# Unity Collisions and Physics Support Materials

## Colliders

* The more complicated the collider the more calculations.
* Sometimes it is better to achieve the shape of the object using few collider components instead of one mesh collider
* Colliders can be manipulated
* Types
* Box
* Sphere
* Capsule
* Mesh
* Wheel
* Terrain

## Main Collider Properties

* Is Trigger - If true collision will be detected but it will be like in ghost mode. Objects will go through one another
* Material - Physics material that dictates the friction and bounceness. Few slides later it will be shown in detail
* Center, Size - the dimensions of the collider shape
* In order for collisions to be detected one of the objects in collision must have RigidBody added!

## Detection Types

* IsTrigger -> False
  + void OnCollisionEnter(Collision collision)
  + void OnCollisionExit(Collision collision)
  + void OnCollisionStay(Collision collision)
* IsTrigger -> True
  + void OnTriggerEnter(Collider collider)
  + void OnTriggerExit(Collider collider)
  + void OnTriggerStay(Collider collider)

## Physics Manager

* The one place where we adjust the physics engine. Except the layer matrix you will rarely need to change anything
* Edit->Project Settings-> Physics Properties
* Gravity Vector
* Default Material - once collider is added which physics material to add as default
* Bounce Threshold - if two objects collides and their relative velocity is below that they won't bounce from each other
* Sleep Threshold - rigidbody kinetic energy is below this threshold it will go into sleeping mode and it will become less CPU intensive
* Default Contact offset - the absolute minimum distance b/n two objects to trigger the collision logic
* Default Solver Iterations - the number of tasks that the physics engine will use each frame to solve everything related to physics, like movement of joints, collision b/n two objects etc.. the bigger the number the more agile the physics are but the more heavy on the CPU
* Default Solver Velocity Iterations - same as above but targets the velocity related stuff. The bigger the number the more accurate velocity calculations.
* Layer Collision Matrix - Use this to define how the layer-based collision detection system behaves.

## Physics Materials

* Used to adjust friction and bouncing effects of colliding objects
* Dynamic Friction - The friction used when already moving. Usually a value from 0 to 1. A value of zero feels like ice, a value of 1 will make it come to rest very quickly unless a lot of force or gravity pushes the object.
* Static Friction - The friction used when an object is laying still on a surface. Usually a value from 0 to 1. A value of zero feels like ice, a value of 1 will make it very hard to get the object moving.
* Bounciness - How bouncy is the surface? A value of 0 will not bounce. A value of 1 will bounce without any loss of energy.
* Friction Combine - How the friction of two colliding objects is combined.
  + Average - The two friction values are averaged.
  + Minimum - The smallest of the two values is used.
  + Maximum - The largest of the two values is used.
  + Multiply - The friction values are multiplied with each other.
* Bounce Combine - same as above, but for bouncing

## Raycasts

* Casts a ray, from point origin, in direction direction, of length maxDistance, against all colliders in the scene. You may optionally provide a LayerMask, to filter out any Colliders you aren't interested in generating collisions with.
* Refrain from using it in Update methods. It is not a good idea to fire a ray each frame...

## Rigidbody

* Rigidbodies enable your GameObjects to act under the control of physics. The Rigidbody can receive forces and torque to make your objects move in a realistic way. Any GameObject must contain a Rigidbody to be influenced by gravity, act under added forces via scripting, or interact with other objects through the NVIDIA PhysX physics engine.
* Properties
  + Mass - The mass of the object (in kilograms by default).
  + Drag - How much air resistance affects the object when moving from forces. 0 means no air resistance, and infinity makes the object stop moving immediately.
  + Angular Drag - How much air resistance affects the object when rotating from torque. 0 means no air resistance. Note that you cannot make the object stop rotating just by setting its Angular Drag to infinity.
  + Use Gravity - If enabled, the object is affected by gravity.
  + Is Kinematic - If enabled, the object will not be driven by the physics engine, and can only be manipulated by its Transform. This is useful for moving platforms or if you want to animate a Rigidbody that has a HingeJoint attached.
  + Interpolate - Try one of the options only if you are seeing jerkiness in your Rigidbody’s movement.
    - None - No Interpolation is applied.
    - Interpolate - Transform is smoothed based on the Transform of the previous frame.
    - Extrapolate - Transform is smoothed based on the estimated Transform of the next frame.
  + Collision Detection - Used to prevent fast moving objects from passing through other objects without detecting collisions.
    - Discrete - Less CPU intensive, default, most cases are ok to use them.
    - Continuous - Use it when you are dealing with fast moving objects. It is CPU intensive, but gives better results.
    - Continuous Dynamic - Use it when you are dealing with other Continuous or Continuous Dynamic fast moving objects. Again used for fast moving objects.
  + Constraints - Restrictions on the Rigidbody’s motion:-
    - Freeze Position Stops the Rigidbody moving in the world X, Y and Z axes selectively.
    - Freeze Rotation Stops the Rigidbody rotating around the local X, Y and Z axes selectively.

## Basic RigidBody scripting

* **AddForce/AddRelativeForce(Vector3, ForceMode)** - remember the force direction can be global and local, force mode dictates the type of the force :
  + 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.
* Velocity - self explanatory. You can change the velocity instead of adding force, but methods like adding force are preferable
* MovePosition/MoveRotation - to move the object while keeping in mind the physical laws. It is always better to use that instead of Transform.Position/Rotation if the GameObject has RigidBody.

## Joints

* Fixed
* Spring
* Hinge

## 