Foreword

Thank you for purchasing the Bone Controller!

I'm an independent developer and your feedback and support really means a lot to me. Please don't ever hesitate to contact me if you have a question, suggestion, or concern.

The latest version of the documentation can be found online:
http://www.ootii.com/Unity/BoneController/BCGuide.pdf

I'm also on the forums throughout the day:
http://forum.unity3d.com/threads/bone-controller.298345/

Tim
tim@ootii.com

Contents

Overview

In today's games, having the ability to alter baked animations at run-time is crucial. Whether you're trying to adjust foot placement, ensure the character is actually looking at the target, or reacting to impact, your players expect characters to react to the environment in a natural and predictable way. Unfortunately, pre-animating every possible situation is impossible.

The Bone Controller solves this by giving you direct access to your characters' bones. It doesn't matter if it's a human, a dinosaur, or a misshapen monster, you can take control of the bones. In fact, with the Bone Controller, you can connect bones together to create whole new skeletons.

Along with the bones themselves, the Bone Controller provides joints constraints for limiting rotation, colliders for testing hits, and motors for making the bones move and react. The whole purpose of the Bone Controller was so you and I could create and share motors. Extend the existing motors or simply replace them to fit your needs.

Features

The Motion Controller supports the following features:

- Extract bone information from any skinned mesh
- Create bones from static meshes (no vertex weighting)
- Apply joint constraints to the bones
- Apply bone colliders for collision detection
- Leverage skeleton based ray-casting for bone collisions
- Use pre-built motors with your characters
- Create custom motors to drive bone rotations
- Leverage physics based bone chain dragging (pony tails, tails, ropes, etc.)
- Develop with standardized bone orientation for all models
- Blend with existing animations
- Use with prefabs
- Code included

Purpose

The goal of this document is to provide a complete guide to using the Bone Controller. This guide will not just contain information about the tool, but offer some explanations as to why I did things the way I did. Hopefully you'll find it useful even beyond the Bone Controller.

For a quick and dirty introduction, check out the quick-start guide:
http://www.ootii.com/Unity/BoneController/BCQuickStart.pdf

For information about creating motors, check out the motor builder's guide:
http://www.ootii.com/unity/BoneController/BCMotorBuildersGuide.pdf

Beta

Remember, this is beta. There are 100's of 1,000's of artists creating characters and skeletons. This tool is meant to be able to handle most (if not all) of them. However, if you have a model that isn't loading correctly, please let me know. I want to help!

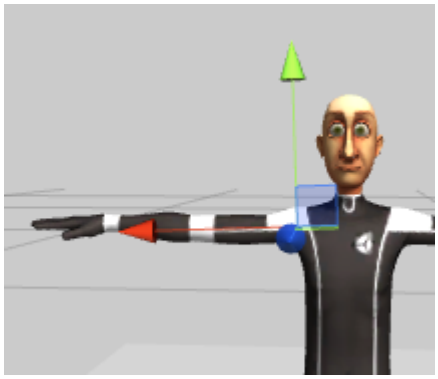
The Basics

Let's talk bones…

Bones

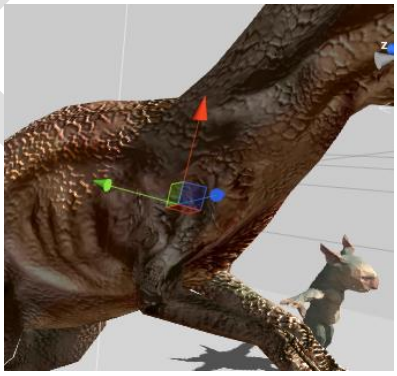
When artists create a skeleton for characters, there's no fixed way they have to do it. That means that we have characters created in all sorts of ways.

Various local transform directions of the right upper arm…



This "bone forward" is actually the transform's x-axis.

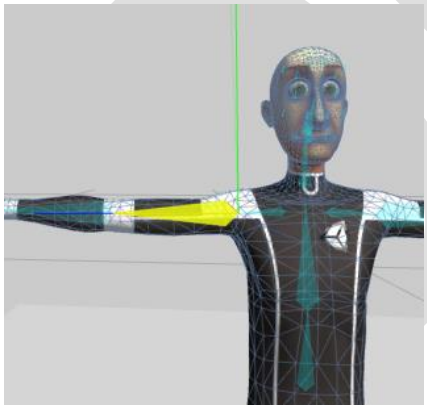
This "bone forward" is the transform's y-axis.

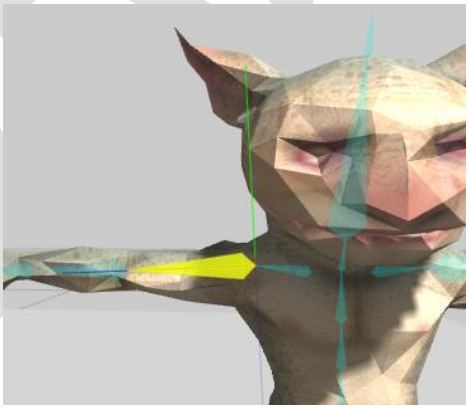


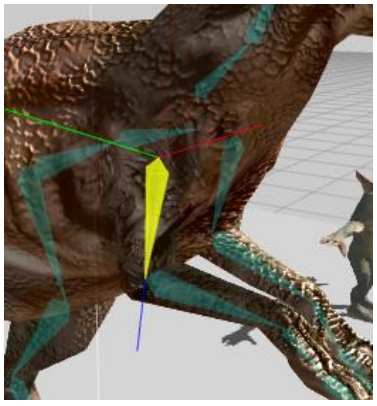
This "bone forward" is the transform's negative x-axis.

So, one of the things that the Bone Controller does is treat all bones as if they run down the positive z-axis. This is Unity's standard "forward" direction, so it only makes sense.

All bones directions are considered z-forward…







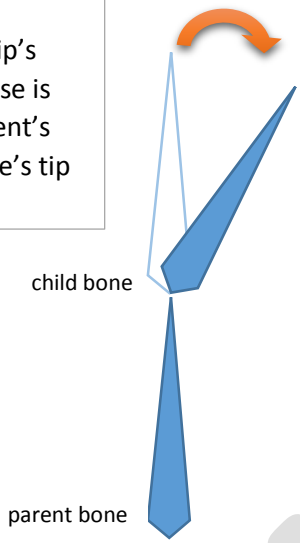
This matters because it makes manipulating bones in a consistent way much easier.

In the Unity hierarchy, these bones are really just transforms. That works pretty good, but there's no information about the bind rotation, bind position, length, etc. The bones in the Bone Controller store this.

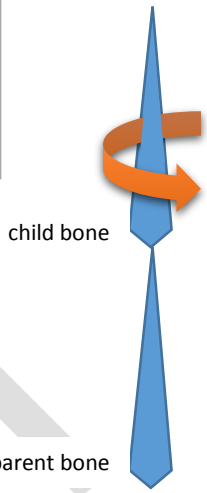
Swing and Twist

When we talk about rotating bones, there's really two ways in which bone rotate: swing and twist.

Bones **swing** by changing their tip's position. The base is fixed to the parent's tip, but this bone's tip moves around.



Bones **twist** by rotating around the axis. Both the base and tip stay in the same position, but the bone spins.

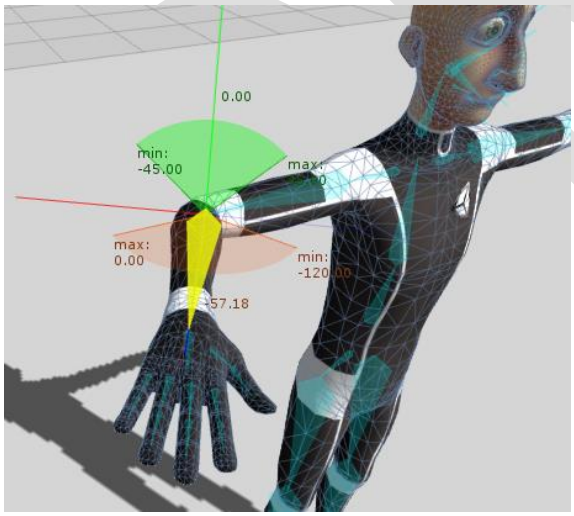


Your finger bones "swing" when they curl, but they can't twist.

Your forearm actually swings and twists.

Joints

Joints are how bones are connected. They define not only the relationship of the bones, but also how much each bone can rotate.



Think of how your forearm is connected to your upper arm. It's connected by the elbow joint. This elbow joint allows you to bend your forearm within some limits.

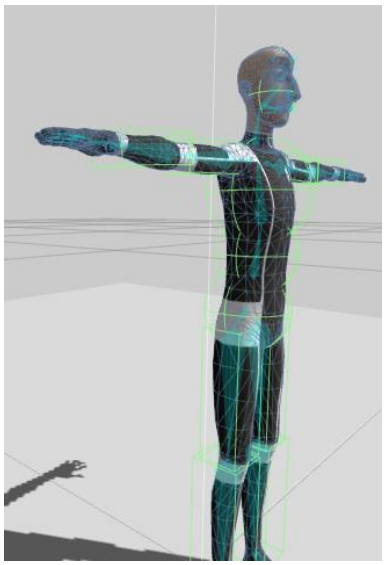
In the Bone Controller, the joint is actually part of the child bone. If you don't set a joint, the child bone can swing and twist as much as it wants.

However, if you do set a joint on the bone then you can determine how the bone will move. In the picture to the left, we only allow the bone to swing and twist within ranges.

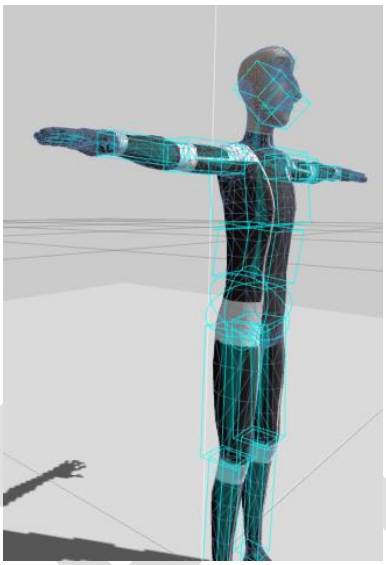
You'll see later that the Bone Controller has the ability to setup up humanoid joints and limits for you automatically.

Colliders

There's two approaches for dealing with bone colliders; "true" Unity colliders and pseudo-colliders.



Unity Colliders



Pseudo-Colliders

Why the difference?

Unity colliders are built to interact with the whole world and require special setup for avoiding other objects. Take a foot bone that has a Unity collider. It will actually hit the ground and cause issues with the normal walk animation. There's also a performance impact since there are now more colliders in the scene. Lastly, I've found that if you're using a capsule collider for your character controller, having smaller colliders that go in and out of the capsule collider (because of animations) causes odd behavior.

"Pseudo" colliders are simple shapes that don't interact with Unity's physics engine. Instead, we use simple math and custom ray-casting to determine if a bone is hit. This gives us a little performance boost and doesn't cause odd behavior within character controller capsule colliders or with environmental objects.

If you want to use both… that's fine too.

Skeletons

Skeletons are really just a collection of hierarchical bones. So, there's one 'root' bone and that root has children… which have children… and so on.

The Bone Controller Skeleton is what holds all of the character's bones. It also provides access to motors.

When I talk about the Bone Controller, I'm really talking about this skeleton. It's the top-level object responsible for managing everything.

Motors

Motors are smaller pieces of code that are responsible to making bones rotate. They typically have a specific purpose and are enabled and disabled as the game runs in order to drive the bones to specific positions.

Take a 'foot placement' motor. It's job is to make sure the knees bend and feet orient to the floor. Well, there's no need for this motor to run if your character is flying. So, you'd disable the motor until you need it again.

The Bone Controller has a set of pre-defined motors, but one of the powers of the Bone Controller is that you can create your own motors and plug them in.

To see the motors in action, check out the web demo:
http://www.ootii.com/Unity/BoneController/Demo/Web.html

Setting Up a Humanoid

Ok, let's setup a humanoid.

For a quick down-and-dirty approach, check out the Quick Start Guide.

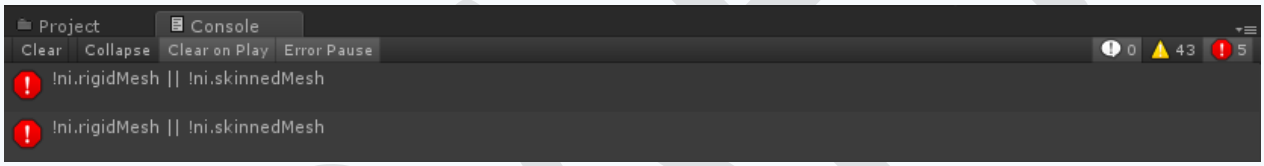
In this guide, we'll start off easy, but I'll go into the settings and gory details. This way you know exactly how things work. What I won't be doing is going into code. That will be covered in the Motor Builder's Guide.

Getting Started

1. Load a new Unity project

2. Download the latest Bone Controller package

3. Import this package into your Unity project

> As Unity imports, you may see a couple of these errors. You can ignore them. They come from the models I'm using in the demo and don't effect the actual Bone Controller.
>
> 
>

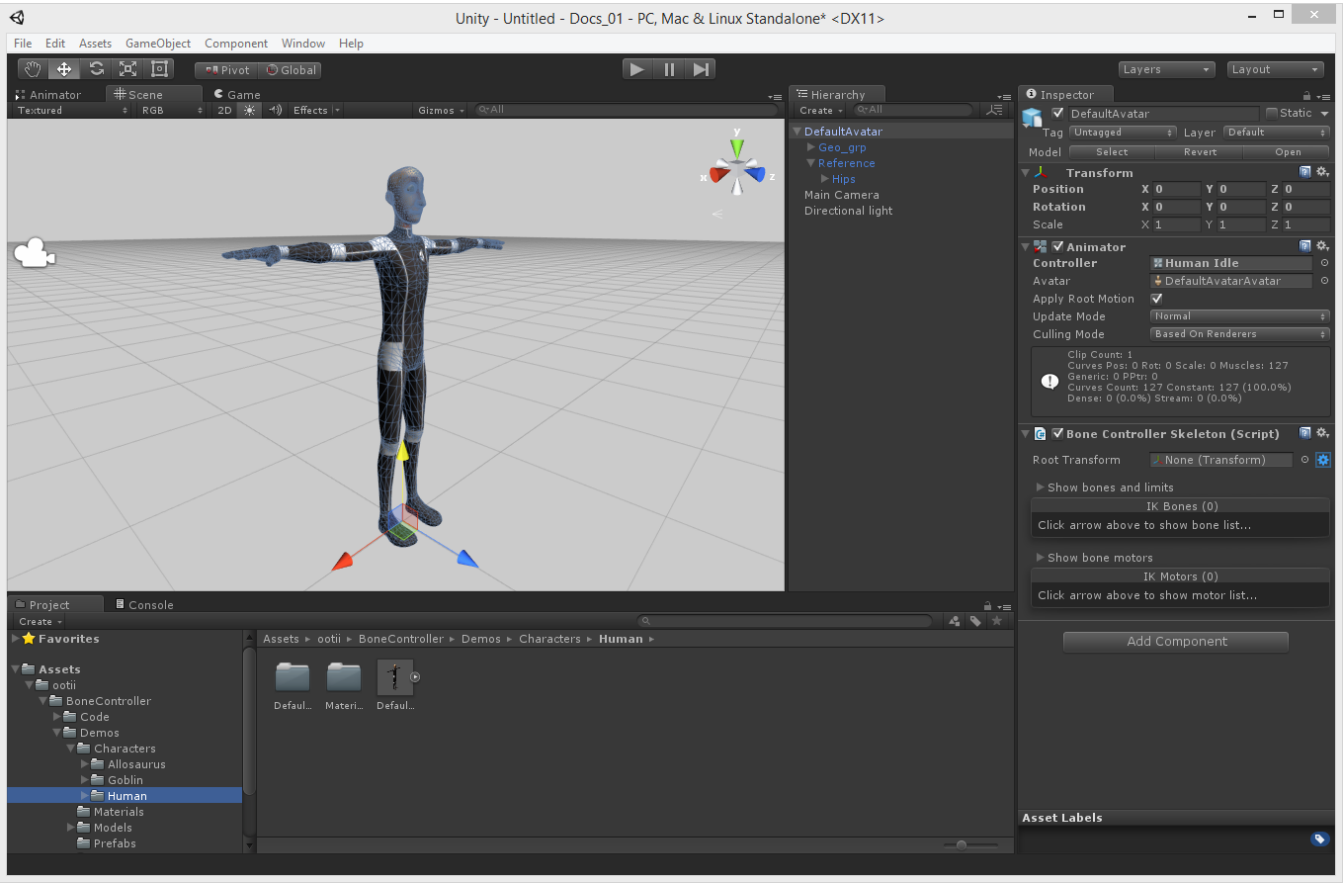
4. Create a new scene.

5. Drag a character into the scene. I'll use the Unity man and you can find him under `Assets\ootii\BoneController\Demos\Characters\Human`.

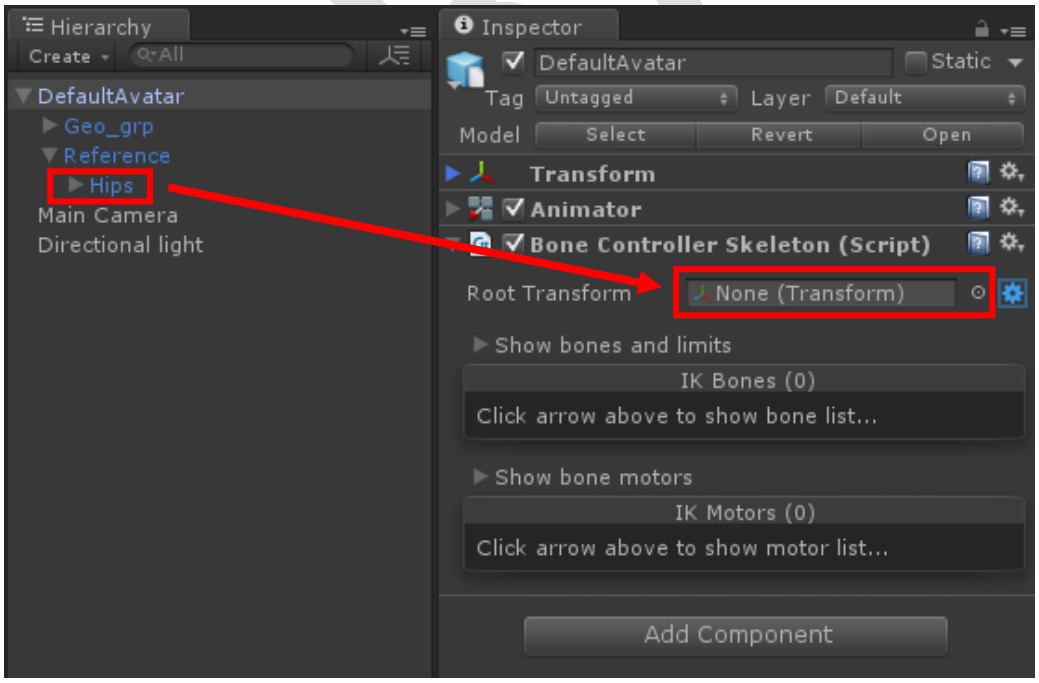
6. On the character, add a Bone Controller Skeleton

```
- Add Component
   - Scripts
      - com.ootii.Object.BoneControllers
         - Bone Controller Skeleton
```

At this point, your scene should look something like this:



7. Tell the Bone Controller Skeleton about the character's root by dragging the root bone over to the `Root Transform` property.

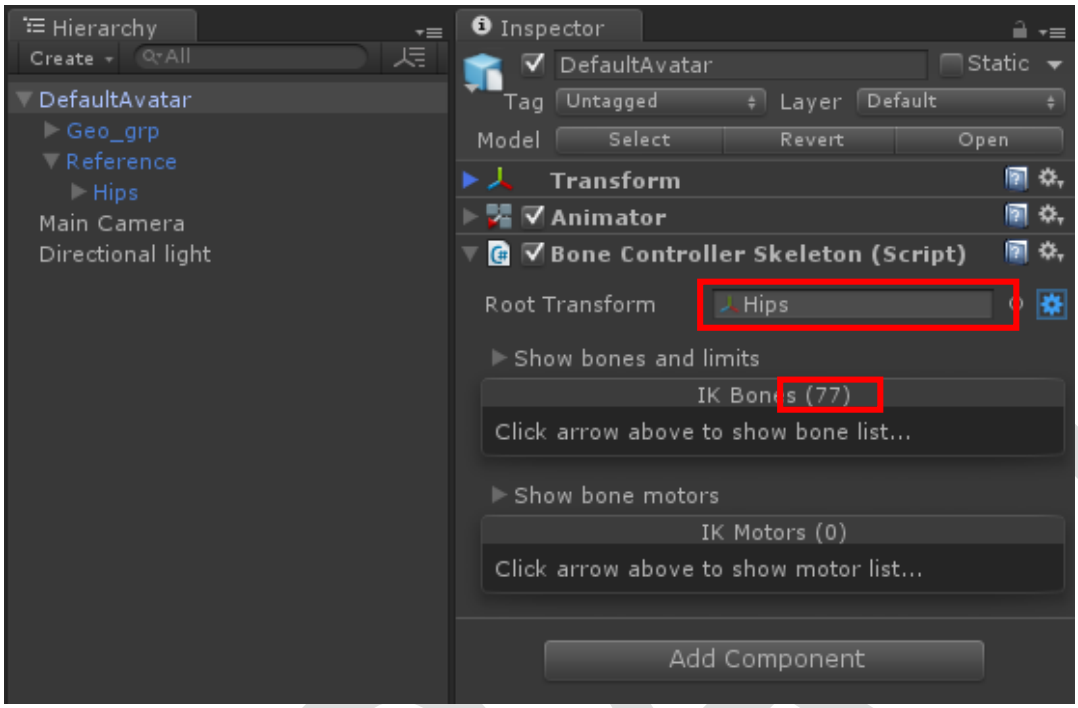


Typically the character's root is its **hip transform**.

It's really the first bone that was created in Maya or some other content creation tool.

What the Bone Controller will do is process each of the bones and set them up so we can use them in our motors. This also gives us a starting point for adding joint limits and colliders.

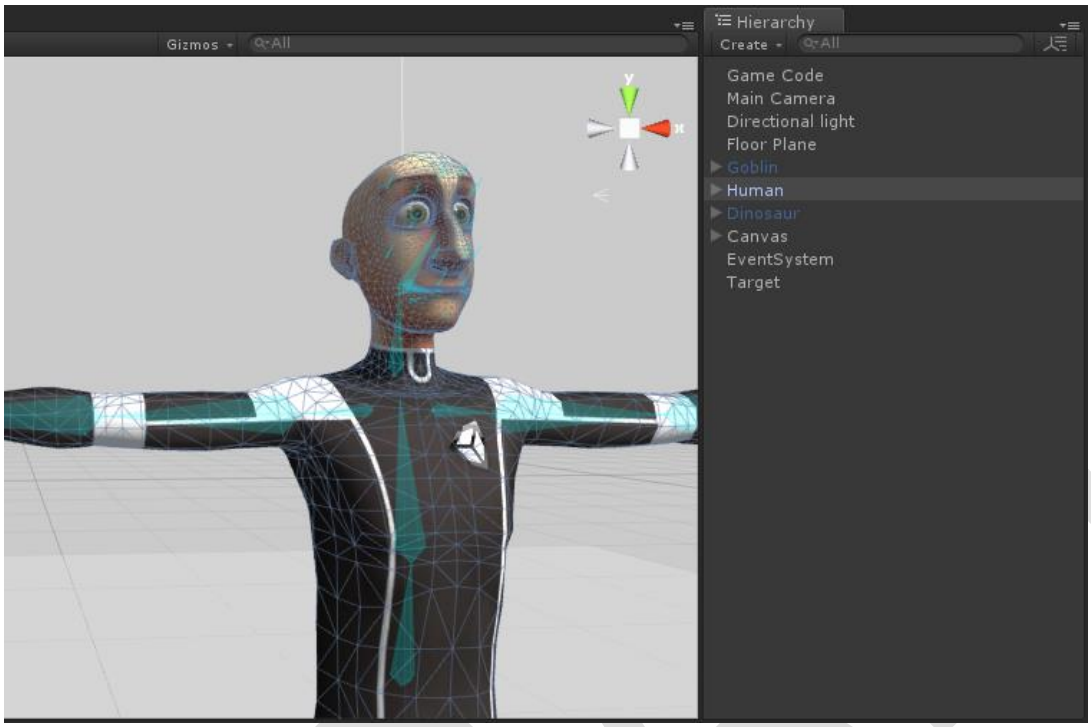
Once done, you can see the bone number has increased.



Not too bad. At this point, you could assign motors and test things out.

Bone Selection

In order to see the bones, limits, and colliders, you need to select the character in the scene. Once you do, you'll see the bones in a light cyan. In this mode, we can then select bones we want to change or add to motors.

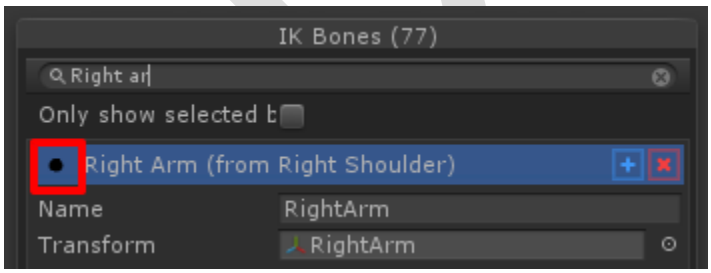


You can select a bone in a couple of different ways:

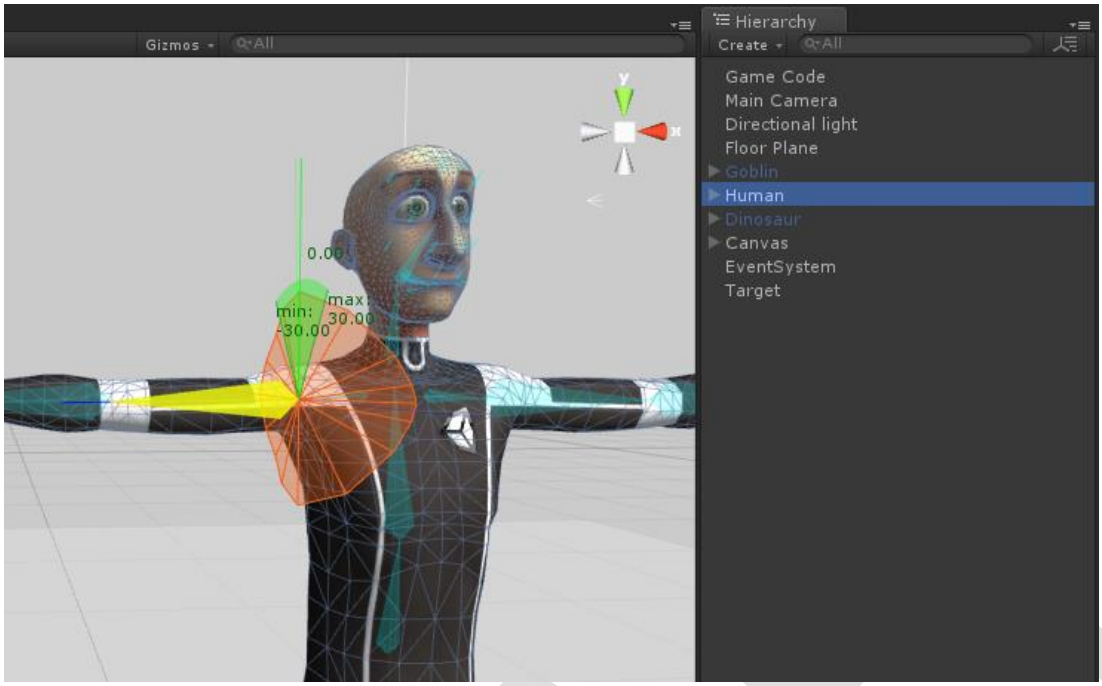
1. In the editor, select the Unity pan tool and then select a bone.

If you don't select the pan tool, Unity will think you're selecting the body's mesh and deselect the character itself.

2. In a bone list for the skeleton or motor, select the 'dot' icon next to the bone name.



Once you do that, you'll see the bone selected in the editor.



If you've selected to show bone limits, you'll also see any rotation limits that are setup on the bones. We'll talk about limits in a bit.

Choose your path…

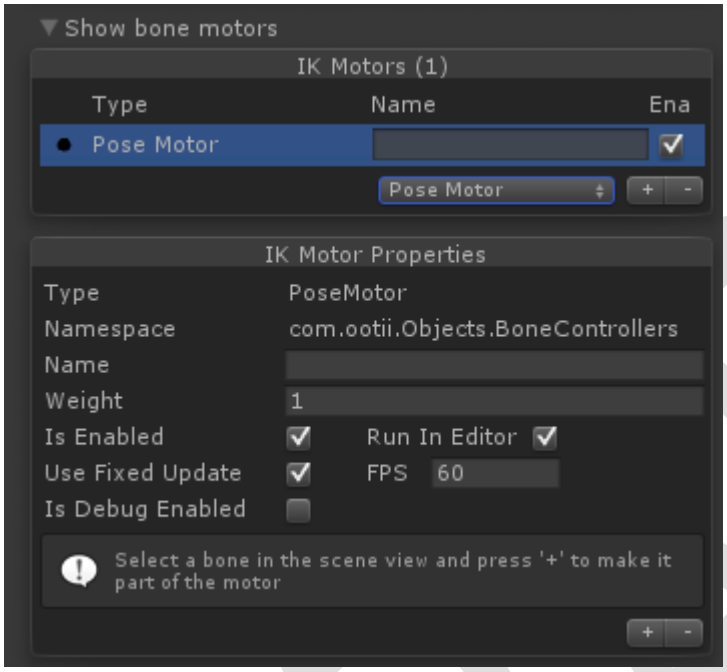
Basic: Adding a Pose Motor

Advanced: Skeleton Settings

Adding a Pose Motor

With the skeleton setup, we can now start adding motors. You can use the stock motors that come with the Bone Controller or you can create your own. If you're creating your own, check out the Motor Builder's Guide.

1. With your character selected, open the IK Motors panel by clicking on the blue triangle to the left of `Show bone motors`.



If you've used the Motion Controller this setup will look somewhat familiar.

First will be a list of enabled motors and under that are the properties associated with the motor.

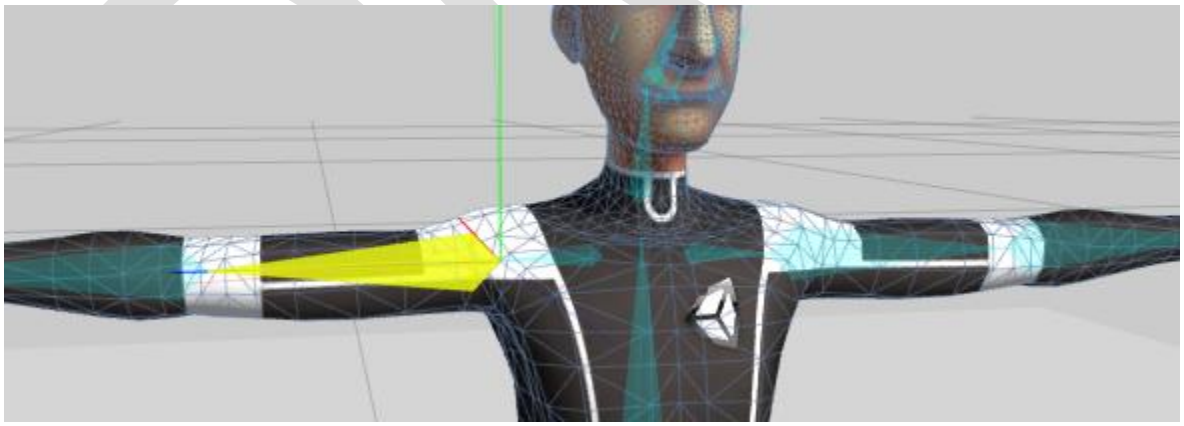
Right now, we have no motors added.

2. Click the drop down and select the `Pose Motor`. Then press the '+' button.

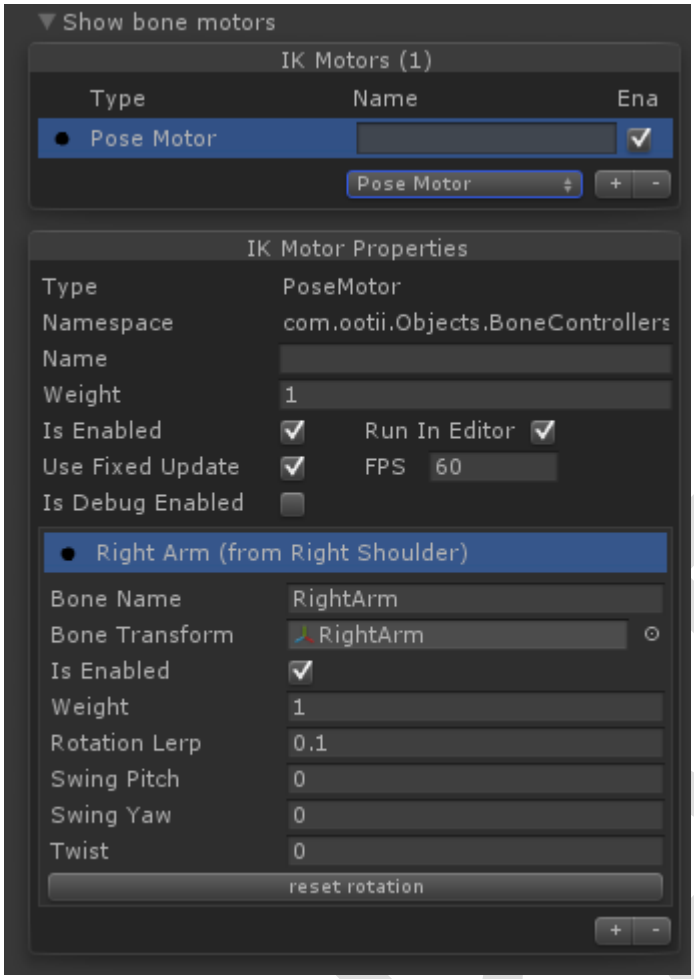
The motor itself has some important properties. See the Motor Settings section for more detail.

You'll actually see these properties on pretty much every motor…

3. With the pose motor, we need to select the bones we want to pose. This keeps us from wasting computing cycles on bones we don't want to manage. Using the steps above, select the right upper arm bone.



4. Then, back in the Pose Motor properties window, press the '+' button at the bottom. This will add the bone to the list of bones the motor controls.

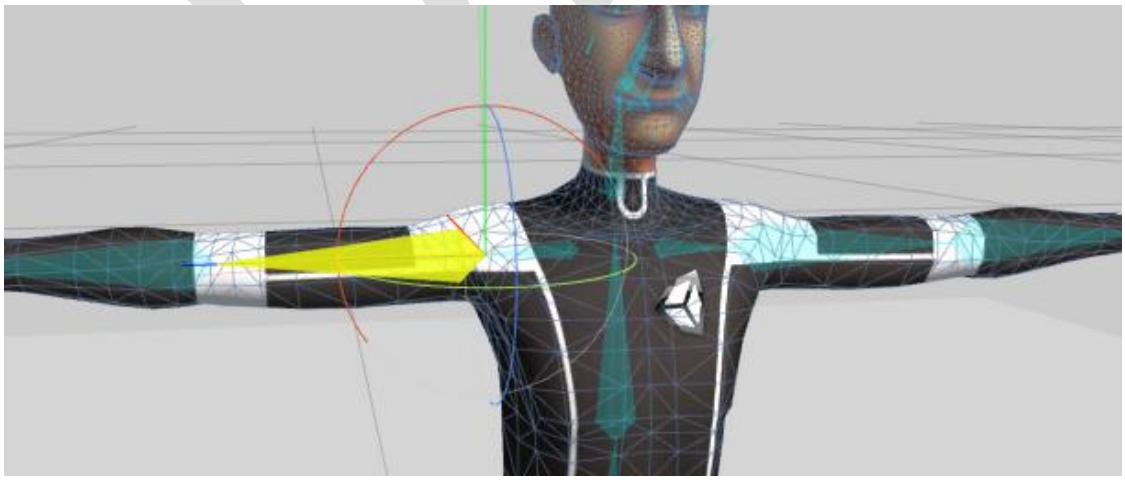


You can see that the Right Arm bone was added to the list. The list isn't actually a list of bones, but more a list of bone wrappers. These wrappers give us extra information to apply to the bones themselves.

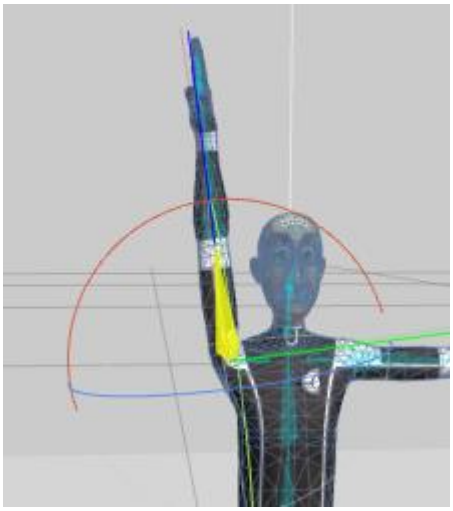
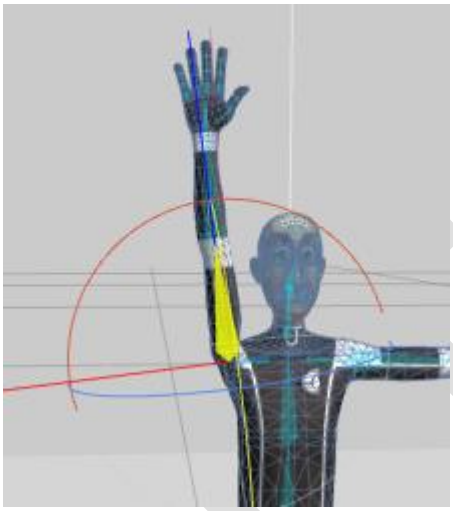
In the case of the Pose Motor, the extra information are things like swing and twist.

5. Pose the bone

When we added the bone, you probably noticed that the scene view updated to include GUI manipulators. You can use these or the Swing Pitch, Swing Yaw, and Twist fields to set the pose you want.

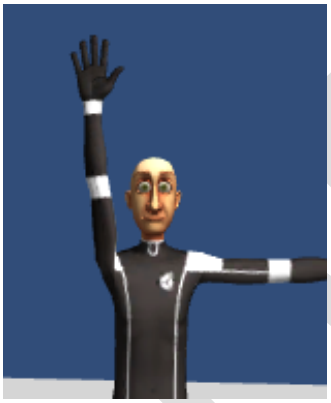


Let's swing the bone down (using the red rotation handle). Then, twist the bone so the palm of the hand is facing forward (using the blue rotation handle).

Remember, we're only posing the right upper arm. The right forearm, hand, and fingers are just coming along for the ride.

6. Run the scene



When we run the scene (with no animation), we see we get the pose we set.

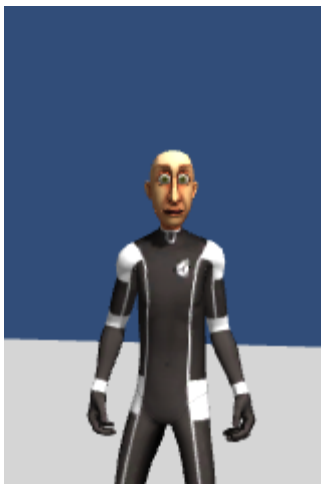
In this case…

- The right shoulder is in the bind rotation.
- The right upper arm is posed (weight was 1).
- The right forearm is in the bind rotation.
- The right hand is in the bind rotation.

If we had set an animation to run (say the idle animation), our bone pose will blend with the currently running animation.

In this case…

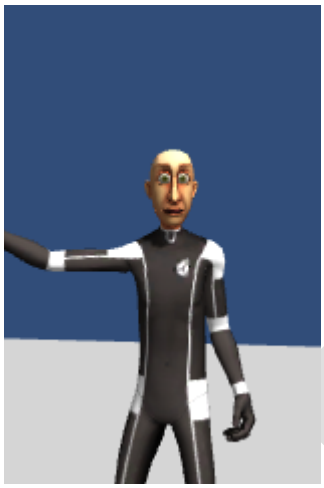
- The right shoulder is animated.
- The right upper arm is posed (weight was 1).
- The right forearm is animated.
- The right hand is animated.



Here, we're running with an animation, but I set the bone weight to 0. That means the right upper arm's pose is completely controlled by the animation. Our motor did nothing.

In this case…

- The right shoulder is animated.
- The right upper arm is animated (since weight was 0).
- The right forearm is animated.
- The right hand is animated.



Here, we're running with an animation, but I set the bone weight to 0.5. That means the right upper arm's pose is half controlled by the animation and half by our motor.

In this case…

- The right shoulder is animated.
- The right upper arm is animated and posed (since weight was 0.5).
- The right forearm is animated.
- The right hand is animated.

Choose your path…

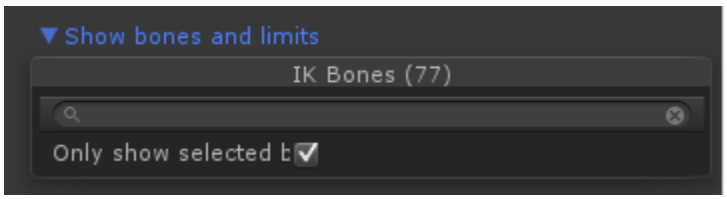
Basic: Adding a Bone Joint

Advanced: Motor Settings

Adding a Bone Joint

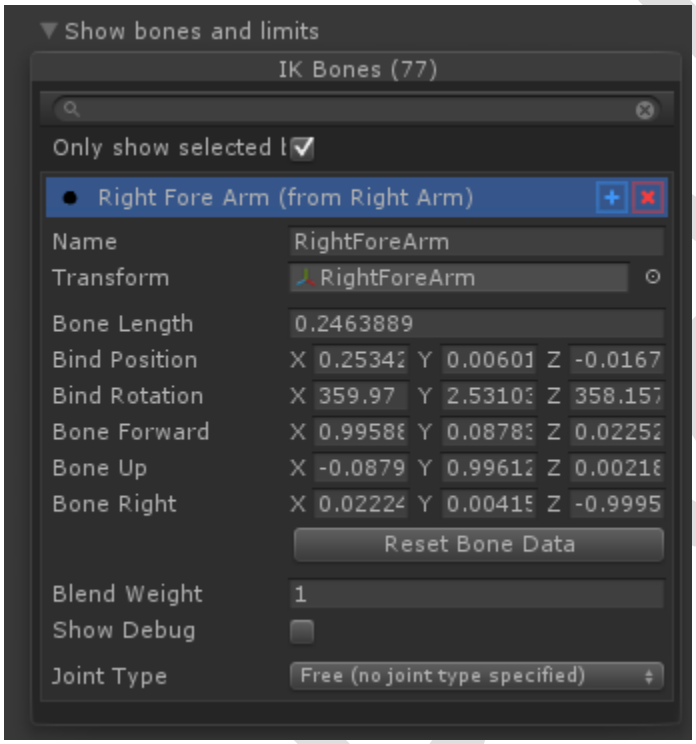
With the skeleton setup, we can add a bone joint to limit the rotation of the bones. Each bone can have one joint. Let's add a hinge joint to the right forearm.

1. Open the IK Bones section of the inspector. When you do, you'll probably just see a check box.



When checked, we'll only show you the selected bone. If you don't see anything, you probably don't have a bone selected.

2. Using the previous steps, select the right forearm.



There we go, that looks better.

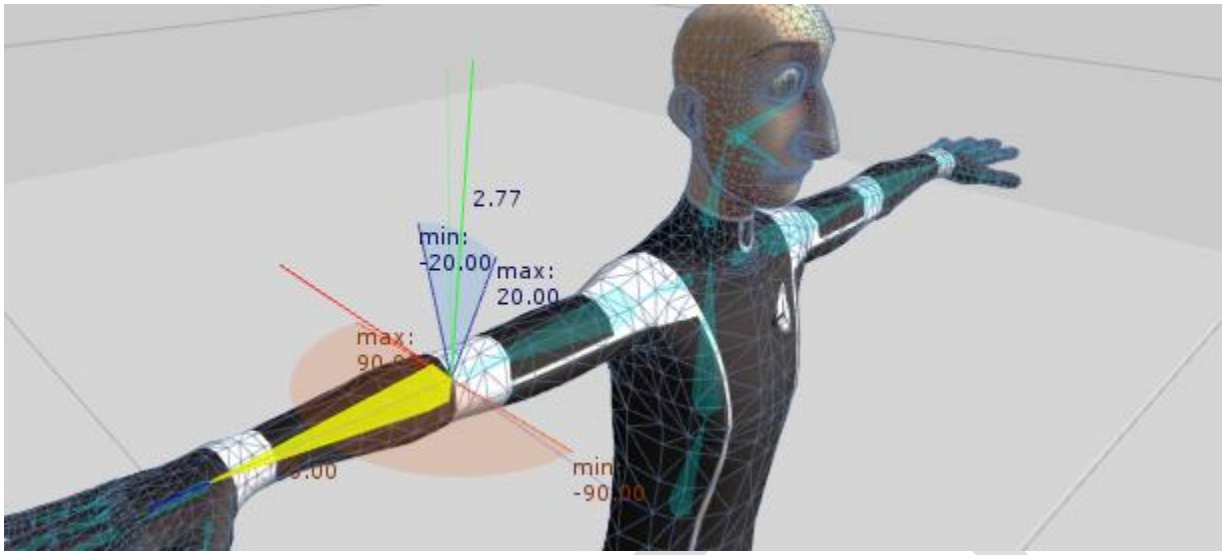
Ignore the top section for the time. If you want to know more about it, head to the Bone Settings section.

For now, all we really care about is that last drop-down. This defines the current joint type.

3. Change the Joint Type to 'Hinge Swing and Twist'. This type allows the bone to rotate on one axis (like a door hinge does), but also to twist.

Two things happened; the scene editor updated and this inspector updated.

The scene editor just shows you the current limits.



Notice how the orange shows you the plane on which the bone will swing. While the blue is a gauge for how much twisting is allowed.

4. Back in the inspector, we can set those limits.

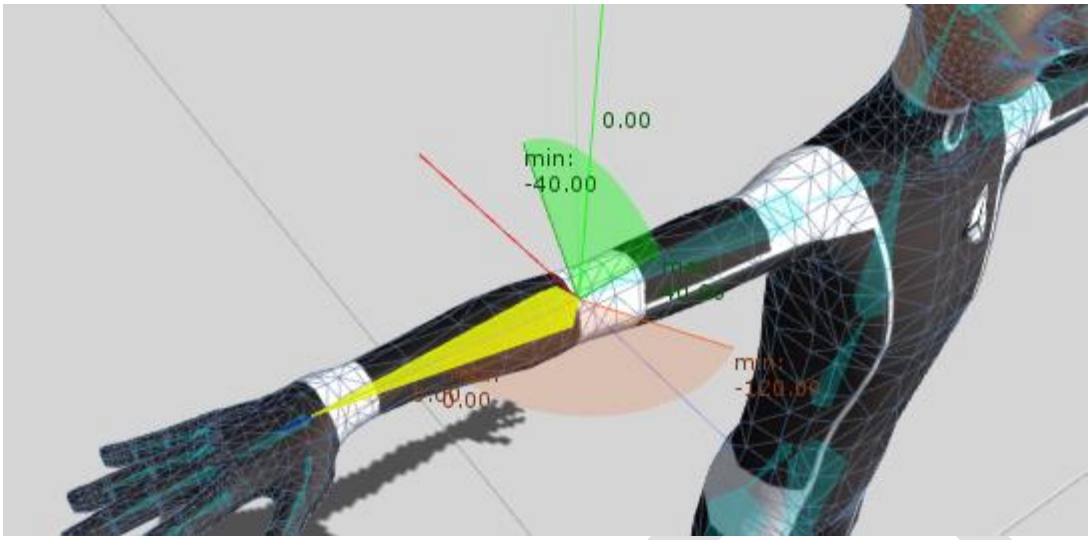
First, we can set the swing axis. This is the axis around which the hinge joint is allowed to rotate.

This axis is relative to the bone's forward direction (z-axis) when in the bind pose. By default it's the bone's up direction (as indicated by the green line point up). Perfect as is.

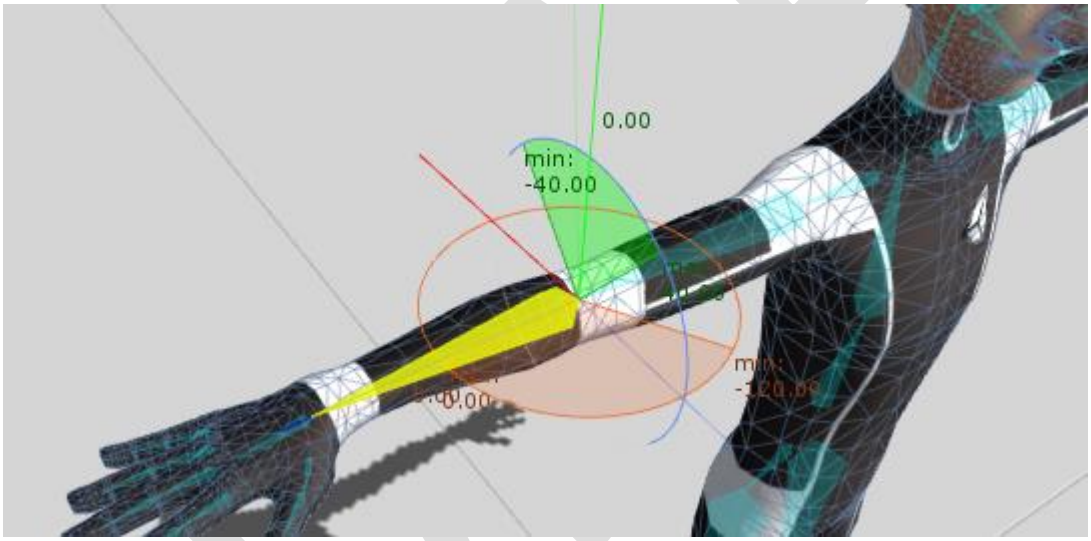
Next, we can determine how much to swing. Set the min value to -120 and the max value to 0.

Finally, determine the twist by setting it to -40 and 40.

You can see the updated changes in the scene:



5. Let's test the limit by adding this bone to the list in the Pose Motor. Do it like we did in the last section.



Notice how the rotation handles appear. There's a blue one for the bone forward (z-axis) and an orange one for the hinge axis. You can rotate with these handles and see the bone won't go beyond the limits.

> If you drag the handle really fast, the bone may jump to the opposite side of the limit, but that's expected as it's still within the limits. You just rotate in a full circle.

It's probably worth clarifying that joints don't move bones. They just limit them. Motors move bones. So, just because you added a joint doesn't mean you can rotate it. Once you add a motor, that's when the magic happens.

Basic Wrap-Up

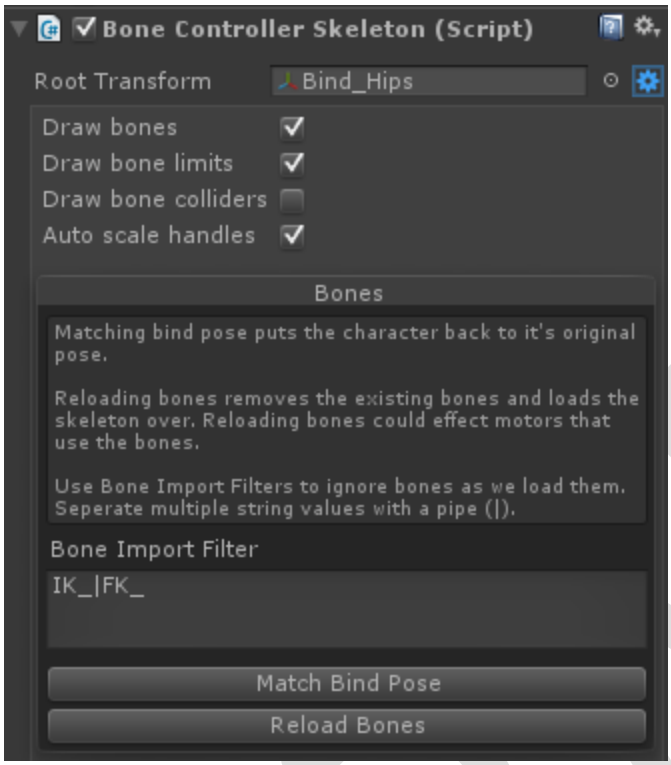
So, we've added a skeleton, selected a bone, added a Pose Motor, and then set limits. That's the high level of what we're doing. With this as a base, we can:

- Modify existing bones
- Create new bones
- Add different joint types
- Add different motor types
- Create new motors

The rest of this document goes into more detail about each of these and shows you how you can get the most out of the Bone Controller.

Skeleton Settings

To the right of the Root Transform is a blue gear icon. This shows and hides the high level settings that are available.



In the first section:

Draw bones – We'll render the bones in the editor when the character is selected.

Draw bone limits – We'll render the currently select bone's limits (if they exist) in the editor.

Auto scale handles – Allows the GUI handles to resize as you zoom to and from the character.

In the second section:

Bone Import Filters allow you to specify bone names (or partial names) that should not be included in the skeleton.

For example, if you import a Maya rig it could have twice as many bones as expected. This is because these transforms are part of the control rig. Fortunately, these transforms have consistent names like "IK_Hand" or "FK_Foot". So, by setting filter values here, we can ignore bones with these values in their names.

Entering "IK_|FK_" for example ignores all bones whose names include "IK_" or "FK_".

Sometimes it's handy to force the character back to the bind pose. This is the pose that the character was actually created in.

Press the **Match Bind Pose** button to do that.

If you've accidentally monkeyed up the individual bone settings, you can use the **Reload Bones** button to reimport the whole skeleton.

> When you reload the bones, new bones are created. Most motors will reconnect to the new bones, but you may need to re-setup some motors. Only do this if you really need to.

Automated Joint Setup

The Bone Controller can automatically create joints for your humanoid character. It does this using the same bone IDs that you used when you imported the character and set its rig to 'Humanoid'.

The **Style** and **Detail** options define how detailed we get in setting up the joint limits. Currently only 'Humanoid' and 'Low' are defined.

The results of using the **Set Joints** button when these options are selected is as follows (see Joint Definitions for more information):

Head	Joint limits swing and twist
Neck	Joint limits swing and twist
Chest	Joint limits swing and twist
Spine	Joint limits swing and twist
Right Upper Arm	Joint limits swing and twist
Right Lower Arm	Joint limits swing to 1 axis with twist
Right Upper Leg	Joint limits swing and twist
Right Lower Leg	Joint limits swing to 1 axis with twist
Left Upper Arm	Joint limits swing and twist
Left Lower Arm	Joint limits swing to 1 axis with twist
Left Upper Leg	Joint limits swing and twist
Left Lower Leg	Joint limits swing to 1 axis with twist

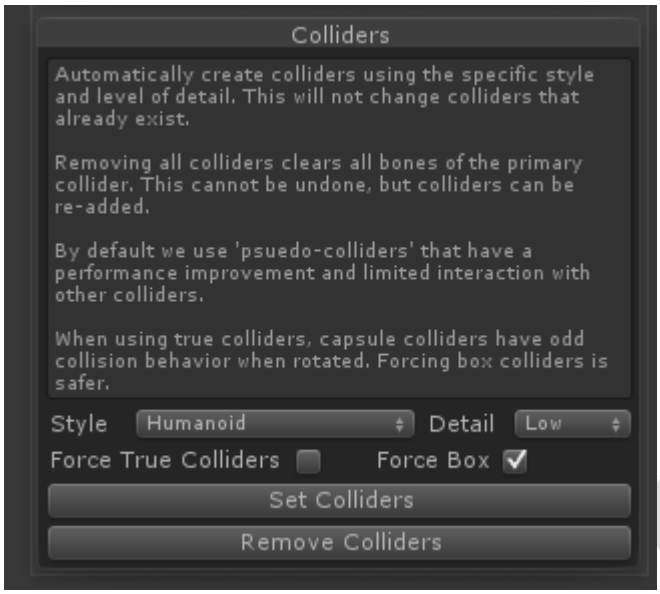
Note that if a joint already exists on the bone, the existing joint won't be modified and no new joint will be added.

Future updates will add more joints for hands, fingers, and feet.

The Bone Controller Skeleton was written so you could inherit from it and then override the functions that build these limits. This way you could customize them as needed. To see the exact limits, see `BoneControllerSkeleton.SetHumanoidBoneJoints()`.

The **Remove Joints** button will remove all joints from the skeleton.

Automated Collider Setup



Colliders are similar to joints in that they can be automatically added for a basic humanoid setup. For the difference between Unity colliders and "Pseudo" colliders, check out the introduction to colliders.

Using the **Style** and **Detail** options will determine how the colliders are added.

When "Humanoid" is selected, colliders will be created for specific bones based Unity's Humanoid rig. Using the Detail option will determine which bones have colliders placed on them.

For example, "Low" detail will only put colliders on major bones while "High" detail will put colliders on every bone (including finger bones)…which is way over-kill.

When the style of "Other" is selected, we'll build colliders based on the size and orientation of the bone. The detail option will determine the minimum size bone that gets a collider.

Force True Colliders
If you want to use Unity colliders instead of pseudo-colliders, check this checkbox. However, I don't suggest this as it can impact performance by adding these extra colliders and you may find odd behavior as bones collide with the environment.

Force Box
Typically we'd want the bone colliders to be capsule-colliders. However, in testing I found that the Unity capsule colliders do not rotate well with bones for some reason. When this happens collision occur when they shouldn't or not at all.

The **Force Box** check box will build colliders using box-colliders. I've found these colliders work much better.

The Bone Controller Skeleton was written so you could inherit from it and then override the functions that build these limits. This way you could customize them as needed. To see the exact colliders, see `BoneControllerSkeleton.SetBoneColliders()`.

Remove Colliders
The **Remove Colliders** button will remove the first collider from each bone in the skeleton.

For now, let's keep talking about the bones themselves.

Bone Settings

Under the hood, we're managing data about each bone so that motors can deal with bones consistently. By clicking the triangle next to **Show bones and limits**, you can see (and even change) that data.

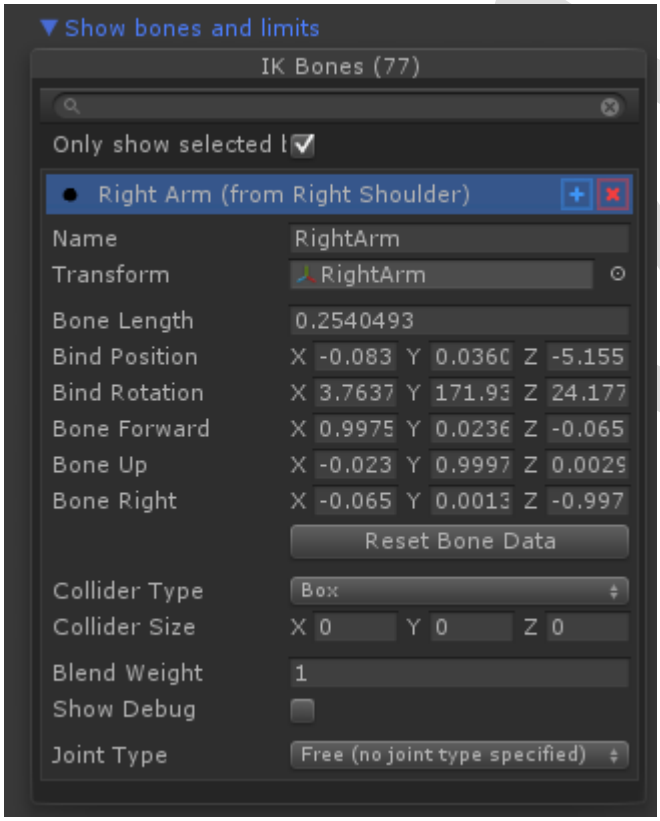
> If the section opens and you see no bones, either select a bone in the scene editor or uncheck the **Only show selected bone** option.

Each bone detail contains two primary sections: bind data and joint information.

Bind Data

The bind data is information that we use to track how the bone was oriented when it was imported into Unity. Remember way in the beginning when I showed bones being defined with different rotations? This is how we standardize them all.

This information maybe too detailed, so feel free to skip it. However, if you ever plan on modifying the bones (yep, you can) or creating new bones (yep, you can), you'll need to know this.



Name – Friendly name we give the bone. You can use this to change the bone transform by searching for a new transform to tie the bone to. I suggest you not do that.

Transform – The transform this bone is tied to

Bone Length – Length from this transform to the child transform (if it exists). If more than one child exists, we use the average length.

Bind Position – Local position of this bone relative to the parent bone.

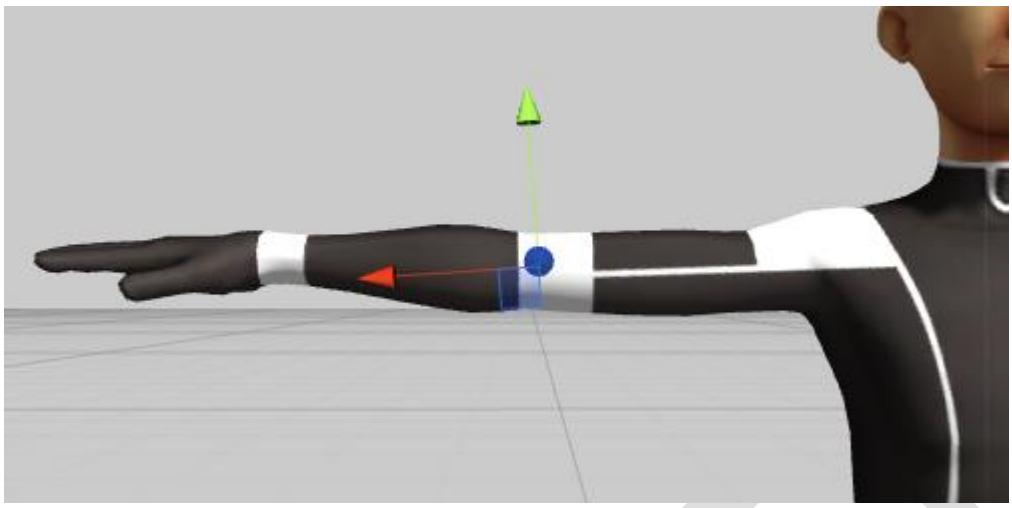
Bind Rotation – Local rotation of this bone relative to the parent bone.

Here's where things get tricky…

Remember when I said that we treat all bones as if their "forward" direction is running down the length of the bone? Here's where this information is stored and where it can be modified if needed.

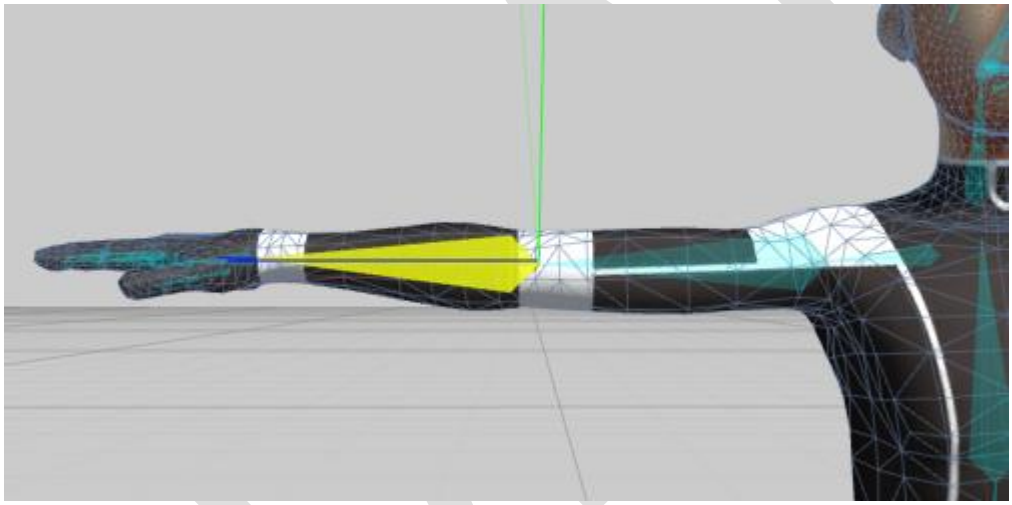
Let's take the character's right forearm as an example:

When the character was created, the artist determined that the right elbow joint would be a specific position from the right upper arm (bind position) and have a specific rotation (bind rotation). In this case, it results in the forearm pointing down the positive x-axis.



Clicking on the RightForeArm transform in the project hierarchy shows this. So, technically, the "right" (red) axis is the bone's forward direction.

However, we apply a rotation internally that changes it so the "forward" (blue) axis becomes the bone's forward direction. That information is then stored in **Bone Forward**, **Bone Up**, and **Bone Right**.



I know this sounds odd or maybe worthless, but it gives us the ability to change how a bone is oriented. In turn, that can make creating motors much easier. I'll go into that later.

If you've ever changed this information, press the **Reset Bone Data** to regenerate the original bind information.

Collider Type allows you to choose type of pseudo-collider that we'll use. Currently the options are 'Box' or 'Sphere'. Both of these are based on the bone's base position.

Collider Size/Radius allows you to choose the size of the collider based on the top. These sizes are based on the bone's orientation.

Blend Weight determines how much we blend the original animation-generated rotation with the motor-generated rotation. If we set the value to 0, none of the motor-generated rotations matter. We just use pure animation data from Mecanim. If the value is set to 1, we override the animation data with motor data.

Show Debug allows us to enable extra information to the screen about the bone. Currently it isn't used.

Creating and Adding Bones

This has become a pretty useful feature. Using the Bone Controller, we can actually create and add new bones and bone chains to the skeleton.

The Bone Chain Drag Motor demo is a perfect example. In this example, I created a pony-tail for the goblin.

By linking these normal Unity capsules together and adding them to the skeleton, I'm able to create bones.

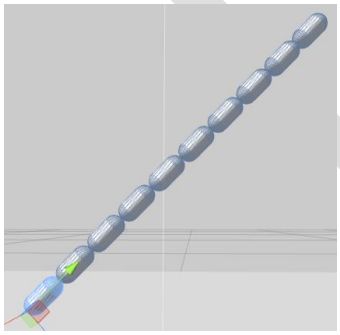
Obviously they don't live outside of Unity, but they pose, constrain, and work like any other bone.

You can change their settings and you can use them with any motor.

Using this approach, you could also take a skinned mesh (like a tail) and attach it to another skeleton. For all intents and purposes, those bones are now part of the full character skeleton.

To do this:

1. Create the structure (mesh) of objects or import the skinned mesh

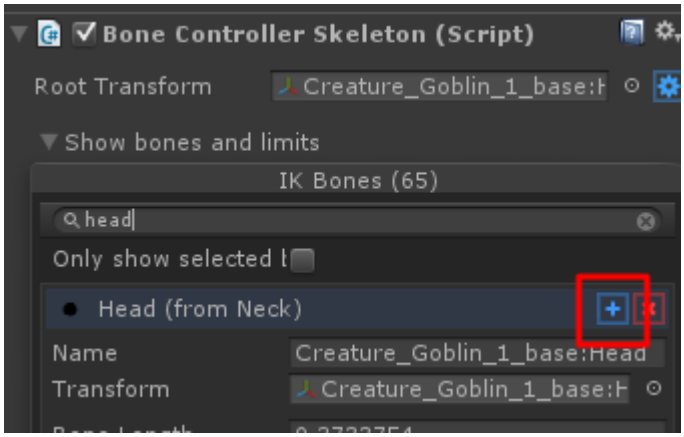


For this example, I took capsule and moved its pivot point to the bottom center using the great little tool from Yilmaz Kiymaz.

Then I copied and stacked the capsules. Using Unity's project hierarchy, I made the second one a child of the first, the third a child of the second, etc.

Finally I moved the whole system to where I want on the goblin. Right where a pony-tail would be.

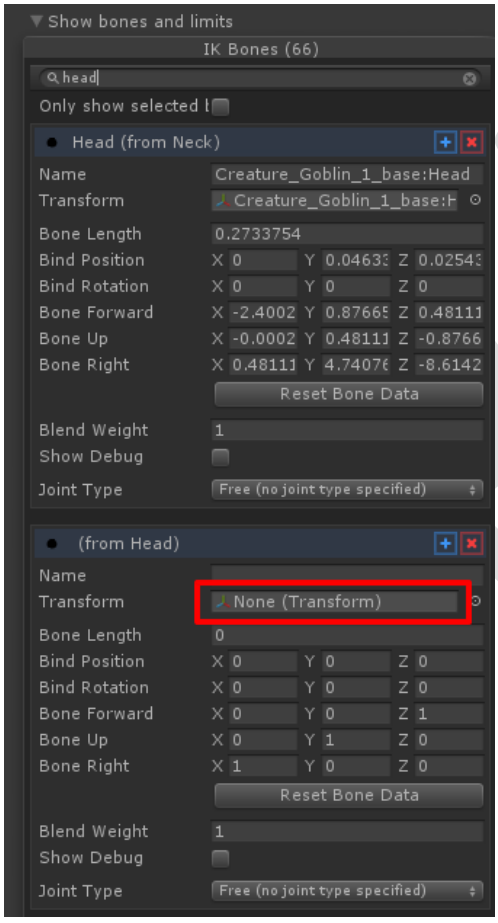
2. In the IK Bones list for your character, find the bone you want to tie this mesh to. In this case, it's our goblin's head.



3. With the bone found, press the blue '+' symbol to add a new child bone.

Under this bone, you'll see an empty bone is created in the view.

4. Now, drag the root of your new mesh structure or skinned mesh into the new bone's Transform field.



Once you do this, the new mesh structure (or skinned mesh) becomes parented to the bone you added to. It will move in the hierarchy and act like any other child bone.

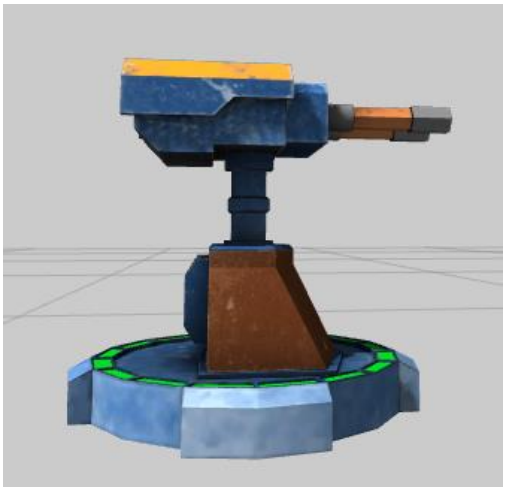
Use it with limits, motors, etc.

Removing a Child Bone

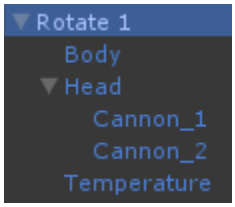
As you can imagine, removing a child bone is just a matter of pressing the red 'x'. Note that I don't delete the object. I simply remove it from the skeleton and place it in the root of the scene.

Skeletons From Static Meshes

Another great use for creating bones is for creating a skeleton for objects that don't actually have one. Take this awesome turret from the Turrets Pack by Vertex Studio.

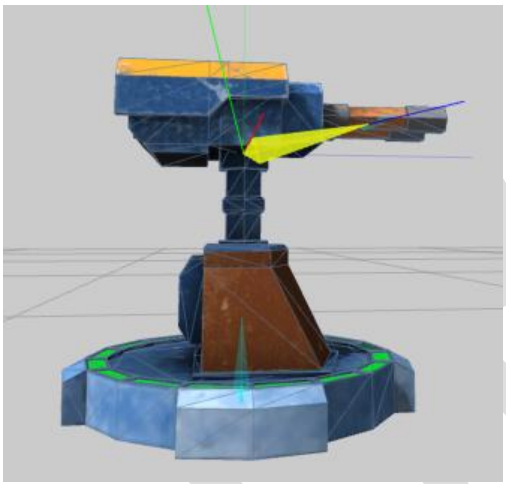


It's a hierarchy of static meshes



No bones, no animations, just meshes.

Using the Bone Controller, we can assign it as a skeleton and bones would be extrapolated out of the meshes.



Now that we have bones, we can use the motors to have it look at a target, rotate over time, respect rotation limits, or create new motors to control its behavior.

In this case, we now have bones for the base, the head, and one for each of the canons. The head could be rotating towards a target while the canons are spinning.

To do this…

1. Select your static mesh object.

2. Add a Bone Controller Skeleton Controller component.

3. Drag the static mesh into the 'Root Transform' field of the Bone Controller Skeleton.

Joint Settings

Note: In the pictures, you'll see the "twist limit" is in green. However, the twist limits are now blue to match the standard z-axis (our bone forward).

Joints are used to limit how the bone can rotate. Without a joint specified, the bone is free to swing and twist however we want.

To learn more about bone rotations, see the introduction section on swing and twist.

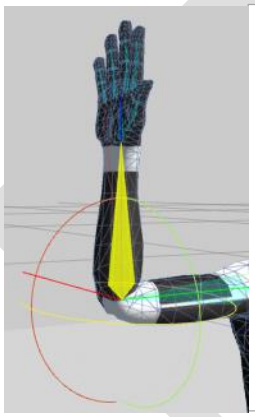
The reality is that a lot of times, having no joint is fine. After all animations will typically control the rotation and they don't need limits. However, when we start using motors and allow the environment to change the animation data, limits can ensure we don't do something whacky.

Besides "no joint specified", there are four types of joints.

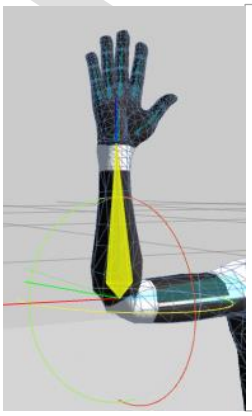
> It's important to realize that joints DO NOT rotate bones. Joints limit bone rotation. Setting a bone joint does not automatically enable rotation handles... setting a motor does.

Preventing a Swing's Twist

Through the natural swinging of the bone, the bone may seem like it twists. However, that's different than actually manipulating the twist. Let me show you what I mean…



This pose was created by using the red handle and pulling the bone up.



This pose was created by using the green handle and pulling the bone forward. Then, using the red handle and pulling the bone up.

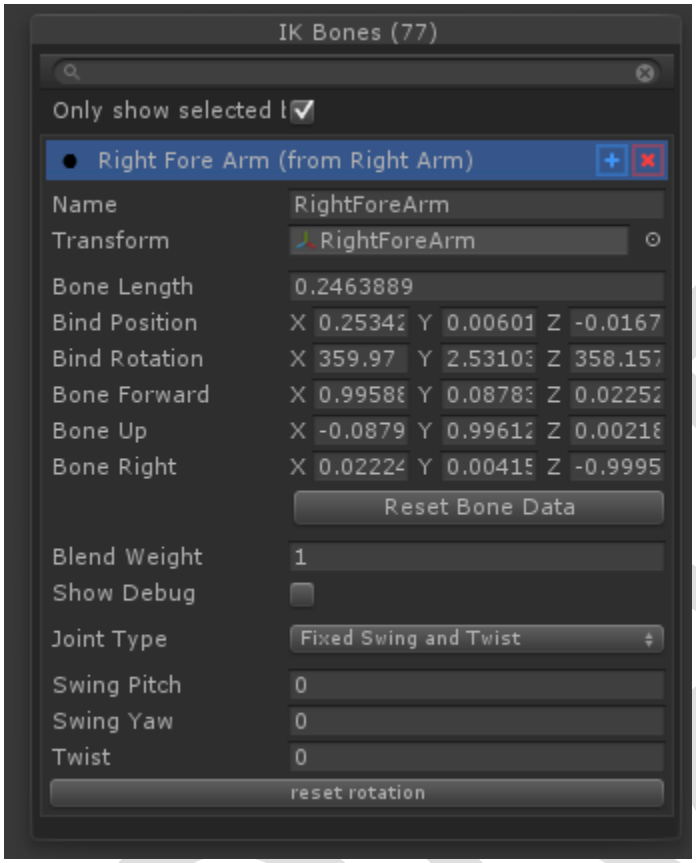
It may look like the bone was twisted, but it was actually part of the swing. Each of the joints has a property **Prevent Swing Twisting** that will attempt to remove this kind of twisting.

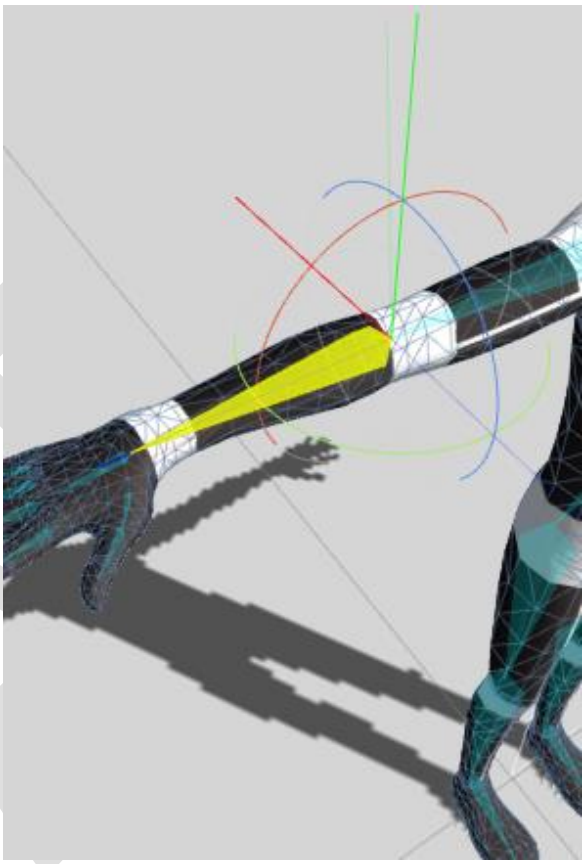
Just be aware that as you're using this field and swinging the bone in the editor, it will be fighting with you a bit to keep extract the twist from the swing as you're swinging.

Fixed Swing and Twist

This joint is sort of unique in that it locks the bone in place. With this joint, we fix or weld the bone so it won't swing or twist. It's the only joint that allows us to rotate the bone from the joint inspector.

The reason for this is that you are setting the rotations as the limits.





Once selected, you can change the swing using either the inspector or using the handles in the scene.

Swing Pitch (red handle) – This rotates the bone around the bone's x-axis or "Bone Right"
Swing Yaw (green handle) – This rotates the bone around the bone's y-axis or "Bone Up"
Twist (blue handle) – This rotates the bone around the bone's z-axis or "Bone Forward"

> If you were to add a Pose Motor to the Bone Controller and add this bone to it, you'd see there are no options to pose the bone. That's because it has a fixed joint.

Free Swing and Twist

This joint allows the swing to go anywhere, but limits the twist.

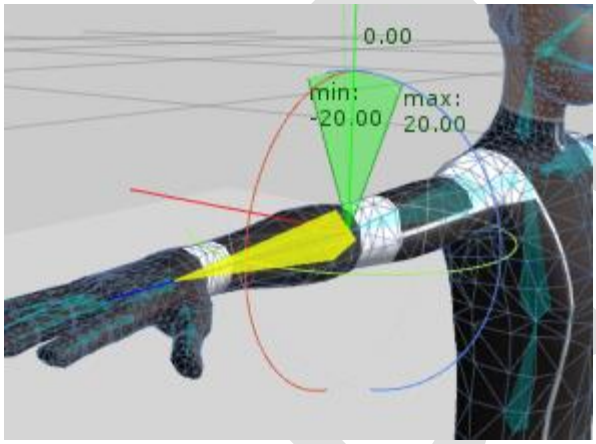
Since it's a "free swing", the only thing we can limit is the twisting.

Prevent Swing Twisting:
When checked, we'll attempt to remove any twisting that occurs due to the swing itself. Remember we talked about this a while back?

When checked, if you're swinging the joint in the editor, you may get some jumpy behavior. This is the joint removing the twist from the swing as you're swinging.

Allow Twist:
Determines if we will allow a non-swing-based twist to occur.

Limit Twist:
Determines if we'll limit the range of the twisting. If we do, you can set the range here. Note that the twisting is always relative to the bone's up axis (the green line).



In the scene, the allowable twist range will be shown in blue (which I will probably change to blue).

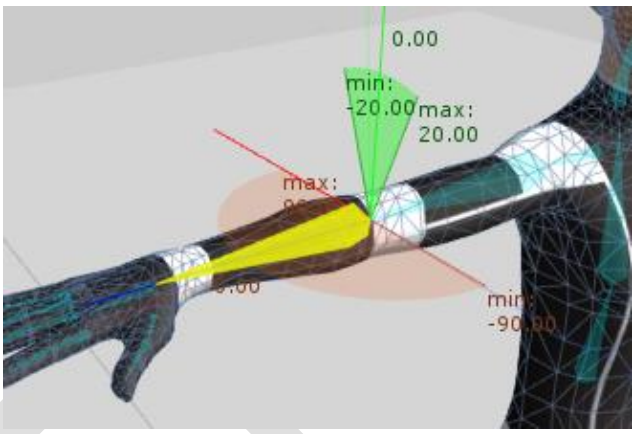
You can see the min value, max value, and current value (at the top).

If you do not allow twisting, these helpers will not show.

Hinge Swing and Twist

This joint allows the swing to happen only around one axis. This creates a plane on which the bone can rotate.

Think of your arm. It uses the shoulder for nearly free rotation. However, your elbow only allows your forearm to bend forward and then back.



If you were to stretch your right arm out (like in the picture above), the hinge or swing axis would come out of your elbow and go up. So, the first thing we define is this axis.

Swing Axis:
Axis the bone can rotate around. It's based on the original bind pose of the bone. Note that you can't swing around the z-axis (or bone forward) since that's what you twist around. So, you can't set a value in the z component.

Prevent Swing Twisting:
When checked, we'll attempt to remove any twisting that occurs due to the swing itself. Remember we talked about this a while back?

When checked, if you're swinging the joint in the editor, you may get some jumpy behavior. This is the joint removing the twist from the swing as you're swinging.

Limit Swing:
Determines if we'll force the bone to only swing between the min and max limits. These limits are in degrees and depend on the swing axis.

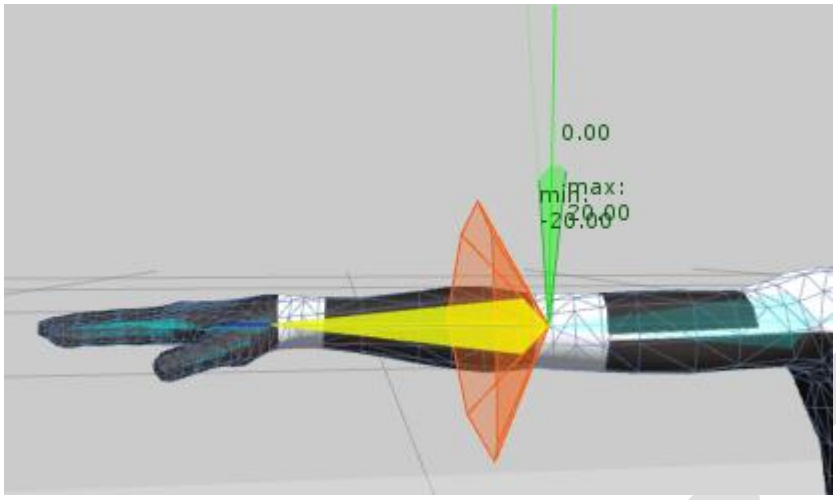
Allow Twist:
Determines if we will allow a non-swing-based twist to occur.

Limit Twist:
Determines if we'll limit the range of the twisting. If we do, you can set the range here. Note that the twisting is always relative to the bone's up axis (the green line).

Limited Swing and Twist

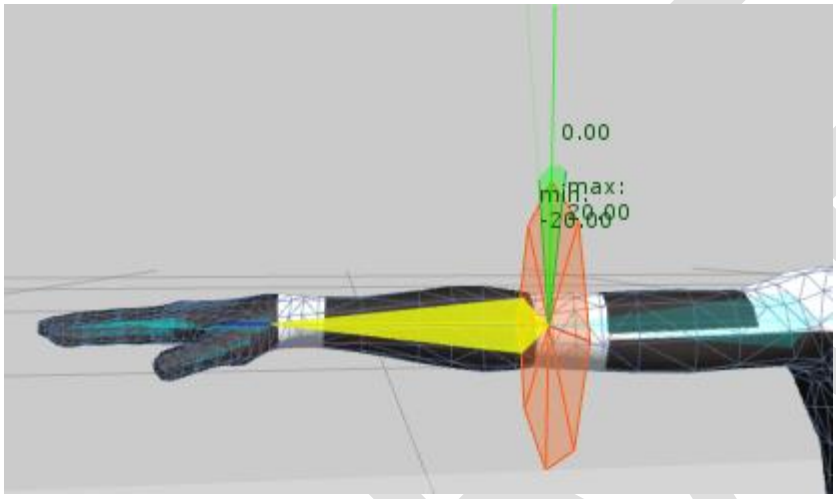
Hold on folks! Things are about to get bumpy…

The limited swing uses reach-cones for joint limits. This is based on the paper from Jane Wilhelm and Alan Van Gelder. No, you don't need to read it. However, it pretty interesting.

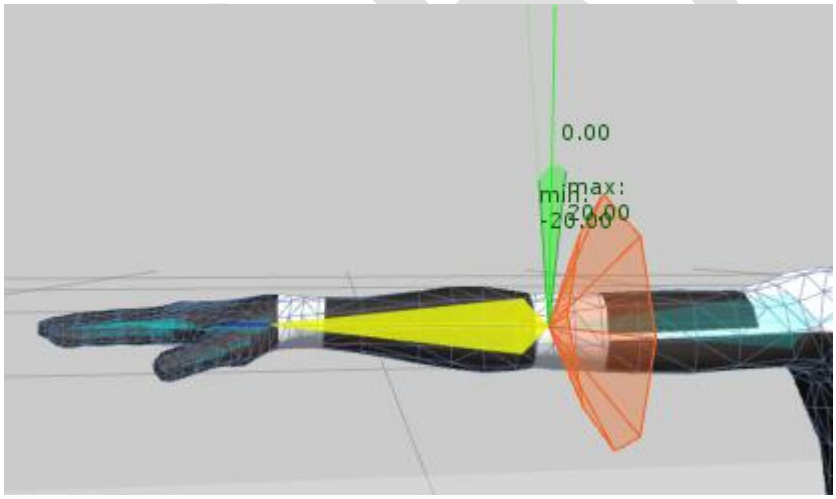


The orange area to the left represents the valid area that the bone can rotate in. If you tried to swing the bone outside of the orange code, it would stop.

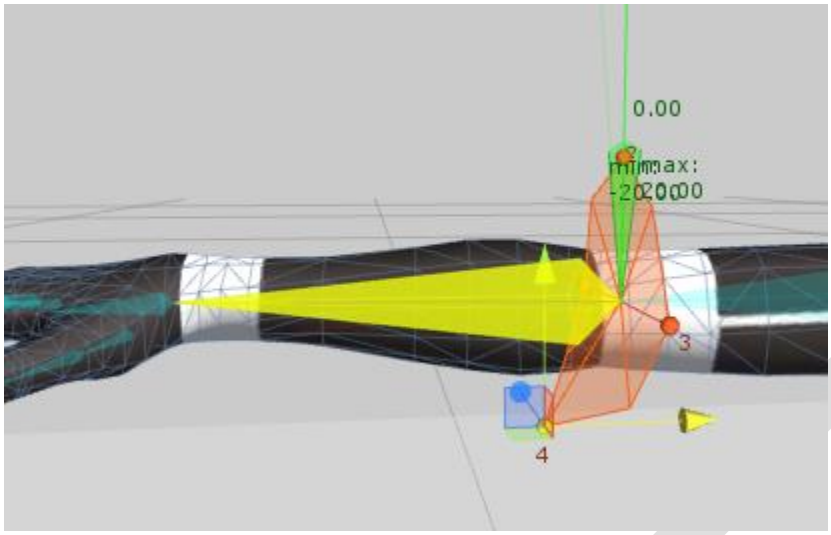
The thing is, the orange area doesn't have to be a cone…



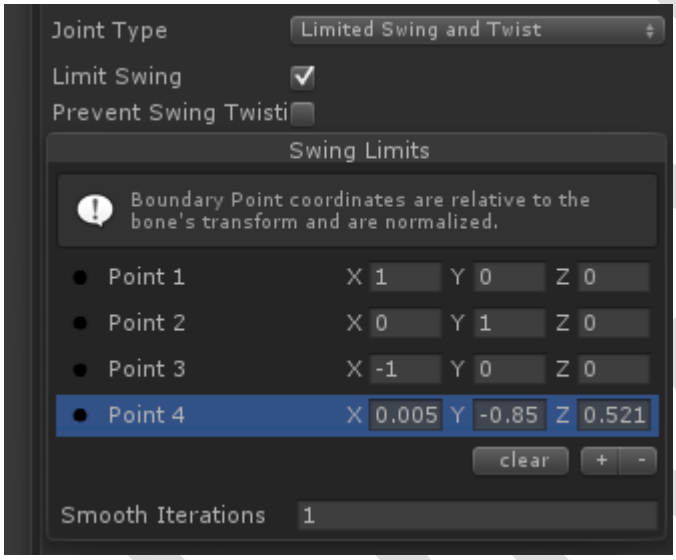
Here, the valid range doesn't go beyond a 90 degree pitch (up/down) or 90 degree yaw (right/left).



In this setup, the bone can rotate beyond 90 degrees.

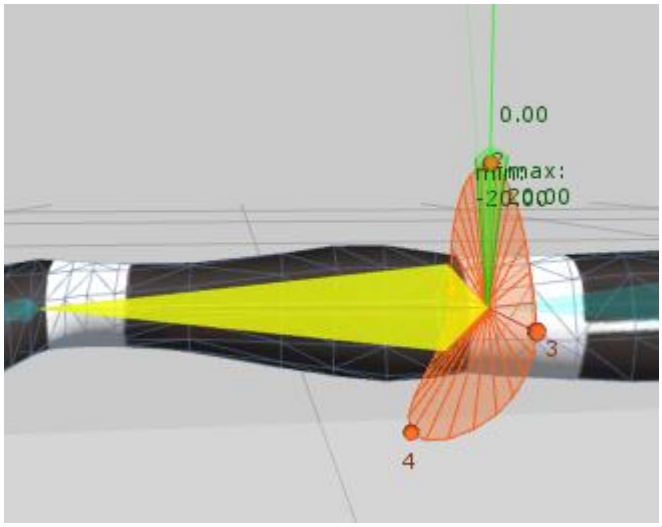


Using “Boundary Points”, you can change the shape of the area the bone can move in. You can add, remove, and move the boundary points as needed.



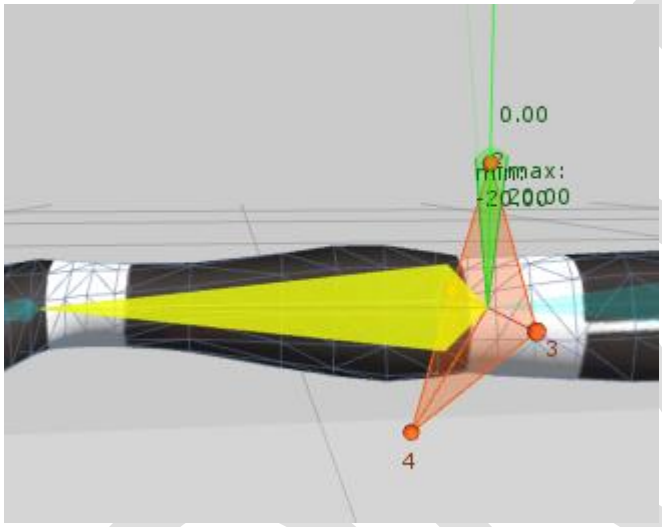
In the inspector, you can see the list of boundary points. These points are relative to the… you guessed it… bone forward.

What you see to the left matches what we’re doing in the picture above. The 4th point is being moved.

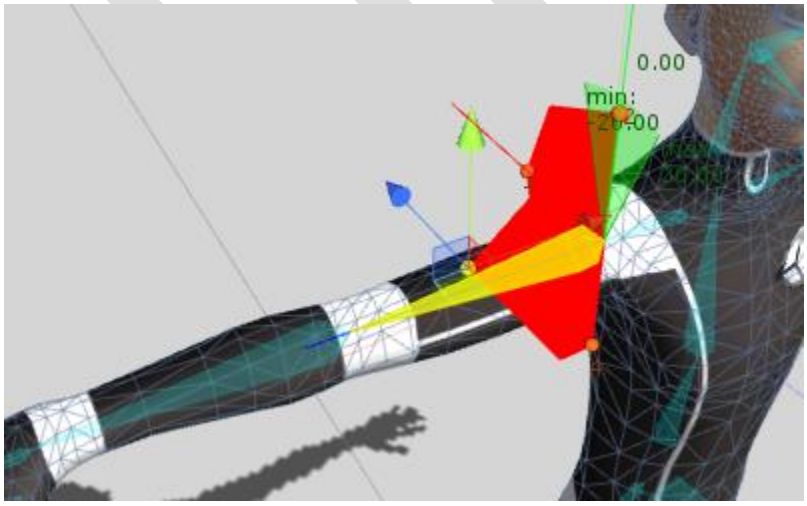


The **Smooth Iterations** determines how much we interpolate additional points in order to smooth out the shape. This is nice for smoother rotations along the boundary edge.

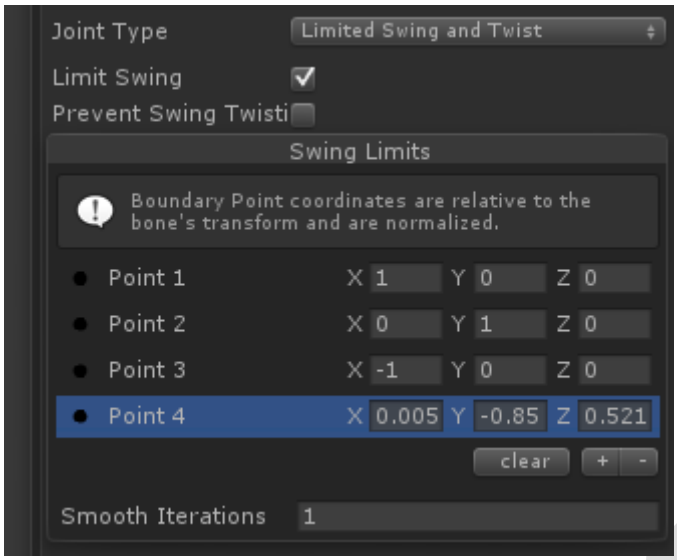
The picture to the left has a Smoothing Iteration value of 3. In order to protect system resources, you can only use a smoothing value of 0, 1, 2, or 3.



This is the same setup with a Smoothing Iteration value of 0.



If you move the boundary points into an invalid position, the entire structure will turn red. Doing this invalidates the joint limit and it will be ignored.



To reset the boundary points, simply press the "**clear**" button.

As a standard, press '+' to add a new point and '-' to remove the selected point.

Points need to be set in clockwise order (if you're looking from the top of the bone to the start).

If you want to hide the bind point editor handles in the scene, just click the bone selector "dot" at the top of the section.

We jumped over some settings…

Limit Swing:
Determines if we're using and processing the limit

Prevent Swing Twisting:
When checked, we'll attempt to remove any twisting that occurs due to the swing itself. Remember we talked about this a while back?

When checked, if you're swinging the joint in the editor, you may get some jumpy behavior. This is the joint removing the twist from the swing as you're swinging.

Allow Twist:
Determines if we will allow a non-swing-based twist to occur.

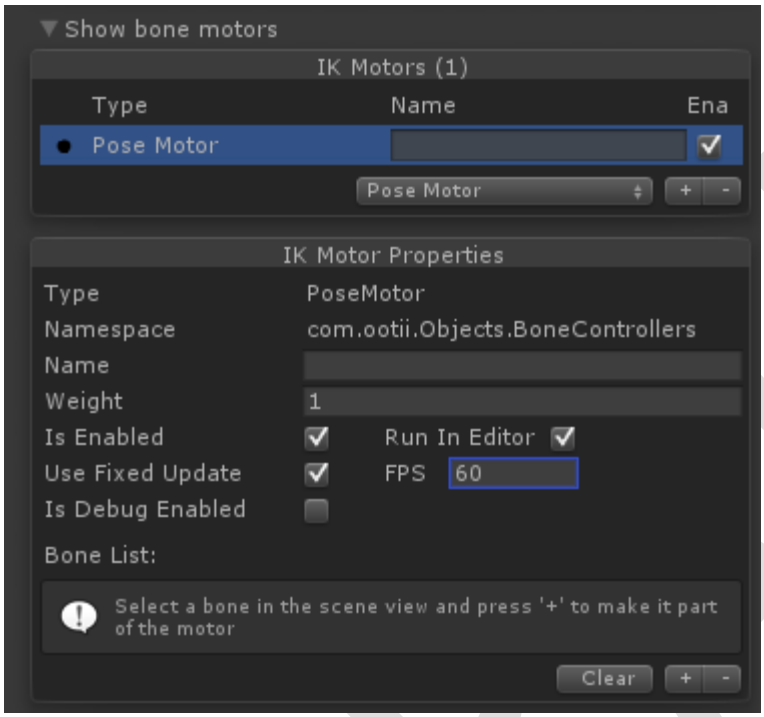
Limit Twist:
Determines if we'll limit the range of the twisting. If we do, you can set the range here. Note that the twisting is always relative to the bone's up axis (the green line).

Motor Settings

You can use the stock motors that come with the Bone Controller or you can create your own. If you're creating your own, check out the Motor Builder's Guide.

If you've used the Motion Controller this setup will look somewhat familiar.

First will be a list of enabled motors and under that are the properties associated with the motor.



Adding a Motor: Click the drop down and select the motor you want to add. In this case, we'll choose the `Pose Motor`. Then press the '+' button.

Sometimes you'll add a motor once. Sometimes you'll add it multiple times. For example, the Foot Placement motor is added once for the left leg and once for the right leg.

Removing a Motor: Select the motor by clicking the dot to the left of the motor type. Then, press the '-' button.

When deleting a motor, there is no 'undo'. Ensure you're deleting the motor you intend.

The motor itself has some important properties. You'll actually see these properties on pretty much every motor…

Name:
Allows you to give a custom name for the motor. This is important so that you can retrieve the motor later through code.

Weight:
This weight effects all bones in the motor. It helps to determine how much of the final bones' rotations (from this motor) will come from this motor vs. the original animations.

0 = All data is from the original animation
1 = All rotation data is derived from this motor

Is Enabled:
Used to determine if the skeleton will even run this motor.

Run In Editor:
We can actually run the motors in the editor. This is great for posing objects in the scene and just seeing how things work. Since there are no animations running, there really won't be any blending with the weight, but it's a good way to start testing the motor.

Use Fixed Update:
Some motors will apply gravity, physics, or lerp values. In these cases, we don't really want the bones to update as fast as possible. Instead, we want them to update on a consistent interval. This is very similar to Unity's "Update" vs. "FixedUpdate" functions.

If you are requiring a steady update, check this box.

FPS:
If you checked the Fixed Update box, this is the frame rate the system will attempt to stick to. It won't be 100% perfect, but it will be close. Similar to Unity's approach.

Is Debug Enabled:
Some motors contain extra debugging information that they will show during editing. Check this box to show that information (if it exists).

Bone List:
The Bone List contains all the bones that are being effected by the motor. Motors may allow you to pick your own bones, some will auto select bones, and some won't show this area at all. It's up to the motor builder.

Bind Pose Motor

A very simple motor that forces the character into the bind pose. This is really just for testing.

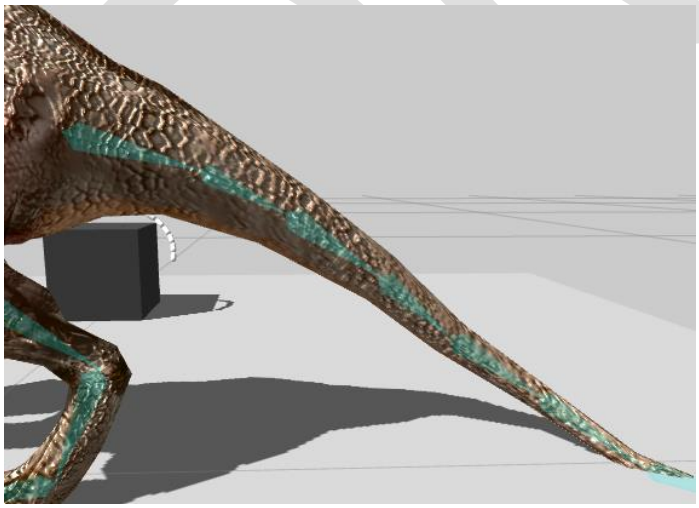
Bone Chain Drag Motor

This is a great motor that is used to move a chain of bones as if they were a rope or chain. Using the gravity and stiffness properties, you can create movement for tails, pony tails, vines, and other swinging structures.

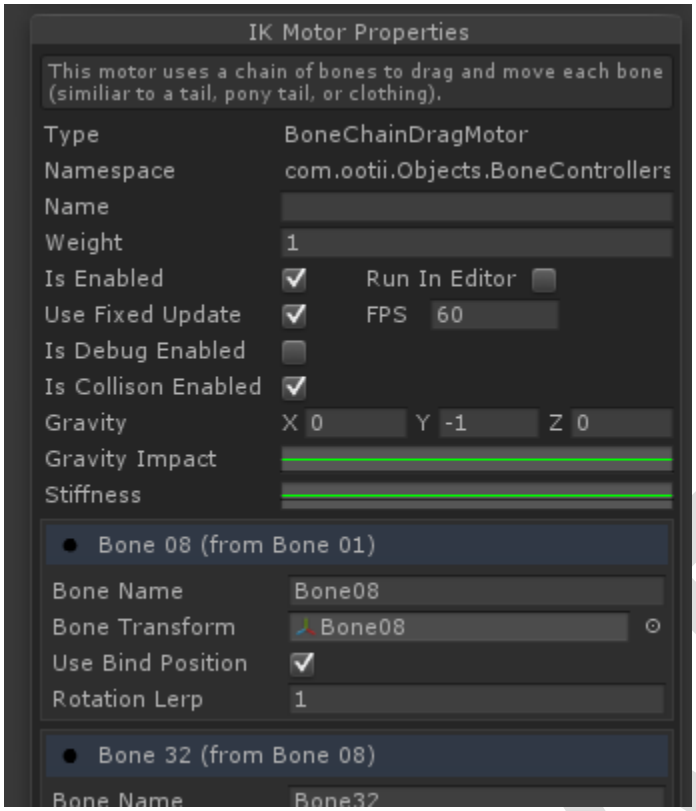
The motor also supports a very basic “surface collision” ability. If the bones are drug over an object that has a collider (like the floor, a table, etc.), the bones usually won’t fall through the surface. I say usually because they can be pulled through it and the ends may go through.

I took this approach for performance reasons. So, the chain won’t react if something pushes into it, but it will react when dragged over an object.

In this case, we took some capsules, turned them into bones, and then used the motor to drag them.



Here, we took bones that were part of the original skeleton and applied the drag motor. Now the dinosaurs tail moves more naturally and responds to movement.



Is Collision Enabled:
Determines if we enable the simple "surface collision" that I mentioned in the beginning.

Gravity:
The direction of gravity to apply. If the value is set to 0, no gravity will be used and the chain will look like it's floating.

Gravity Impact:
This curve represents the entire length of the chain. Each bone will get the value at its relative position in the curve and use this to modify the gravity that impacts it. This way, we can do some cool effects (like later chains down).

Stiffness:
This curve represents the entire length of the chain. Each bone will get the value at its relative position in the curve and use it as it's 'weight'. Meaning the value is used to determine if the original bind pose (or animation) determines it's rotation or the motor.

Bone List:
The Bone Chain Drag Motor contains a the list of bones that it manages. By selecting a bone in the skeleton and pressing the '+' button, that bone and all its children are added to the list. These bones represent the full chain.

As with the other motors, this list isn't truly the bones. Instead, it's a list of wrappers that each hold a bone. This wrapper approach works so we can add extra details that are important for this specific motor.

For each bone/wrapper, the following properties exist:

Bone Name:
Search field for the bone transform. You can use this to change the bone transform by searching for a new transform to tie the bone to.

Bone Transform:
The transform this bone/wrapper is tied to

Use Bind Position:
Determines if we use the animated position as the basis for dragging the bone or the original bind position. For bones we create, typically they aren't animated and we use the bind position. Even with "natural" bones, I'd keep this checked.

Rotation Lerp:
As the bone is drug from position to position, it determines how quickly it gets there. Use a value from 0 to 1 where one means "move instantly".

Finger Pose Motor

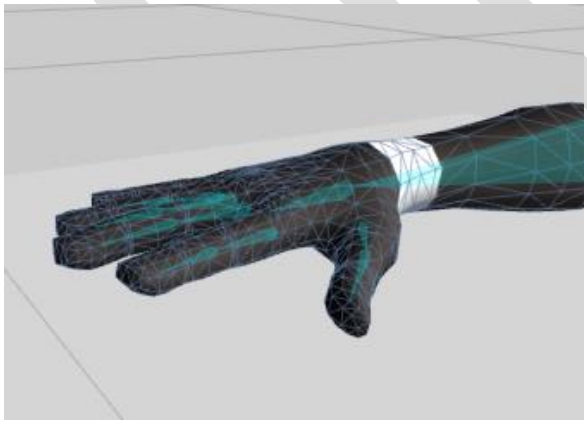
The whole purpose of this motor is to pose the fingers into specific positions. This can be important because a lot of animations tend to ignore the hands. Take for example, a character holding a pistol vs. a sword vs. a thick-handled-axe. In all these case, positioning the fingers correctly matters.

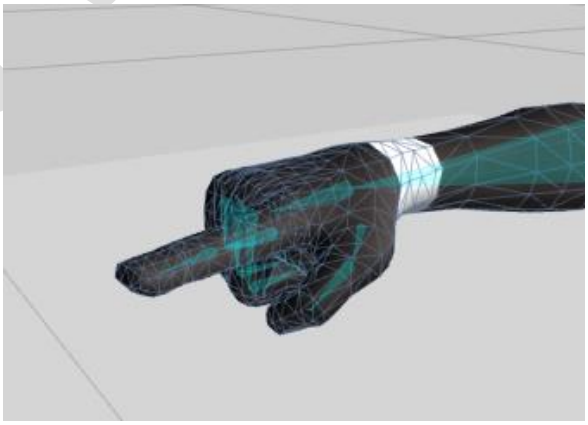
This motor is built for characters imported as a Mecanim “Humanoid” rigs. By doing this, you don’t have to assign the finger bones manually.

The motor is also useful for cut scenes if your character is trying to point out directions or use his hands.

Each of the sliders to the left basically extend or curl the specific hand/finger.

When the slider is to the left, the fingers are extended to the bind pose. When the slider is to the right, the fingers are curled into a fist.





Foot Ground (2 Bone) Motor

I expect this is going to be a pretty well used motor.

The purpose of this motor is to ensure the feet are correctly position on the ground even if the ground is uneven. The current version expects a 2-bone setup; thigh and calf. It then uses IK to bend the knee to the proper position while ensuring the foot matches the tilt of the ground.

In the left most picture, the bone motor is enabled. So, the character's feet are correctly positioned on the platform. The knees are also bent based on the original hip position and the new foot position.

On the right, the motor has been disabled. Here, the character would simply push through the platform.

Use Bind Rotation:
When calculating the base pose for rotating the legs, it determines if we use the bind rotation or the current rotation. Typically you'd leave this unchecked.

Use Plane Normal:
Determines if the bend axis is based on the bone's bend axis (as set in the bone list) or by the plane normal that is created by the three points (hip, knee, and foot). Typically you'd leave this unchecked.

Apply Joint Limits:
Gives this motor the ability to ignore joint limits and bend as needed.

Rotate Foot to Ground:
Determines if we'll attempt to rotate the foot to match the ground's tilt.

Rotate On Movement:
Determines if the foot will be rotated to match the ground while the character is moving. This helps to prevent the character from running "flat footed".

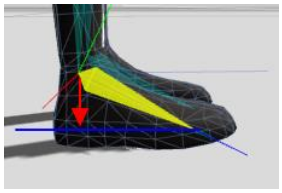
The "flat footed" behavior can happen because people tend to run on their toes, but the motor wants to have the foot match the ground surface angle.

Requires Both Foot and Toe:
There are actually two rays cast out of the foot. One from the heel and one from the toes. If this is checked (and Rotate Foot to Ground is checked), we'll only rotate the foot if both raycasts result in a collision. This is useful so we don't lock the foot down when a walk cycle just has the toes on the ground.

Allow Extension:
Typically, the foot would be pushed up. However, we also may want the foot to reach out and attempt to place itself on the floor. This field along with the Extension Distance will determine if we do that.

Extension Distance:
If the 'Allow Extension' option is checked, this is the distance from the heel that the ray will extend in order to find the floor and stretch the foot out to reach in.

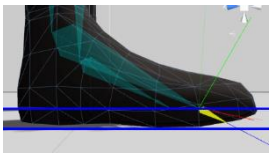
Foot Toe Distance:



In order to help position the foot correctly, we need to understand the vertical between the foot bone and the toe bone.

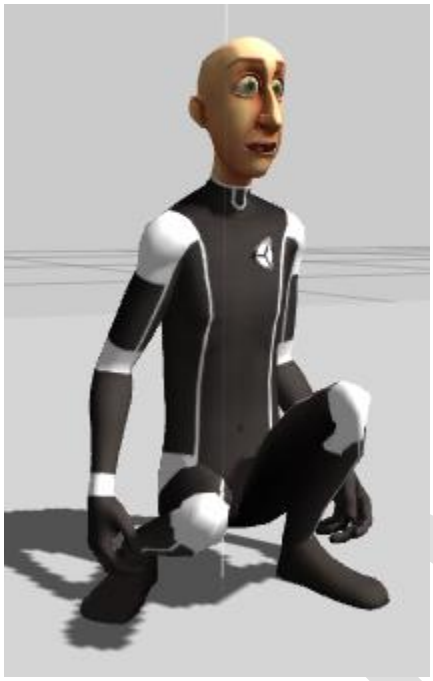
If you leave the value as 0, I'll actually figure out the value on my own.

Toe Sole Distance:



Some models have spacing between the toe bone and the actual sole of the foot. Adding a distance here will make sure the foot is property spaced from the ground.

Start Distance:
Distance from the foot bone that the rays for collision detection will start. Typically this would be the middle of your character. So, if the character is 1.8 meters tall than a value from 0.5 to 0.9 is good.



The floor can go up as high as the Start Distance. However, if the start distance is too high (ie up past the hips), you could get some uncomfortable looking results.

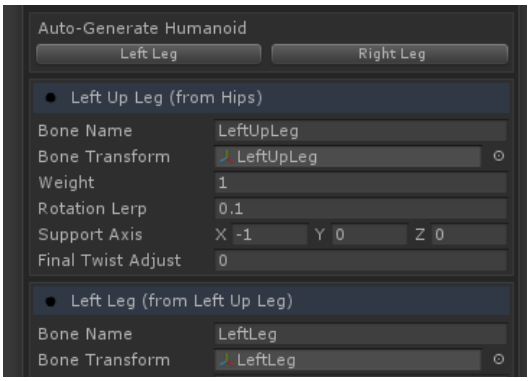
If the floor is higher than the Start Distance, the feet will ignore any collision and go back to their animated pose.

Bone List:
This motor contains a list of bones that represent the leg structure. There are typically four bones (in this order):

- Upper Leg (required)
- Lower Leg (required)
- Foot (required)
- Toe (optional)

If you're using a character with a Mecanim humanoid rig, you can auto generate the settings by pressing one of the two buttons below. These will set the typical values you'd use.

Since every character is different, the default values may still need adjusting.



Bone Name:

Search field for the bone transform. You can use this to change the bone transform by searching for a new transform to tie the bone to.

Bone Transform:
The transform this bone/wrapper is tied to

Weight:
Value from 0 to 1 that determines how much of the motor's rotation is applied to the final rotation result.

Rotation Lerp:
As the bone moves from frame to frame, it determines how quickly it gets there. Use a value from 0 to 1 where 1 means "move instantly".

Support Axis:
Specifically for the upper and lower leg bones (bone #1 and #2), the support axis is used to determine how the bones will naturally bend.

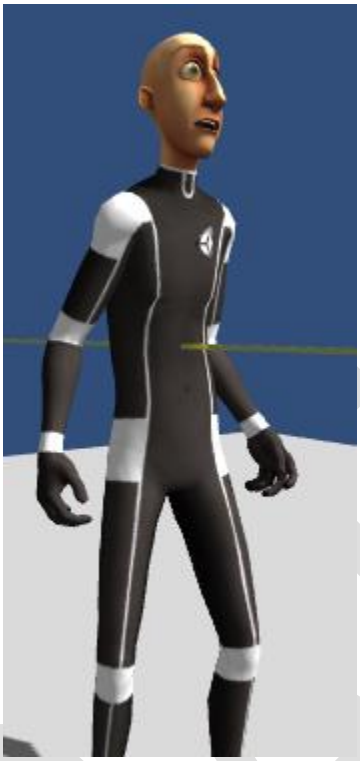
For the second bone, it acts similar to a hinge joint and forces the lower leg to rotate on that axis. Flipping the sign of the axis will flip the direction that the bone bends.

As always, the value is relative to the bone's forward direction.

Final Twist Adjust:
For some systems, we may need to force some twist into the bone to make it look more natural. You can do that here.

Impact Motor

The impact motor is used to bend bones temporarily to simulate the impact of an arrow, bullet, or even sword.



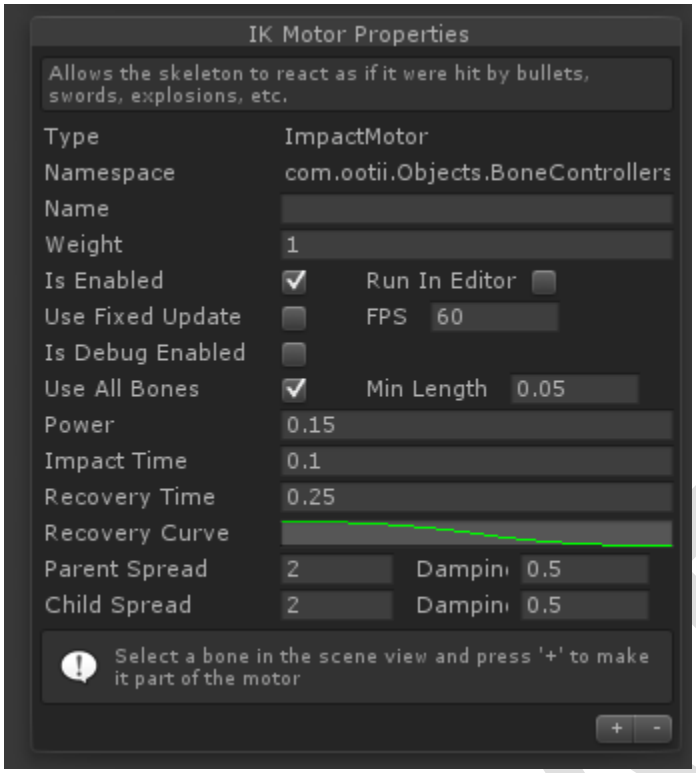
It may be a little hard to see in these static images, but the ray is shooting into our character's belly. That impact is causing him to bend forward slightly and then for his head to go back.

The "impact" happens within a specific time and then the "recovery" happens within a specific time. Typically both phases occur within a fraction of a second.

The impact is able to span a length of bones. This way the result is more natural.

The impact occurs because a "`RaycastImpact()`" function is called through script. This function will shoot a ray based on the parameters and if it collides with a bone collider the impact occurs.

Note that this impact motor only works with pseudo-colliders.



Use All Bones:
Instead of limiting the bones that will respond to the `RaycastImpact()` call, you can use this check box to allow all bones to be tested.

Min Length:
If you do allow all bones to be impacted, you can specify a minimum length for the bone. This is useful for ignoring small bones…like fingers.

Power:
When the `RaycastImpact()` function is called, the velocity of the ray is used to determine how much power it as. This property can temper the power of all calls. It's a multiplier that is applied to the call.

Impact Time:
Time (in seconds) for the "impact phase" of the motor to move each impacted bone into the impacted position.

Recovery Time:
Time (in seconds) for the "recovery phase" of the motor to move each impacted bone back to its normal position.

Recovery Curve:
This curve represents the length of the full impact chain and modifies the recovery time per bone. It allows the recovery time to be spread out from the first impacted bone out to the children. So, in the picture above the values go from 1 down to 0. That means the first bone's recovery time would be (Recovery Time * 1.0). The last bone in the chain would have a recovery time of (Recovery Time * 0.0)…meaning it instantly recovers.

Parent Spread:
Determines how many "generations" the impact will spread along the parent bones. This creates a more natural look as an impact to the upper arm would affect the rotation of the upper arm… and shoulder… and chest.

Damping (Parent):
Each generation we spread the impact, we reduce the impact by this amount. This way the shoulder would feel less impact than the upper arm and the chest would feel lest impact than the shoulder.

Child Spread:
Determines how many "generations" the impact will spread along the child bones. This creates a more natural look as an impact to the upper arm would affect the rotation of the upper arm… and forearm… and possibly the hand. The impact for children is actually inverted. This creates almost a curl around the impacted area.

Damping (Child):

Each generation we spread the impact, we reduce the impact by this amount. This way the forearm would feel less impact than the upper arm and the hand would feel lest impact than the forearm.

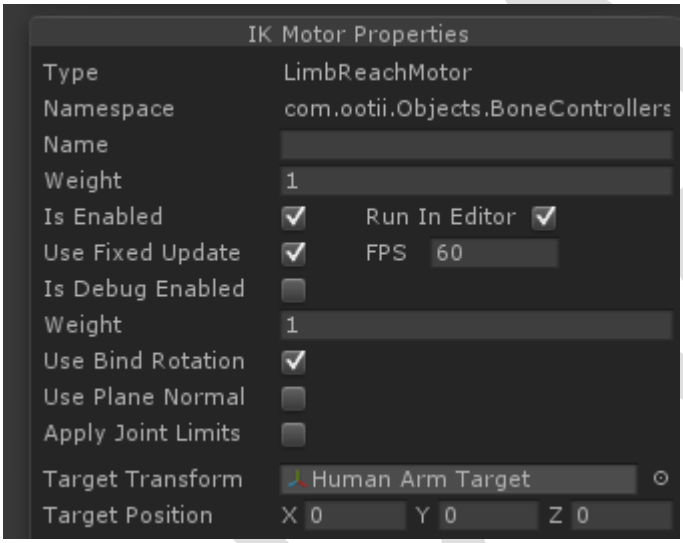
Limb Reach Motor

The Limb Reach Motor is very similar to the Foot Placement Motor. In fact, they both use the same cosine solver to bend the 2-bone system.

That brings up a good point… this motor expects two bones: upper bone (ie leg or arm) and a lower bone (ie leg or arm).

In this picture, the ball represents a target object and the 2-bone system includes the right upper arm and the right lower arm.

As the ball moves, the arm moves and bends naturally to reach its position.



Use Bind Rotation:
When calculating the base pose for rotating the legs, it determines if we use the bind rotation or the current rotation. Typically you'd leave this unchecked.

Use Plane Normal:
Determines if the bend axis is based on the bone's bend axis (as set in the bone list) or by the plane normal that is created by the three points (hip, knee, and foot). Typically you'd leave this unchecked.

Apply Joint Limits:
Gives this motor the ability to ignore joint limits and bend as needed.

Target Transform:
Object whose position will be used as the target to reach for.

Target Position:
Fixed position that will be reached for. If a transform is set, that will take precedence.

Bone List:
This motor contains a list of bones that represent the structure. There can only be 2 bones for this motor.

If you're using a character with a Mecanim humanoid rig, you can auto generate the settings by pressing one of the buttons below. These will set the typical values you'd use.

Since every character is different, the default values may still need adjusting.

Bone Name:
Search field for the bone transform. You can use this to change the bone transform by searching for a new transform to tie the bone to.

Bone Transform:
The transform this bone/wrapper is tied to

Weight:
Value from 0 to 1 that determines how much of the motor's rotation is applied to the final rotation result.

Rotation Lerp:
As the bone moves from frame to frame, it determines how quickly it gets there. Use a value from 0 to 1 where 1 means "move instantly".

Support Axis:
The support axis is used to determine how the bones will naturally bend.

For the second bone, it acts similar to a hinge joint and forces the lower leg to rotate on that axis. Flipping the sign of the axis will flip the direction that the bone bends.

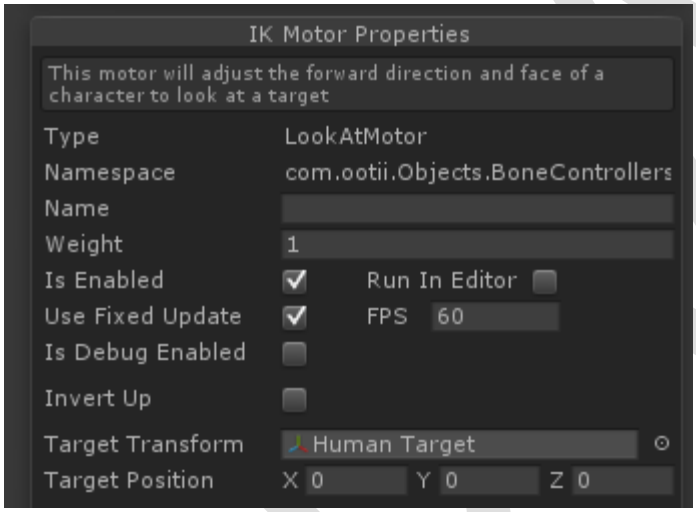
As always, the value is relative to the bone's forward direction.

Final Twist Adjust:
For some systems, we may need to force some twist into the bone to make it look more natural. You can do that here.

Look At Motor

The Look At motor is used to rotate a chain of bones that typically represents the head, neck, and spine of the character. It bends those bones at a decreasing rate to simulate how the human body rotates to look at an object.

In most cases, the bone chain just works out-of-the-box. However, with some models (like the Allosaurus) each bone needs to have an offset.



Invert Up:
To help with the rotations of the bones, you can invert the "up" vector of the bone. Typically this isn't needed.

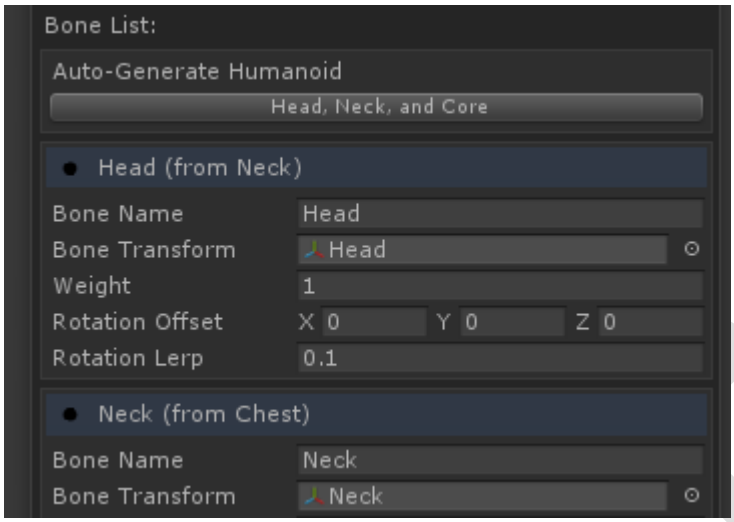
Target Transform:
Object whose position will be used as the target to reach for.

Target Position:
Fixed position that will be reached for. If a transform is set, that will take precedence.

Bone List:
This motor contains a list of bones that represent the chain to be rotated. Typically you start with the head and then move down the chain… neck, chest, spine, etc. Add as many bones as you feel it takes to make the look natural.

If you're using a character with a Mecanim humanoid rig, you can auto generate the settings by pressing the button below. These will set the typical values you'd use.

Since every character is different, the default values may still need adjusting.



Bone Name:
Search field for the bone transform. You can use this to change the bone transform by searching for a new transform to tie the bone to.

Bone Transform:
The transform this bone/wrapper is tied to

Weight:
Value from 0 to 1 that determines how much of the motor's rotation is applied to the final rotation result.

Rotation Offset:
Since the bone forward of each bone can be pointing in different directions, this allows us to add some adjustment.

I found this especially useful for tilting the head in order to make sure the eyes are actually looking at the target.

This was also an important feature for the Allosaurus. See below…

Rotation Lerp:
As the bone moves from frame to frame, it determines how quickly it gets there. Use a value from 0 to 1 where 1 means "move instantly".

Because we're trying to support any character…including non-humans, some models need a little extra love to help the rotations look natural. This was the case of the Allosaurus.

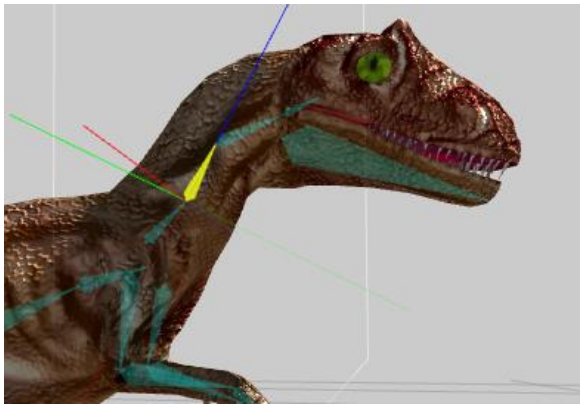
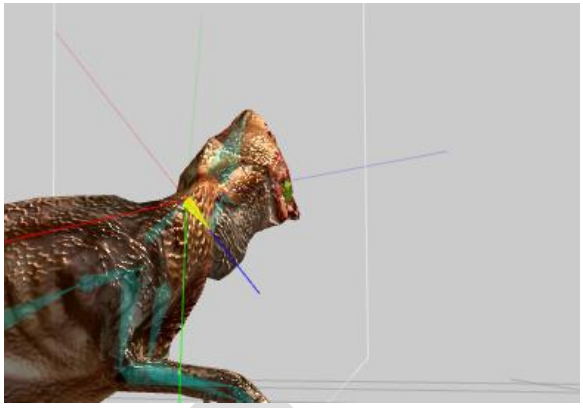
Allosaurus Adjustments

First, special thanks to Tibor Szijjarto for letting me use his awesome Allosaurus!

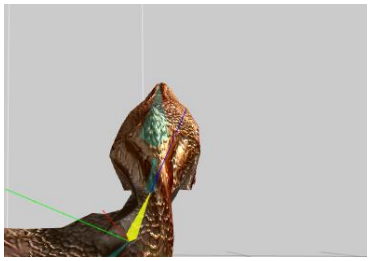
Because the allosaurus' shape is elongated and the head-to-spine structure isn't vertical, the **Rotation Offsets** needed to be adjusted per bone.

This basically means that I had to unwind the bones.

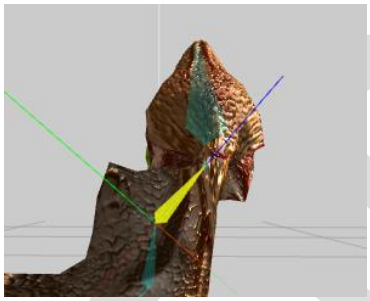
Below is a comparison of the original bind rotations of the allosaurus vs. what happens when the Look At motor is applied (before Rotation Offsets are set)

Again, this is because the natural spine of the dinosaur isn't vertical. But, it's really not that big of a deal. We can clean this up.

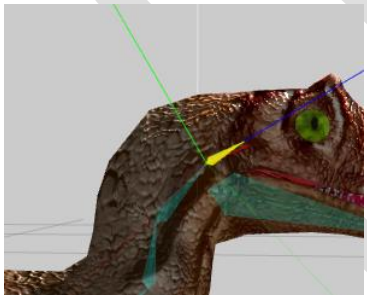


Starting with the base bone (spine), I rotated each axis from the inspector until the bone was roughly in the same position as the original bind pose.



Then I move to the next bone and do the same thing. I just work my way up the chain towards the head. As you unwind the bones, make sure you look from all angles.

I also modify the z-axis (twist) last. It just makes it easier.



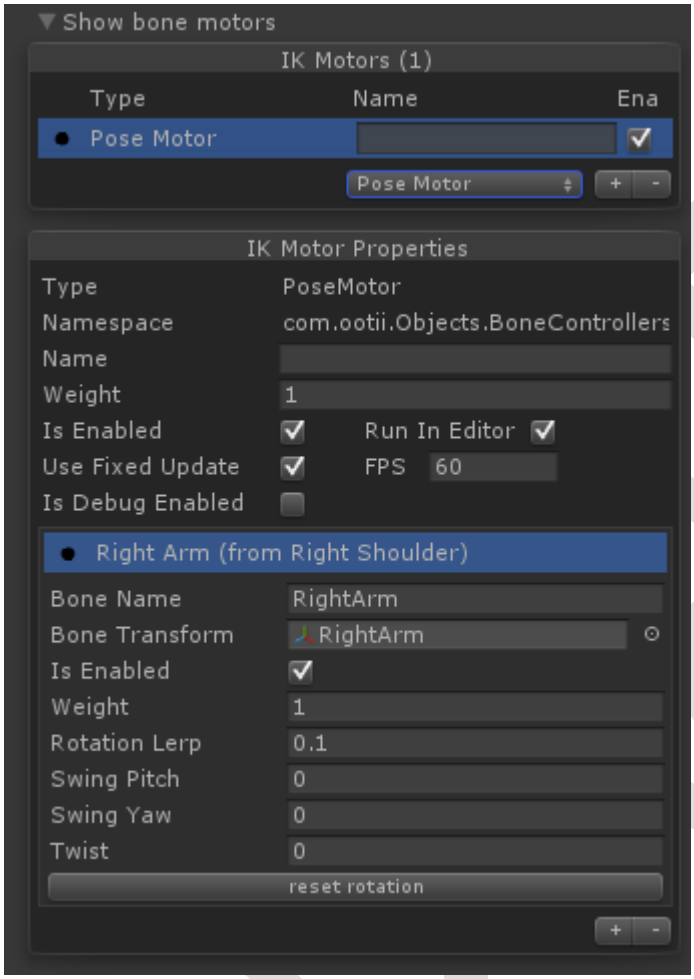
Altogether, the allosaurus took me about 3 minutes to unwind the bones. At one point, I did zero the rotations out and start over. It just takes a little playing with it.

I'll work to try to automate this, but for now the unwinding isn't tough and it depends on how the character was created.

Pose Motor

The Pose Motor allows you to pose individual bones and have those poses override or blend with animations. Most important to the Pose Motor is the list of bones that it manages. We don't want to waste time processing every bone, so only ones that are part of the motor will be posed.

You can see that the Right Arm bone was added to the list. The list isn't actually a list of bones, but more a list of bone wrappers. These wrappers give us extra information to apply to the bones themselves.



In the case of the Pose Motor, the extra information are things like swing and twist.

Bone Name:
Name of the bone that is used for searching. This doesn't rename the bone, but searches for a transform to choose.

Bone Transform:
The transform representing the bone we are wrapping.

Is Enabled:
Determines if this motor will control the bone.

Weight:
When the motor processes, it determines how much we use the original animation's rotation vs. this motor's rotation to control the final bone rotation.

0 = All data is from the original animation
1 = All rotation data is derived from this motor

Rotation Lerp:
Determines how quickly the bone will move to the final rotation. This is a great way to make things look more realistic as the bone can slowly rotate to position vs. just popping there.

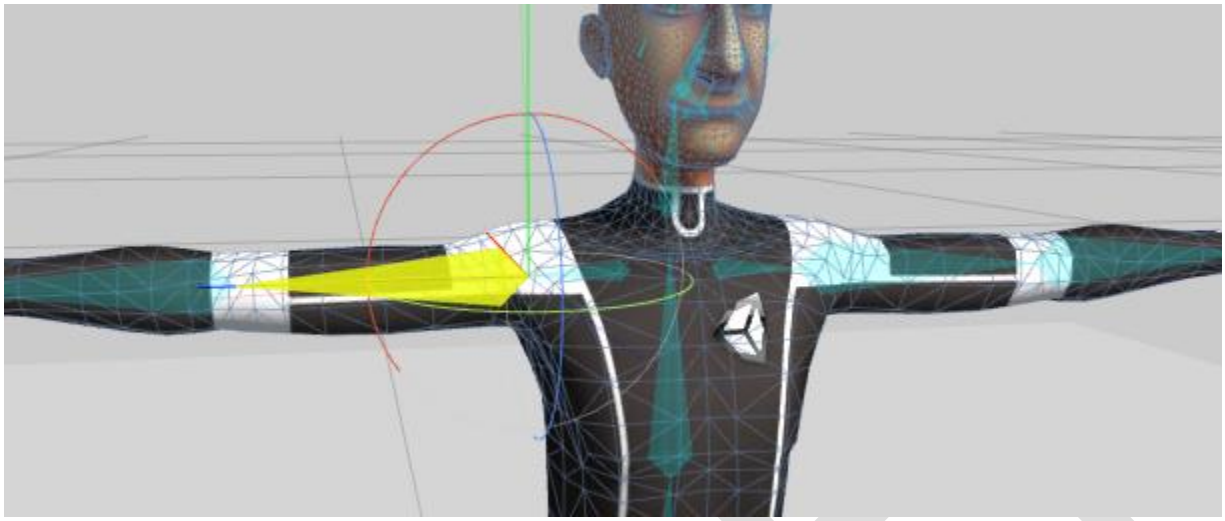
Swing Pitch:
Remember we talked about swing vs twist? This controls the rotation of the bone around the x-axis. Typically that's the up/down rotation.

Swing Yaw:
This controls the rotation of the bone around the y-axis. Typically that's the right/left rotation.

Twist:
This controls the rotation of the bone around the z-axis. Remember the z-axis is the bone's "forward direction". so it's the spinning of the bone along its length.

Manipulating Bones

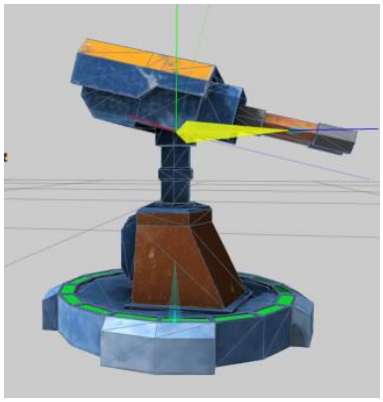
When we added the bone, you probably noticed that the scene view updated to include GUI manipulators. You can use these or the Swing Pitch, Swing Yaw, and Twist fields to set the pose you want.



Rotation Motor

The Rotation motor is a little more generic, but has a lot of potential uses. This motor simply rotates a bone over time. The rotation can be based off of the bone forward or the original model forward.

This is a great motor for animating static objects in the scene. Things like turrets, gears, or even body parts. I use it to automatically rotate the camera too. It will respect limits just like any other motor.

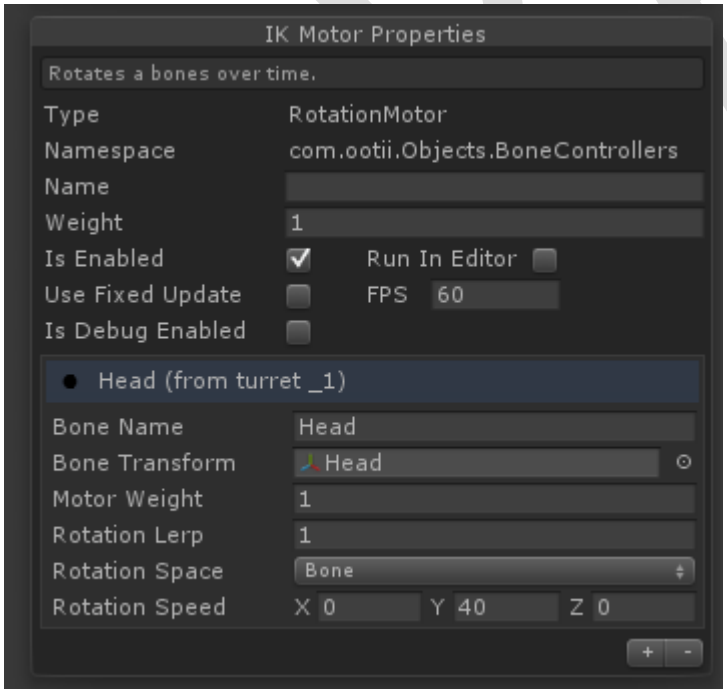


This is an example of a turret that came in with no bones. It's just a couple of static meshes put together.

With the Bone controller, we were able to add "pseudo-bones" that we can then use. I say "pseudo-bones" because Unity doesn't really know about the bones and there won't be any animations for them…since they don't exist.

However, from the Bone Controller's perspective they are just bones and can be used with any motor.

Bone List:
This motor contains a list of bones that represent the bones to rotate. Each bone will act independently, but control its children (as we'd expect).



Bone Name:
Name of the bone that is used for searching. This doesn't rename the bone, but searches for a transform to choose.

Bone Transform:
The transform representing the bone we are wrapping.

Weight:
When the motor processes, it determines how much we use the original animation's rotation vs. this motor's rotation to control the final bone rotation.

0 = All data is from the original animation
1 = All rotation data is derived from this motor

Rotation Lerp:
Determines how quickly the bone will move to the final rotation. This is a great way to make things look more realistic as the bone can slowly rotate to position vs. just popping there.

Rotation Space:
This drop down will determine if we're rotating based on the bone's original axes or our calculated "bone forward", "bone up", and "bone right" axes.

Rotation Speed:
The speed in "degrees per second" that we'll rotate. The rotation occurs around each axis as defined by the Rotation Space above.

Swing Motor

The Swing Motor is actually inherited from the Look At motor. In fact, the properties are exactly the same. So, I'm going to send you back there for the details. ☺

With the swing motor, you tend to want to give more weight to the lower bones (chest, spine, etc). This will result in the character's body facing the target and the swing aiming in the right direction.

So, why a separate motor?

1. It makes it easier to find when you want to enable/disable it.
2. Allows us to change the default values from the Look At motor.
3. Provides a foundation in case we want to customize it.

Bone Motor Builder's Guide

Check out the Bone Motor Builder's Guide at:

http://www.ootii.com/unity/BoneController/BCMotorBuildersGuide.pdf

Support

If you have any comments, questions, or issues, please don't hesitate to email me at support@ootii.com. I'll help any way I can.

Thanks!

Tim