Colliders

Other Versions

Collider components define the shape of an object for the purposes of physical collisions. A collider, which is invisible, need not be the exact same shape as the object's mesh and in fact, a rough approximation is often more efficient and indistinguishable in gameplay.

The simplest (and least processor-intensive) colliders are the so-called *primitive* collider types. In 3D, these are the Box Collider, Sphere Collider and Capsule Collider. In 2D, you can use the Box Collider 2D and Circle Collider 2D. Any number of these can be added to a single object to create *compound colliders*.

With careful positioning and sizing, compound colliders can often approximate the shape of an object quite well while keeping a low processor overhead. Further flexibility can be gained by having additional colliders on child objects (eg, boxes can be rotated relative to the local axes of the parent object). When creating a compound collider like this, there should only be one Rigidbody component, placed on the root object in the hierarchy.

Note, that primitive colliders will not work correctly with shear transforms - that means that if you use a combination of rotations and non-uniform scales in the tranform hierarchy so that the resulting shape would no longer match a primitive shape, the primitive collider will not be able to represent it correctly.

There are some cases, however, where even compound colliders are not accurate enough. In 3D, you can use Mesh Colliders to match the shape of the object's mesh exactly. In 2D, the Polygon Collider 2D will generally not match the shape of the sprite graphic perfectly but you can refine the shape to any level of detail you like. These colliders are much more processor-intensive than primitive types, however, so use them sparingly to maintain good performance. Also, a mesh collider will normally be unable to collide with another mesh collider (ie, nothing will happen when they make contact). You can get around this in some cases by marking the mesh collider as **Convex** in the inspector. This will generate the collider shape as a "convex hull" which is like the original mesh but with any undercuts filled in. The benefit of this is that a convex mesh collider *can* collide with other mesh colliders so you may be able to use this feature when you have a moving character with a suitable shape. However, a good general rule is to use mesh colliders for scene geometry and approximate the shape of moving objects using compound primitive colliders.

Colliders can be added to an object without a Rigidbody component to create floors, walls and other motionless elements of a scene. These are referred to as **static** colliders. In general, you should not reposition static colliders by changing the Transform position since this will impact heavily on the performance of the physics engine. Colliders on an object that *does* have a Rigidbody are known as *dynamic* colliders. Static colliders can interact with dynamic colliders but since they don't have a Rigidbody, they will not move in response to collisions.

The reference pages for the various collider types linked above have further information about their properties and uses.

Physics materials

When colliders interact, their surfaces need to simulate the properties of the material they are supposed to represent. For example, a sheet of ice will be slippery while a rubber ball will offer a lot of friction and be very bouncy.

Physics Material 2D for further details on the available parameters. Note that for historical reasons, the 3D asset is actually called **Physic Material** (*without* the S) but the 2D equivalent is called **Physics Material 2D** (*with* the S).

Triggers

The scripting system can detect when collisions occur and initiate actions using the `OnCollisionEnter` function. However, you can also use the physics engine simply to detect when one collider enters the space of another without creating a collision. A collider configured as a **Trigger** (using the **Is Trigger** property) does not behave as a solid object and will simply allow other colliders to pass through. When a collider enters its space, a trigger will call the `OnTriggerEnter` function on the trigger object's scripts.

Script actions taken on collision

When collisions occur, the physics engine calls functions with specific names on any scripts attached to the objects involved. You can place any code you like in these functions to respond to the collision event. For example, you might play a crash sound effect when a car bumps into an obstacle.

On the first physics update where the collision is detected, the `OnCollisionEnter` function is called. During updates where contact is maintained, `OnCollisionStay` is called and finally, `OnCollisionExit` indicates that contact has been broken. Trigger colliders call the analogous `OnTriggerEnter`, `OnTriggerStay` and `OnTriggerExit` functions. Note that for 2D physics, there are equivalent functions with **2D** appended to the name, eg, `OnCollisionEnter2D`. Full details of these functions and code samples can be found on the Script Reference page for the MonoBehaviour class.

With normal, non-trigger collisions, there is an additional detail that at least one of the objects involved must have a non-kinematic Rigidbody (ie, *Is Kinematic* must be switched off). If both objects are kinematic Rigidbodies then `OnCollisionEnter`, etc, will not be called. With trigger collisions, this restriction doesn't apply and so both kinematic and non-kinematic Rigidbodies will prompt a call to `OnTriggerEnter` when they enter a trigger collider.

Collider interactions

Colliders interact with each other differently depending on how their Rigidbody components are configured. The three important configurations are the *Static Collider* (ie, no Rigidbody is attached at all), the *Rigidbody Collider* and the *Kinematic Rigidbody Collider*.

Static Collider

This is a GameObject that has a Collider but no Rigidbody. Static colliders are used for level geometry which always stays at the same place and never moves around. Incoming rigidbody objects will collide with the static collider but will not move it.

The physics engine assumes that static colliders never move or change and can make useful optimizations based on this assumption. Consequently, static colliders should not be disabled/enabled, moved or scaled during gameplay. If you do change a static collider then this will result in extra internal recomputation by the physics engine which causes a major drop in performance. Worse still, the changes can sometimes leave the collider in an undefined state that produces erroneous physics calculations. For example a raycast against an altered Static Collider could fail to detect it, or detect it at a random position in space. Furthermore, Rigidbodies that are hit by a moving static collider will not necessarily be "awoken" and the static collider will not apply any friction. For these reasons, only colliders that are Rigidbodies should be altered. If you want a collider object that is not affected by incoming

This is a GameObject with a Collider and a normal, non-kinematic Rigidbody attached. Rigidbody colliders are fully simulated by the physics engine and can react to collisions and forces applied from a script. They can collide with other objects (including static colliders) and are the most commonly used Collider configuration in games that use physics.

Kinematic Rigidbody Collider

This is a GameObject with a Collider and a *kinematic* Rigidbody attached (ie, the *IsKinematic* property of the Rigidbody is enabled). You can move a kinematic rigidbody object from a script by modifying its Transform Component but it will not respond to collisions and forces like a non-kinematic rigidbody. Kinematic rigidbodies should be used for colliders that can be moved or disabled/enabled occasionally but that should otherwise behave like static colliders. An example of this is a sliding door that should normally act as an immovable physical obstacle but can be opened when necessary. Unlike a static collider, a moving kinematic rigidbody will apply friction to other objects and will “wake up” other rigidbodies when they make contact.

Even when immobile, kinematic rigidbody colliders have different behavior to static colliders. For example, if the collider is set to as a trigger then you also need to add a rigidbody to it in order to receive trigger events in your script. If you don’t want the trigger to fall under gravity or otherwise be affected by physics then you can set the *IsKinematic* property on its rigidbody.

A Rigidbody component can be switched between normal and kinematic behavior at any time using the *IsKinematic* property.

A common example of this is the “ragdoll” effect where a character normally moves under animation but is thrown physically by an explosion or a heavy collision. The character’s limbs can each be given their own Rigidbody component with *IsKinematic* enabled by default. The limbs will move normallly by animation until *IsKinematic* is switched off for all of them and they immediately behave as physics objects. At this point, a collision or explosion force will send the character flying with its limbs thrown in a convincing way.

Collision action matrix

When two objects collide, a number of different script events can occur depending on the configurations of the colliding objects’ rigidbodies. The charts below give details of which event functions are called based on the components that are attached to the objects. Some of the combinations only cause one of the two objects to be affected by the collision, but the general rule is that physics will not be applied to an object that doesn’t have a Rigidbody component attached.

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						

Kinematic Rigidbody Trigger Collider						

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

Leave feedback

Is something described here not working as you expect it to? It might be a **Known Issue**. Please check with the Issue Tracker at **issuetracker.unity3d.com**.

Tutorials Community Answers Knowledge Base Forums Asset Store