (Input.GetKeyDown(KeyCode.J)) {
        jumpInput = true;
    }
}

void FixedUpdate() {
    // If jumpInput is true, apply the jump force and then set it back to
    // false.
    if (jumpInput) {
        Debug.Log("Applying force to " + gameObject.name);
        rigidbody.AddForce(Vector3.up * rigidbody.mass * 300, ForceMode.Force);
        jumpInput = false;
    }

}
```

Scripting for Rigidbody Components

- Rigidbody.AddForce(Vector3 force, ForceMode mode)
  - “force” is a Vector3 representing the direction and strength of the force to be applied, in world space
  - “mode” is one of the available force modes:
    - Force, Acceleration, Impulse, VelocityChange

Scripting for Rigidbody Components

- ForceMode:
  - Force – Add a continuous force to the rigidbody, using its mass.
  - Acceleration – Add a continuous acceleration to the rigidbody, ignoring its mass.
  - Impulse – Add an instant force impulse to the rigidbody, using its mass.
  - VelocityChange – Add an instant velocity change to the rigidbody, ignoring its mass.

- http://docs.unity3d.com/ScriptReference/ForceMode.html

Scripting for Rigidbody Components

```
using UnityEngine;
using System.Collections;

public class RigidbodyControls : MonoBehaviour {

    Rigidbody rigidbody;

    public float power = 100.0f;

    // Use this for initialization
    void Start() {
        rigidbody = GetComponent<Rigidbody>();
        if (rigidbody == null) {
            Debug.LogError("A Rigidbody component is required.", this);
        }
    }

    void FixedUpdate() {
         // Gather movement input
        Vector3 moveInput = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));

        // Calculate the force to apply
        Vector3 appliedForce = moveInput * power;

        // Apply the force to the Rigidbody
        rigidbody.AddForce(appliedForce, ForceMode.Force);
    }

}
```

Scripting for Rigidbody Components

- Rigidbody.MovePosition(Vector3 position)
  - “position” is a Vector3 representing the desired position in world space the Rigidbody is to move to

Scripting for Rigidbody Components

```
using UnityEngine;
using System.Collections;

public class RigidbodyControls : MonoBehaviour {

    Rigidbody rigidbody;

    public float moveSpeed = 1.0f;

    // Use this for initialization
    void Start() {
        rigidbody = GetComponent<Rigidbody>();
        if (rigidbody == null) {
            Debug.LogError("A Rigidbody component is required.", this);
        }
    }

    void FixedUpdate() {
         // Gather movement input
        Vector3 moveInput = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));

        // Calculate the desired movement
        Vector3 desiredMove = moveInput * moveSpeed * Time.deltaTime;

        // Request a physics move to that position
        // Note that MovePosition takes the desired position, so we add the
        // desired movement to the current position.,
        rigidbody.MovePosition(rigidbody.position + desiredMove);
    }

}
```

Scripting for Rigidbody Components

- Find more at:
  - http://docs.unity3d.com/ScriptReference/Rigidbody.html

Interpolation

- As previously mentioned, physics updates and scene updates can happen at different rates.

- This means that some rendered frames do not have an associated physics update.
  - Physics objects will not be visually updated on those frames.
  - This can cause “choppiness” or “jitter”.

- To control this, Rigidbody gives us the following “Interpolation” settings:
  - None, Interpolate, Extrapolate

Interpolation

- None
  - The most current physics values are used as-is.
  - Does not reduce jitter.
  - Does not introduce lag or errors.
- Interpolate
  - Always animates towards the most recent value.
  - Generally gives the smoothest result.
  - Introduces slight lag as it is always slightly behind.
- Extrapolate
  - Predicts the current position based on most recent values.
  - Can introduce errors with fast-moving objects.

Interpolation

- Recommended starting point:
  - Use “None” for most objects.
    - Only add interpolation where jitter is an issue.
  - Use “Interpolate” for player objects followed by camera.
  - Don’t forget to tune the physics timestep for your game.
    - Edit -> Project Settings -> Time -> Fixed Timestep

References

- *Unity Manual*, Physics chapters, Unity Technologies, accessed 10/12/2015
  - http://docs.unity3d.com/Manual/PhysicsSection.html