How To Perform Accurate Targeting Of Entities

BigWorld Technology 2.1. Released 2012.

Software designed and built in Australia by BigWorld.

Level 2, Wentworth Park Grandstand, Wattle St
Glebe NSW 2037, Australia
www.bigworldtech.com

Table of Contents

Chapter 1. Introduction

By default BigWorld uses the entities bounding boxes for targeting. This works well for most cases, but some times more accurate targeting is needed. This document describes a way of performing accurate targeting of entities using the inbuilt `SkeletonCollider` functionality. The `SkeletonCollider` uses hit-boxes attached to the entity skeleton for collisions, rather than using the entities bounding boxes.

Chapter 2. Creating the targeting boxes

2.1. Setup

To set up the accurate targeting, the easiest option is to use the `BigWorld` python console. The first thing we need to do is the create an instance of the model we want to create targeting boxes for. You can create an offline instance of an entity at the players position like this: (in this example we are creating an offline instance of the creature entity)

```
p = BigWorld.player()
entityDict = {'creatureType':1,'creatureName':'MyTarget'}
entityId = BigWorld.createEntity('Creature', p.spaceID, 0, p.position,
 (0,0,0), entityDict)
creature = BigWorld.entities[entityId]
```

2.2. Creating the boxes

After creating our model we need to create `BoxAttachment` objects that the `SkeletonCollider` can use for the collisions. We give the attachments the same name as the node they are to be attached to so they are easier to manage. We also give the box attachments a rough starting size.

Note

To get the names of the nodes you can enable the watcher `"render/Draw Node Labels"`, this will display the node names on the screen.

```
headBox = BigWorld.BoxAttachment()
headBox.name = 'biped Head'
headBox.minBounds = (-0.1, -0.1, -0.1)
headBox.maxBounds = (0.1, 0.1, 0.1)
pelvisBox = BigWorld.BoxAttachment()
pelvisBox.name = 'biped Pelvis'
pelvisBox.minBounds = (-0.2, -0.2, -0.2)
pelvisBox.maxBounds = (0.2, 0.2, 0.2)
```

Note

For simplicity this example only uses two `BoxAttachment` objects. In a real world game you would probably want to use more than two boxes, and you are free to use as many or as few bounding boxes that you like

The boxes are then attached to the entity's nodes so that we can use them for targeting.

```
creature.model.node(headBox.name).attach(headBox)
creature.model.node(pelvisBox.name).attach(pelvisBox)
```

We can now display the boxes by enabling the watcher `"Client Settings/displayImpactBoxes"`, this can be done by entering the following in the python console

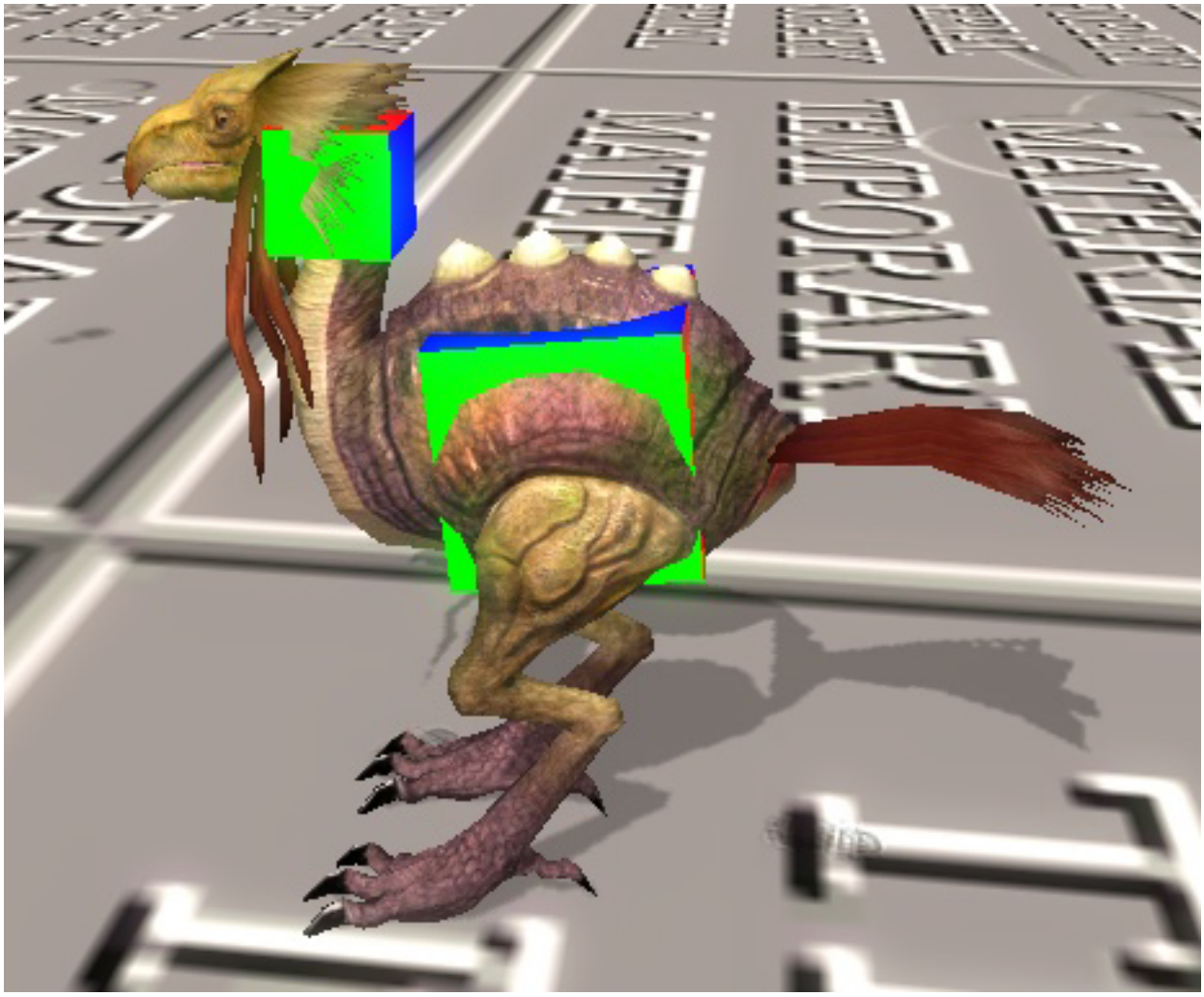
```
# This watcher is only available after the first call to
 BigWorld.BoxAttachment()
```

```
BigWorld.setWatcher('Client Settings/displayImpactBoxes',True)
```

Alternatively, this setting can be found by navigating through the watcher debugging console, (but only after creating your first `BoxAttachment`).

Note

Alternatively you could use the BigWorld maxscripts `"Add HitBox"` and `"Export HitBox to XML"` to create both the hitboxes and skeleton collider. More information on these maxscripts can be found in the maxscript section of the `"Content_creation.chm"`.



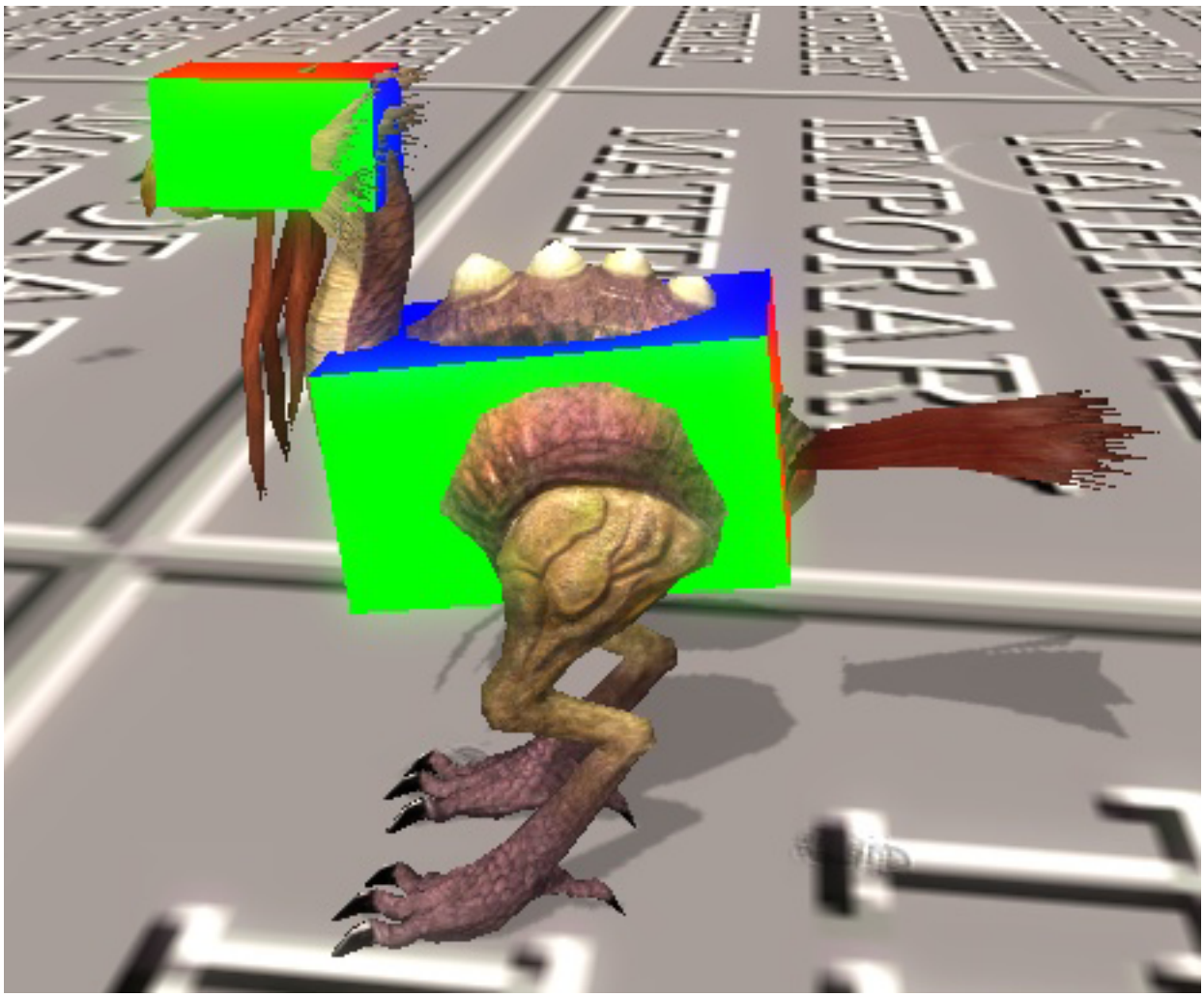
Displaying the impact boxes

2.3. Tweaking the boxes

There are two boxes attached to the body, we would like them to match the parts of the body they are meant to cover. To do this we need to tweak the boxes. There are some visual queues to which values are not correct, the bounding box `x-axis` is `red`, the `y-axis` is `green` and the `z-axis` is `blue`. We can change the min and max bounds of the boxes to something that fits the objects better:

```
headBox.minBounds=(0,-0.1,-0.05)
```

```
headBox.maxBounds=(0.2,0.1,0.3)
pelvisBox.minBounds=(-0.3,-0.2,-0.2)
pelvisBox.maxBounds=(0.4,0.2,0.2)
```



The fixed impact boxes

Chapter 3. Using the SkeletonCollider

3.1. Creating the SkeletonCollider

Now that we have created our bounding boxes we need to create the `SkeletonCollider` for the object and make it interact with the box attachments. A new instance of the `SkeletonCollider` class needs be created and added to the entity's `skeletonCollider` property, we also need to add the `BoxAttachments` to the `SkeletonCollider` so it knows which boxes to use for targeting.

```
#Create the Skeleton Collider object
creature.skeletonCollider=BigWorld.SkeletonCollider()
#Add our colliders to it
creature.skeletonCollider.addCollider(headBox)
creature.skeletonCollider.addCollider(pelvisBox)
```

Note

It is important that the new `SkeletonCollider` object is assigned to the entity's `skeletonCollider` property as this is recognised by the targeting system.

3.2. Enabling skeleton checking

Now the only thing we need to do is to enable skeleton checking in the targeting system. This will mean that any entity that has a `skeletonCollider` attribute will use this for targeting rather than the model bounding box. This will need to be done from script whenever you want to enable skeleton targeting.

```
#enable skeleton checks
BigWorld.target.skeletonCheckEnabled=True
```

You should now only be able to target the creature entity when looking at its head or body.

Chapter 4. Saving/loading the SkeletonCollider

4.1. Saving

The SkeletonCollider can be saved to a DataSection by using the save method. The following python code will create a new xml section called creature_skeleton.xml in the folder data and save the SkeletonCollider to it.

```
import ResMgr
ds=ResMgr.openSection('data/creature_skeleton.xml',True)
creature.skeletonCollider.save(ds)
ds.save()
```

4.2. Loading

The SkeletonCollider can be loaded from a DataSection by using the load method. After loading the SkeletonCollider, the colliders need to be attached to the appropriate nodes in the model. This is why it is important to name your BoxAttachments using the actual node names from the model.

```
import ResMgr
ds=ResMgr.openSection('data/creature_skeleton.xml')

# Create and load the SkeletonCollider
creature.skeletonCollider=BigWorld.SkeletonCollider()
creature.skeletonCollider.load(ds)

# Attach the collider objects to the model
for i in range(0,creature.skeletonCollider.nColliders()):
  collider=creature.skeletonCollider.getCollider(i)
  creature.model.node(collider.name).attach(collider)
```