



Collision Rules

A way to manage to interaction between entities and their environment.

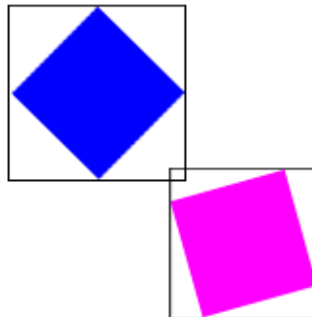
1 | Introduction

The collision rules system allows precise control over how entities and groups of entities interact with each other. Every possible collision between two collidable objects is controlled by one of these collision rules.

To understand exactly how this system works, it helps to first understand how the collision pair system works.

1.A | BroadPhase: Creating a Collision Pair

The first stage in the pipeline, known as “broad phase” collision detection, determines which collidable objects are in danger of colliding. Each collidable object has its own axis-aligned bounding box which fully contains the collision shape of the collidable object. A bounding box overlap signifies a potential collision pair, such as in the image below.



The BroadPhases of Bepu physics identify these pairs using various acceleration structures. The current recommended BroadPhase, DynamicBinaryHierarchy, uses incrementally updated hierarchical bounding volumes that can be traversed for quick pair finding.

Once a bounding box overlap is found, the pair is tested for validity. To be valid, a pair must pass a few internal tests as well as the user-defined CollisionRule, described later. A validated pair is added to the list of CollisionPairs in the space. Note that a valid collision pair does not necessarily mean that the geometry of the collidable objects is intersecting.

1.B | Contact Generation

The second stage, termed "narrow phase" collision detection, is responsible for collecting detailed information about a collision. If any geometry intersection is found, data such as collision location, surface normal, and penetration depth are stored inside Contact objects, which represent individual contact points.

Every contact generation algorithm works by creating a set of these Contacts that approximate the touching area between two shapes. A single corner touching another shape could be represented by a single contact point, an edge-surface collision by two contact points, and a surface-surface by three or four contact points.

1.C | Collision Response

The last stage calculates how to make each shape react to the collisions it is a part of. Each individual Contact can be thought of as a constraint that disallows penetration between two objects.

BEPUpysics iterates over all of these Contact constraints repeatedly to converge towards a solution that satisfies every constraint.

2 | User-Defined Collision Rules

Through specifying collision rules, it is possible to selectively stop the collision pair system at an arbitrary point in the pipeline. There are multiple entries in the CollisionRule enumeration that represent these different stages:

- **CollisionRule.Defer:** Defers the decision over a collision rule to the next lower priority stage in the CollisionRule system, explained later.
- **CollisionRule.Normal:** Allows a collision pair to undergo every aspect (broad phase, narrow phase, collision response) in the collision pair pipeline.
- **CollisionRule.NoResponse:** Allows a collision pair to undergo broad phase and narrow phase, but skips collision response.
- **CollisionRule.NoContacts:** Allows a collision pair to undergo broad phase only, resulting in a CollisionPair instance but no further computations.
- **CollisionRule.NoPair:** Prevents a CollisionPair instance from being created between the two objects. No collision can occur.

2.A | Collision Rule Options

Every collidable object (including `StaticTriangleGroups`, `Terrains`, and `StaticGroups`) has a `CollisionRules` property containing three different settings which are combined and tested against other entities to determine a collision pair's ultimate `CollisionRule`. As a rule of thumb, the more specific rules take precedence over the more general rules.

- **SpecificEntities:** Dictionary containing a listing of other entities and what `CollisionRule` the collidable object associates with them. Entities not contained within the list are considered to have a relationship of `CollisionRule.Defer`.
- **Personal:** `CollisionRule` of the collidable object. Defaults to `CollisionRule.Defer`.
- **Group:** `CollisionGroup` of the collidable object. When added to a space, an object will receive a default collision group if a different one has not already been set.

2.B | Collision Groups

The `CollisionGroup` setting offers a significant amount of control. In a collision pair, unless the `CollisionRule` is determined before the collision groups are considered (as explained later), the two colliding objects' `CollisionGroups` are tested against each other.

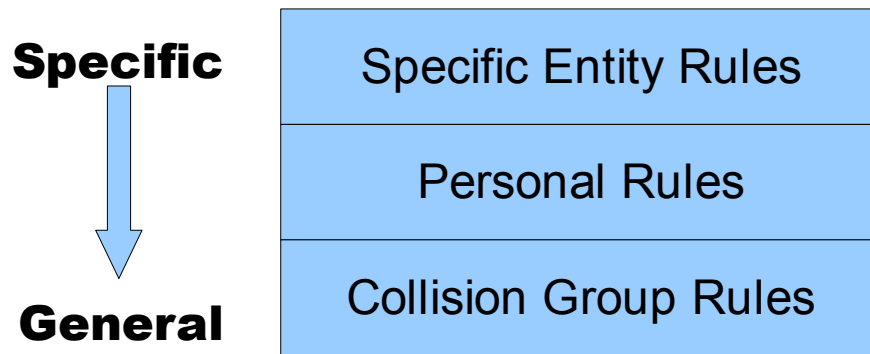
To perform this test, the `Space.SimulationSettings.CollisionDetection.CollisionGroupRules` dictionary is accessed. This dictionary contains entries with keys of `CollisionGroup` pairs and values of `CollisionRules`. New relationships can be added to this dictionary, though it by default includes a relationship only between the `DefaultKinematicCollisionGroup` and itself (`CollisionRule.NoPair`, since kinematic objects shouldn't collide with other kinematic objects by default).

To use custom collision groups, simply create a new `CollisionGroup` instance. You can reuse this instance across all the collidable objects that you want by setting the `CollisionRules.Group` field of each one.

Then, create a `CollisionGroupPair` using that new group and any other group and add it to the `Space.SimulationSettings.CollisionDetection.CollisionGroupRules` dictionary with whatever `CollisionRule` type you'd like. You can also add and modify the `CollisionRules` of the default `CollisionGroups`. The `CollisionGroup` class contains a variety of static helper functions to make rule assignment between `CollisionGroups` and sets of `CollisionGroups` easier.

2.C | Combining Collision Rules

Since there are three different settings for each collidable object in a pair, some computation must be done to determine what CollisionRule to use for the collision pair. This is done through prioritizing more specific rules over more general relationships. The SpecificEntities rules are considered the most specific, the personal rules are in the middle, and the collision groups are the most general.



The SpecificEntities rule, if defined, overrides the personal rule which, if defined, overrides the group rule. If after going through all stages no CollisionRule has been defined, the simulation settings' DefaultCollisionRule is returned.

In the SpecificEntities rule, each collidable object can consider the other with a different CollisionRule since each object has its own SpecificEntities dictionary. Similarly, each collidable object can have its own personal rule. Since it is possible for rules of equal priority to contradict each other, the most severe 'restriction' is used at a given priority level. From most severe restriction to least severe, the CollisionRules are ordered as follows: NoPair, NoContacts, NoResponse, Normal.

3 | Advanced

There are a few extra details which can be leveraged for more control and are necessary for a complete understanding of how the collision rule system works.

3.A | Changing Collision Rules

Internally, the CollisionRule for a collision pair is computed before the pair can be validated by the BroadPhase. This allows the validation method to test for CollisionRule.NoPair rule, which prevents a CollisionPair instance from being created.

The CollisionRule computed by the BroadPhase is passed to the CollisionPair instance after it is created. It will not be recomputed for the lifespan of the CollisionPair. However, since the CollisionPair's CollisionRule field is public, it can be modified later.

One useful application of this is using immediate entity events (those with present tense naming) to allow arbitrary code to be executed right in the middle of the engine's execution. It is possible to intercept a `CollisionPair` once it is created (or after a variety of other events) and, based on whatever conditions desired, modify the pair's `CollisionRule`.

`CollisionPairs` also have a separate but similar field, named `ManifoldUpdateSetting`, that can be changed to override all of the other conditions for updating the contact list. If set to `AlwaysUpdate`, contacts will be maintained every frame regardless of `CollisionRule`. If set to `NeverUpdate`, contacts will never change.

Changing a collidable object's `CollisionRules` settings will not automatically update the `CollisionRules` of associated `CollisionPairs`. To update them, the Entity's `ForceCollisionRuleRecalculation()` method must be called.

It is also possible to use the same instance of `CollisionRules` for multiple collidable objects if you wish to re-use the same settings. To accomplish this, simply set the `CollisionRules` property to the other instance. If there are any `CollisionPairs` involved with the entity, you will still need to call the entity's `ForceCollisionRuleRecalculation()` method to update their `CollisionRules`.

3.B | Special Cases

At some points in the engine, certain combinations of settings are ignored in favor of some internal test. One example of this is the `BroadPhase`'s requirement that at least one of the entities in a collision pair must be active in order for the collision pair instance to be created. This overrides the bounding box overlap test and the collision rule test.

Additionally, for two kinematic entities in a `CollisionPair` with a `CollisionRule` of `CollisionRule.Normal`, no collision response will take place. In effect, the engine considers the `CollisionRule` to be `CollisionRule.NoResponse` since kinematic entities cannot undergo collision response.

It is also possible for the narrow phase to be short circuited if the engine finds that there would be no change in the contact list due to stationary colliders. This can occur only when the `CollisionPair` has a `CollisionRule` of `CollisionRule.Normal`, so if `CollisionRule.NoResponse` is used instead, the narrow phase cannot be short circuited.

3.C | Compound Bodies

The hierarchical nature of CompoundBody entities opens up a couple of complexities when it comes to CollisionRules.

3.C.a | Setting Collision Rules on a Compound Body

A CompoundBody's CollisionRules property can be set directly, like a normal Entity's. However, in the CompoundBody's case, the set value will propagate to the CompoundBody's children recursively. This can be used to easily overwrite any other settings with a single unified set of rules.

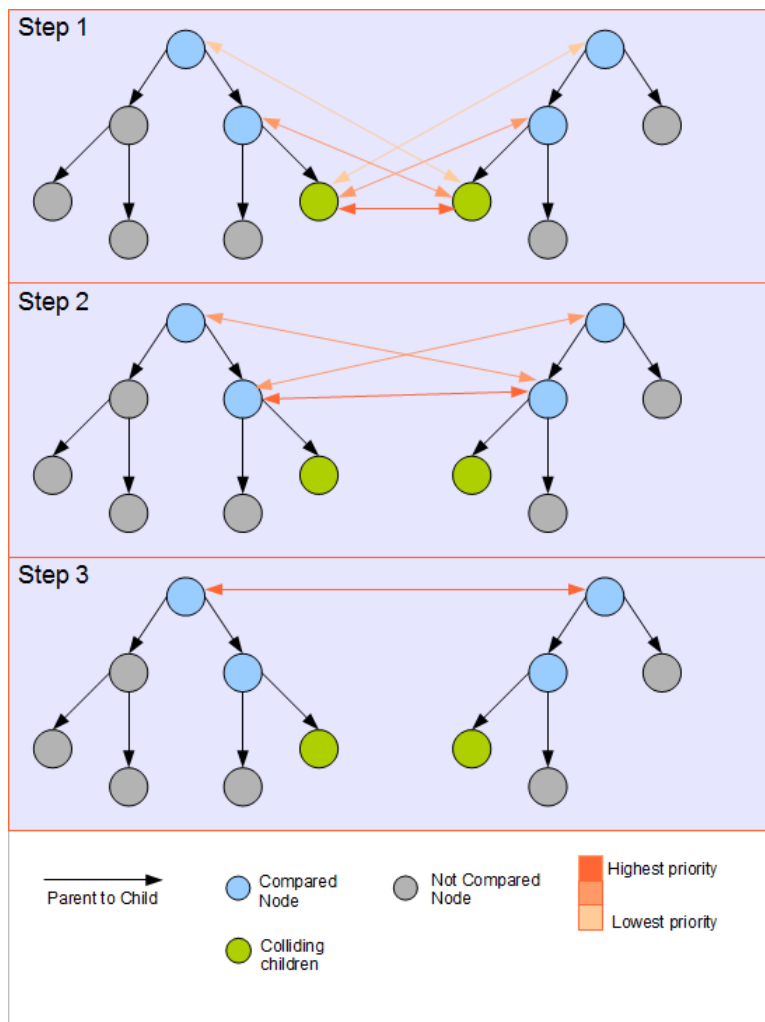
The CompoundBody.AddBody method gives an option through a second parameter to set the CollisionRule property of the added entity to that of the parent. If this is false, the child will only inherit the parent's collision group (if one exists). When a CompoundBody is added to a space, the default CollisionGroup will be given to the CompoundBody and its children if they did not previously have a CollisionGroup set.

3.C.b | Combining Collision Rules with Compound Bodies

When considering a potential collision pair between either a CompoundBody and a normal entity or a CompoundBody and another CompoundBody, the engine adds another dimension to the combination process. The “specific to general” guideline of 2.C still applies, but now each stage takes into account collision rules starting at the children and working its way up through parent compound bodies until it reaches the root of the hierarchy. This secondary prioritization can be thought of as a continuation of the “specific to general” guideline, since the entities in the hierarchy that are closest (hierarchy-wise) to the collision are the first ones tested.

This system can be used to specify CompoundBody-wide rules easily while allowing for the addition of specific branches or entities inside the CompoundBody that have their own rules.

The following diagram shows an example of how the system would work between two multi-level compound bodies where two children (green leaf nodes) are colliding. Note that only leaf nodes have collidable geometry; each internal node is a CompoundBody. Each step represents a direct comparison followed by a series of “criss-cross” tests up to the root. The tests closer to the leaf node are higher priority, and the earlier steps are higher priority than later steps.



The implementation of this process starts by looking at the colliding children and their SpecificEntities relationship to each other. If no specific rule is defined between them, the comparison continues to each child being tested against the opposing child's parent in a criss-cross. The criss-cross testing continues from the child to the opposing child's next parent until there are no more parents. At each comparison, the 'most restrictive' rule is kept of the pair.

If the CollisionRule is still CollisionRule.Defer, the comparison continues to testing the parents of the original children against each other. They perform the same direct comparison followed by criss-crossing tests to the root. This repeats until both entities have reached their root or a CollisionRule other than CollisionRule.Defer is found.

If after reaching the root there is still no specific rule defined, the process restarts at the children and begins testing the entities' personal CollisionRules. The 'most restrictive' personal rule in a pair of entities is kept. This stage's process works slightly differently than the other two stages in that no 'criss-cross' tests are necessary. The testing continues to the root once again, or until a CollisionRule other than CollisionRule.Defer is found.

Finally, if no collision rule is found in the personal rule comparison, the process restarts at the children and begins testing the entities' collision group rules. The testing continues to the root once again, or until a

CollisionRule other than CollisionRule.Defer is found.

For the purposes of determining a CollisionRule in a CollisionPair involving a normal entity and a CompoundBody, the normal entity can be thought of as a single-node CompoundBody (i.e., the root is the leaf node).

3.D | Custom Rule Calculation

Every BroadPhase object has a CollisionRuleCalculator delegate which is called to determine the collision rule of a pair of collidable objects. This is stored in the BroadPhase's CalculateCollisionRuleCallback field.

By default, CalculateCollisionRuleCallback is set to the EntityCollisionRules.GetCollisionRule static method. A custom method can be used as the callback to perform alternate logic.