1.12 Artist Best Practices

1.12.1 Introduction

This chapter contains advice and information on best practices for artists when setting up Havok collision objects and scenes.

1.12.2 Collision Detection Optimizations

We will explore how to make your levels more efficient for use with Havok. The first task is to understand the general operations of the Havok collision detection system, and what issues in a scene can lead to poor performance.

First, let's take a quick look at the different levels of collision detection that happen in the simulation world. Havok breaks collision detection up into three phases: broadphase, midphase and narrowphase.

1.12.2.1 Collision Level: Broadphase

Key Points :

- Merge all fixed rigid bodies into a single rigid body wherever possible.
- Keep active dynamic rigid bodies to a minimum.

The broadphase is the highest level collision detection system, and it is used to quickly identify pairs of bodies that could potentially be colliding. Each rigid body is represented by a box (an AABB to be more precise, which is a box aligned to the world axes, i.e. never allowed to rotate).

For the most part, you do not need to understand the inner workings of the broadphase. Simply remember that having fewer rigid bodies in the world is better because the broadphase is less cluttered. While this is true for both fixed and moving rigid bodies, when designing the level you should focus more on limiting the number of **active** moving rigid bodies in the scene.

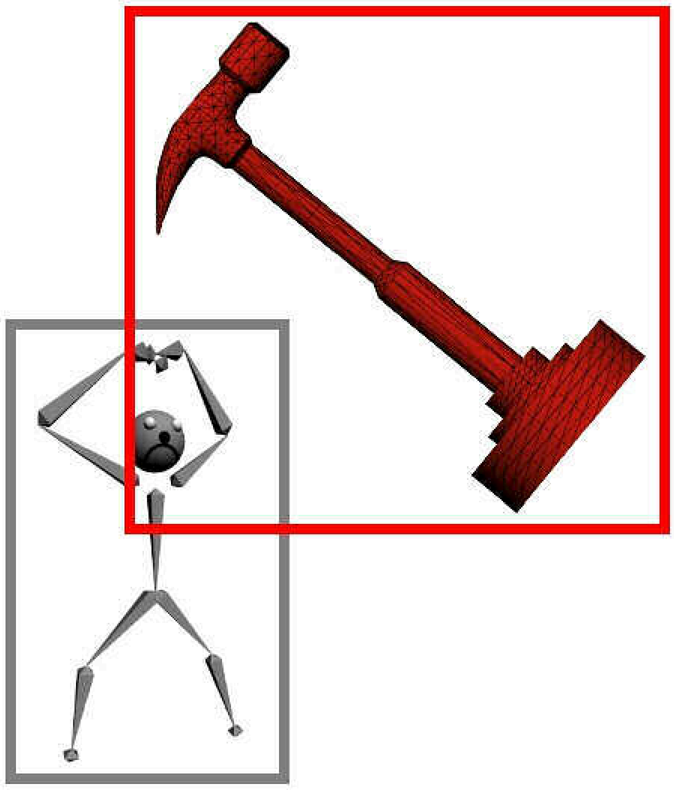

Figure 1.380: Broad-phase collisions (using AABBs)

For each pair of rigid bodies identified in the broadphase, we take a closer look to see if they are actually colliding. This is called the *narrowphase* collision detection: objects are compared on a shape by shape basis (a rigid body being formed of one or more primitive shapes). For interactions of simple rigid bodies (formed of a single Primitive Shape: Sphere, Capsule, Box, Convex Hull, Cylinder or Triangle) only one comparison is carried out.

For rigid bodies formed of a **compound shape** (i.e. a collection of shapes), each individual shape is equivalent collision-wise to a separate simple rigid body. Similarly for a mesh shape (a shape formed of a triangle list), each triangle involved in a *narrowphase* collision is equivalent to (if not more expensive than) a whole separate simple rigid body.

Before discussing the *narrowphase*, we'll explore the *midphase*, which aims to reduce the number of individual shape collision comparisons that must be performed when colliding compound shapes.

1.12.2.2 Collision Level: Midphase

Key Point :

- Wrap all Compound Shapes (i.e. shape collections) in a MOPP structure.

Havok introduces a separate collision detection step (the midphase) to minimize the cost associated with complex shapes collisions (such as triangle landscapes). In general, any shape collection of more than a few objects should be wrapped up in a MOPP. This is especially applicable for triangle landscapes.

When wrapped in a MOPP, a rigid body can be quickly tested for potential collisions against other rigid bodies.

If potential collisions are identified, then only the relevant shapes in this body will be further tested. Consider the example above with the character and the hammer. Without a MOPP around the hammer, every individual triangle forming the hammer would be tested against each shape of the character. With a MOPP, none of the triangles would be tested.

If a portion of the hammer was actually colliding with the character's arm, only the triangles "near" the arm would be tested further for collisions.

MOPPs are primarily intended for large fixed landscapes, but it is applicable to dynamic shape collections as well. However, we also advise against the use of complex moving shape collections, and would recommend investigating if a convex hull would better suit the collision quality needs for such an object.

Please refer to the documentation on the Create Rigid Bodies filter for information on how to automatically create MOPPs for shape collections (also known as compound shapes).

1.12.2.3 Collision Level: Narrowphase

Key Points :

- Avoid unnecessary details in your collision shapes whenever possible.
- Keep the potential number of collision calculations low by using large triangles in the collision landscape and by maintaining shape collections that are as simple as possible.
- Remove all physical detail in regions where physical objects and characters can't enter.

The final collision detection step (narrowphase) is carried out between the colliding objects' basic parts: capsule vs. triangle, capsule vs. capsule, box vs. box, box vs. triangle, etc. Take the example of a table made of a collision list of 3 boxes and let's consider the situation where the broadphase box (AABB) for that table overlaps with every rigid body of a 12 body ragdoll. We'll then obtain 36 collision calculations taking place every frame the bodies are moving, which is not insignificant.

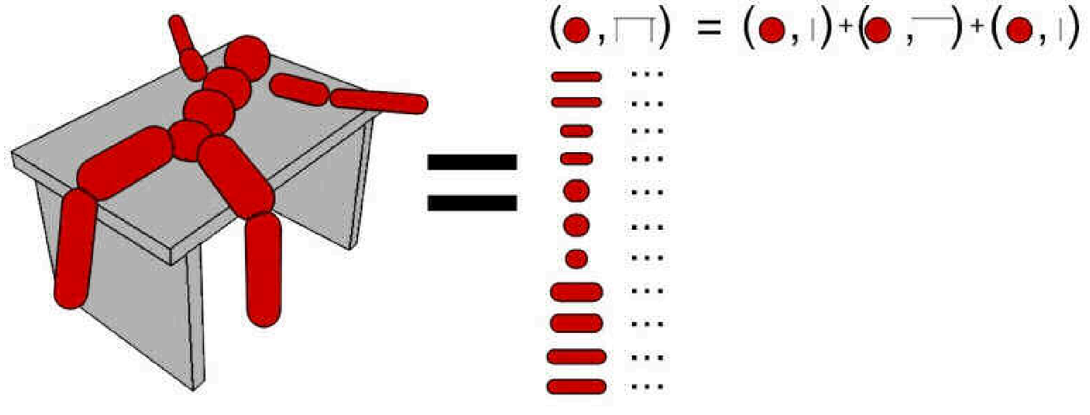

Figure 1.381: Ragdoll colliding with a table.

Now consider the case where the same ragdoll has been draped across a highly triangulated statue in a level. The bounding box for each shape in the ragdoll might overlap 50 triangles (and possibly even more). This works out to 50 * 12 collisions = 600 collision calculations for the ragdoll colliding with the statue.

The ragdoll might seem like an example that people would expect to be costly but imagine the statue has 400 triangles in total and a player walks up against the statue so that the capsule's bounding box (A.I.s and Player Character are usually simulated using capsules) overlaps most of the triangles in the statue. That's 400 collision calculations (or more) per frame for that character just because the character walked close to the statue – the same cost as calculating collisions for 400 capsules falling on one large triangle!

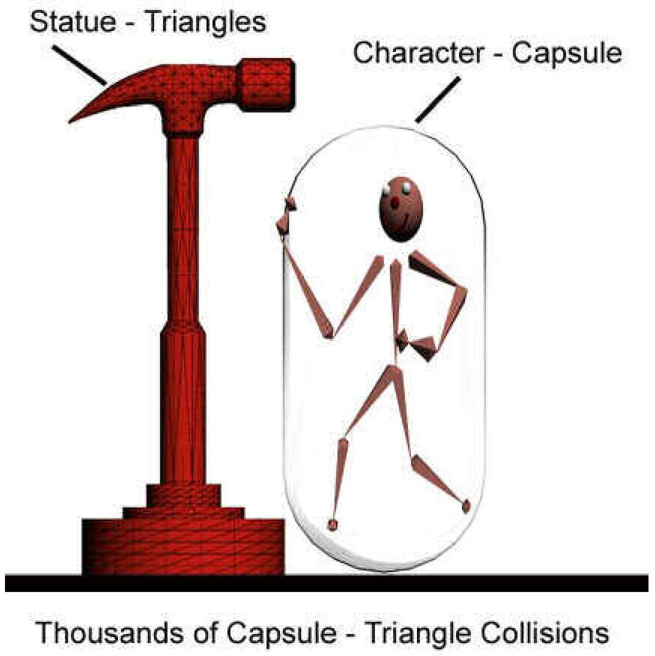

Figure 1.382: Collisions between a capsule and a large triangle list.

During the narrowphase, final collision calculations are carried out at the most basic shape (e.g. triangle) level. This means that when you're calculating potentially expensive areas of narrowphase collision detection, consider each fixed triangle, box, capsule, etc. in the scene as a potential collision object.

You want to avoid scenarios where a moving object or the character could collide against multiple fixed rigid bodies in the scene.

It's hard to let go of detail but in most cases it really won't affect the physical feel of the game. If you can make a chair with two simple boxes, then do so. How many objects in the scene are small enough or would last long enough to actually get in between the legs of the chair? Always ask yourself: is this amount of detail necessary to maintain realistic behavior. Always convert detailed triangle areas to the coarsest possible representation that would still perform the basic requirements of collision detection.

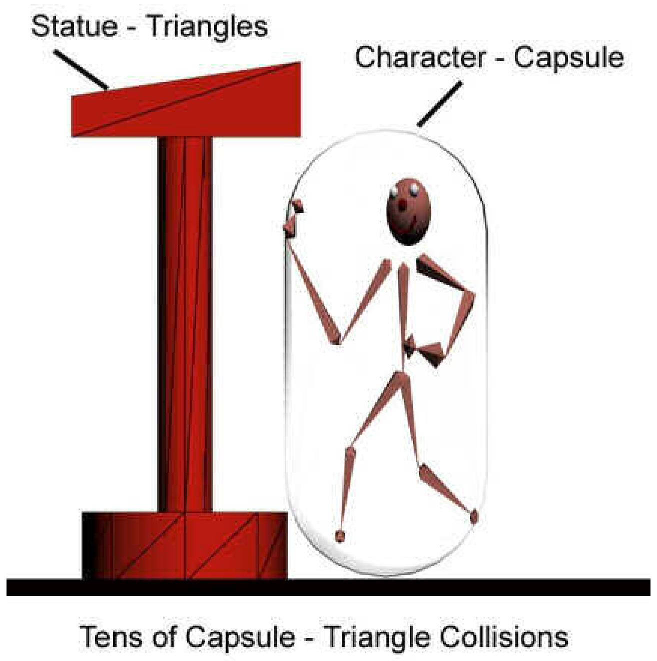

Figure 1.383: Collisions between a capsule and a simpler triangle list.

1.12.2.4 The Big Collision Win

No More Triangles!

We've already discussed how each triangle is equivalent to one collision shape. Havok supports many other types of collision primitives. If these could be used to represent regions of your scene there is an opportunity for large CPU cost savings.

The main advantage to using these alternative objects is that certain objects can be represented using far fewer pieces than if they were constructed using triangles. Consider the previous example of the statue. Even with the representation optimized, the statue will contain nearly 100 triangles; at best a character might overlap only twenty of these but imagine instead if the statue was made up of a cylinder, a capsule and a box, the same collisions can be evaluated using only three collision calculations.

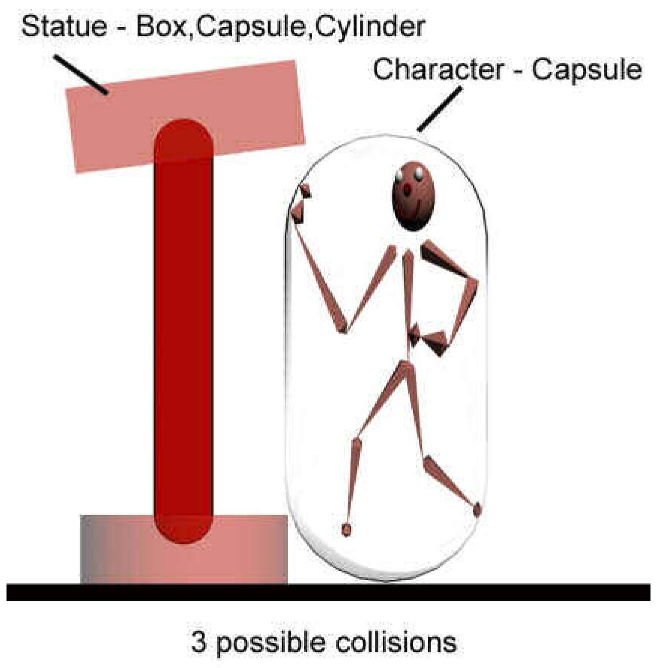

Figure 1.384: Collisions between a capsule and a list of simple shapes.

If you have a rectangular floor formed of triangles, we highly recommend using a single large box for its collision representation instead.

There are other benefits to replacing triangles with simple collision shapes. Triangles are thin and have no concept of "inside" or "outside" in terms of collision. When they are used to enclose volumes they can trap and hold moving bodies. Additionally, if an object sufficiently penetrates a triangle the proper collision response can be to push the object through to the other side. There are various ways this can happen and various methods to prevent it but it is best to simply avoid these situations all together.

Boxes, spheres, capsules, cylinders and convex hulls define collision volumes. The thicker the shape the less likely it is that moving rigid bodies will penetrate it sufficiently for them to pass through.

Consider the following floor made of triangles with a small box falling towards it. When the floor is made of triangles and the box is falling downwards, even though the triangles have some thickness the box only has to pass half way through the triangles in one step before it will be pushed out the bottom of the triangles and probably out of the world.

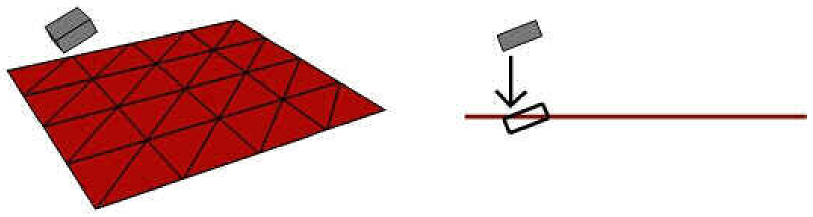

Figure 1.385: A box colliding against a triangle floor.

Now if we replace the floor triangles with a thick box:

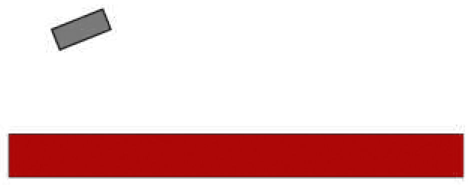

Figure 1.386: A box colliding against a box floor.

The moving box is less likely to move through the floor box in one step before the collision detection catches it and the system solves the collision and pushes the box out correctly.

Consider the scenario where a rigid body is activated at runtime and added to the world. What happens if this new rigid body is penetrating the landscape triangles (when it is added)? The object gets pushed in between the floor triangles which can cause jitter, or even worse: an object falling through the landscape.

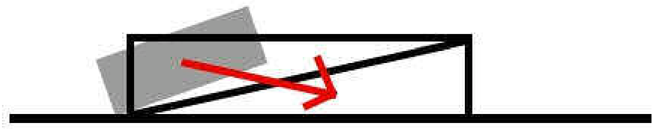

Figure 1.387: A box being pushed through a triangular floor.

If the floor is instead simulated using a fixed box then, on addition, the collision will be handled correctly and the newly added rigid body will quickly be pushed out correctly into the world.

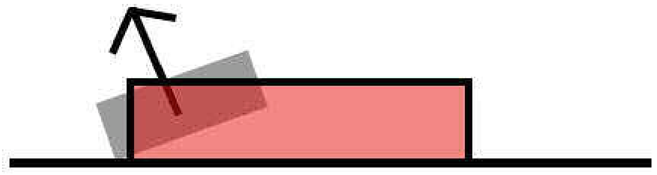

Figure 1.388: A box being pushed back out of a box floor.

These volumes can be also be used to prevent constrained systems of rigid bodies (e.g. ragdolls) achieving undesirable poses. For example a seat modeled as part of the triangulated landscape could hold constrained bodies (upper arms / forearms) apart. If the seat were modeled as a box then nothing could pass through into the space under the seat.

That being said, it's not always possible to replace triangles with collision volume shapes. The need to model accurate holes in floors, to model curved surfaces and to store per-triangle collision properties that can be changed on the fly means that triangles are at times the simplest option.

1.12.2.5 Summary and General Tips

To recap, here are the best ways to optimize your scenes and rigid bodies for performance:

- **Minimize the number of rigid bodies** in the scene (to reduce broadphase cost).
- **Minimize the complexity of rigid bodies** (to reduce narrowphase cost).
- Do not use visual geometry for the collision representation. In almost all cases, it is simply overkill. Use simple collision shapes to broadly define the collision representation.
- Use simpler geometry representations wherever possible. Minimize the number of triangles whenever you can.
- Favor simple shapes over convex hulls and triangles/meshes (see the collision cost table below).
- Avoid overlapping or redundant shapes. In particular, shapes fully contained within another are simply wasted collision processing time.
- Avoid using shapes that are too small, especially in collision list shapes. Small shapes have an increased risk of passing through triangle shapes, and getting ”stuck” in a landscape.
- Keep the numbers of objects in constrained systems to a minimum. All shapes within a constrained system are simulated together.

Performance Cost	Shape Type
Cheapest	Sphere
	Capsule
	Box
	Triangle
	Cylinder
	Convex Hull
Most Expensive	Meshes/Triangle List

Table 1.85: Object Collision Cost - Rule of Thumb

Avoid Using Concave Shapes

Use convex shapes whenever possible. It is usually better to represent a concave object as either a list of convex shapes or a convex hull. Havok provides two very useful tools for this purpose:

- **Convex Decomposition Tool** : Concave meshes should always be simplified for use in Havok. In many cases (such as the Hammer object above), it can easily be approximated with a set of simple shapes. This is the recommended approach. In situations where greater detail is required, Havok has provided the convex decomposition tool. This tool will take in any concave object and produce a set of convex hulls which approximate the shape within some tolerance. For more information please refer to the documentation for the convex decomposition tool.

  Please note that even after decomposing objects, we highly recommend combining these individual shapes into a single rigid body (using a Compound Shape) whenever possible.
- **Convex Hull Tool** : Great speed improvements can be gained by avoiding the use of Meshes whenever possible. For convex meshes, this can easily be obtained by changing the type of the shape to **convex hull** . Internally, this will automatically generate a convex hull for any object. A convex hull is always preferred to a compound shape (shape collection) in terms of performance. You can also generate an explicit convex hull using the Convex Hull Tool. This will allow you to customize the hull to your needs. For more information please refer to the convex hull tool documentation for 3DS Max, Maya or XSI.