BSoD/Introduction to Rigging

by Robert Christian (wavez)

Table of Contents

The Objective

Introduction to Character Rig Design

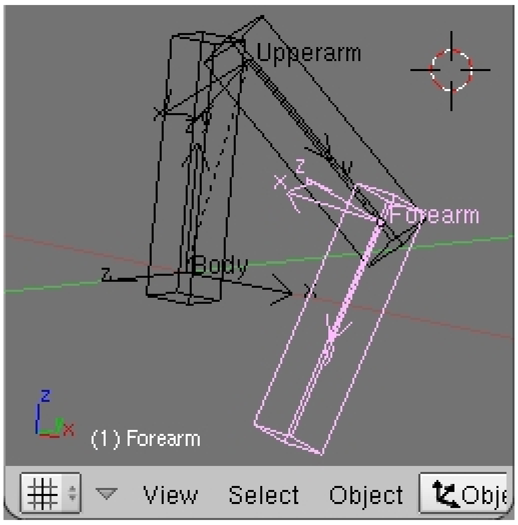
In this section I'm going to show how character animation evolved from its earliest beginnings, into what it is today. This will help give you an understanding of the universal concepts and techniques of character control that are used today in 3D digital character animation. Then we'll delve into the more specific details related to the implementation of these techniques in Blender.

The Early Days

"Money for Nothing"

The first 3D characters started appearing in film between 1980 and 1985. In 1984, the rock group Dire Straits released a music video for their song, "Money for Nothing", which featured a pair of low-detail polygonal characters.

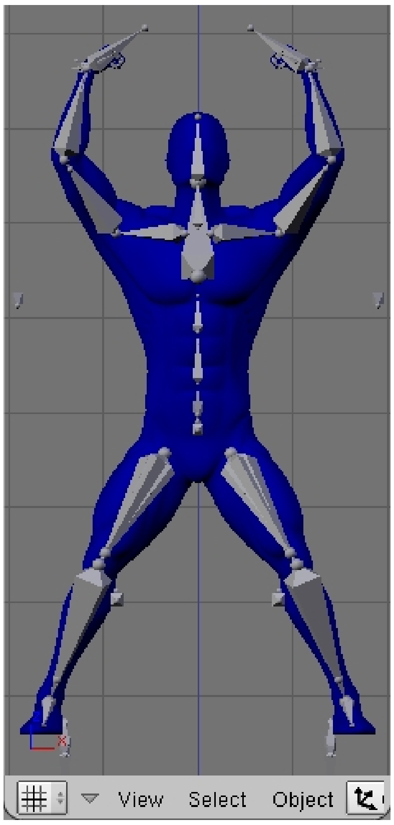
How to Make A Move



There was probably no such thing as a skeleton object in 3D applications back then, so how would someone animate a character if it has no skeleton? The solution was to use a series of multiple objects to approximate the shape of the character, and place them into a **hierarchy(link)**.

This created a fairly straight-forward animation technique, they simply animated the rotation of each of these different objects. You can try this for yourself by creating a hierarchy of objects and then rotating them one at a time. This approach became known as **Forward Kinematics**, or **FK**.

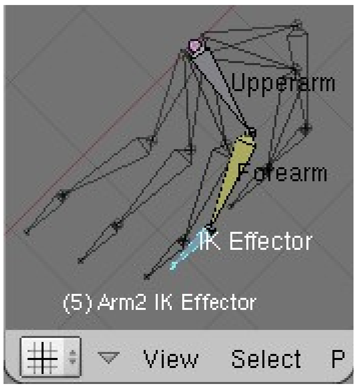
A Skeleton Was Born



FK is a good process for generating movement, but we can't keep using multiple objects for all the parts of our characters because it looks fake. If we want it to look like something organic, the whole character needs to consist of only one mesh. There's no breaks in the skin around a human elbow, and so it should be with the mesh of a human CG character. So the skeleton object was created. It's an object that can be rotated, translated, and scaled, just like most other types of 3D objects, but it's special because we can use it to move the vertices of a mesh object. In Blender, this object is called an **Armature**. Note that **hooks(link)** are also objects that can move the verts of mesh objects.

Back to Kinematics

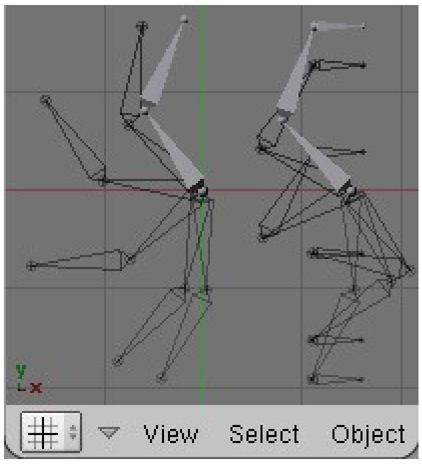
There was still another issue to deal with though. Consider two examples; an arm and a leg. An arm can be moving around in the air, but the leg needs to make contact with the ground. The foot needs to be completely motionless (relative to the ground, not the body) or any chance at the appearence of realistic movement is lost. If we want the character to move from standing to crouching, how do we keep the foot in one place while the knee joint and the hip joint are both rotating?



An IK effector in Motion

The solution to this problem was to create a new method that implements a recursive process which rotates the bones so that they reach toward a target. As long as the target is within range, the chain of objects will rotate as needed in order to touch the target. This process is called **Inverse Kinematics**, or **IK**.

When you work with FK, you rotate the character's joints yourself. When you work with IK, you move a target, and the bones are made to touch the target, so the rotations are all done for you. Sounds great, right? Who needs FK, right? Well, animators do, and the following image shows that IK and FK have very different behaviors.



FK vs IK

Both of these arms have two **key frames**(link here), but the one on the right is animated with IK, and the one on the left with FK. For the arm on the right, this means that the IK effector has a key frame starting position, and a key frame ending position. To Blender, this means the animator is saying "go from point A, to point B", so obviously the effector is going to travel in a strait line. But is that what we want? Maybe it is, or maybe it's not, it depends on the character. If this is the arm of an olympic swimmer doing laps in a pool, then we probably want a movement more like what we see with the arm on the left. But if this is the robotic arms of a CNC machine that is cutting a perfectly straight line into a block of metal, then we probably want something like the motion of the arm on the right. Often, however, we need both IK and FK on the same character, but at different moments. Maybe our olympic swimmer is going to swim over to the steps, walk out of the pool and open a sliding glass door, in which case his hand needs to sync up with the handle of the door as it moves. We'll cover IK and FK blending later though.

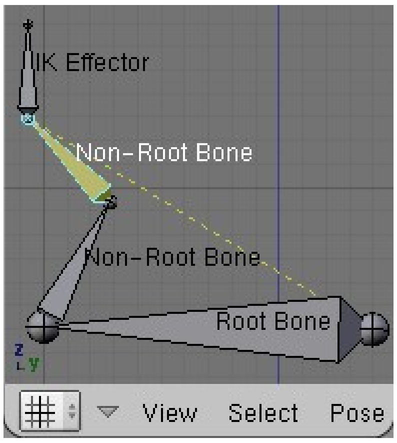
Summary

Hopefully the information we've covered so far has helped you understand how character animation evolved from what it was, into what it is now. Hopefully this has helped you see what the methods and objectives are, and how the FK and IK techniques fit into the bigger picture that is character animation.

As you progress further and further into the topic of character animation, you'll see the terms "animation" and "rigging" thrown around a bit, but you might not completely understand the meanings and differences of these two terms. "Animation" is a couple things. For one, it's the specific act of simulating movement. On the other hand, basicly everything in film, TV, and games, is done for the sole purpose of creating animation.

Often, when industry professionals are asked, "what do you do?", they will respond, "I'm a computer animator". This doesn't have to mean that they actually *do* the animating, but instead means that they work for a company that produces animations. In this regard, all subjects of computer graphics can be placed under the umbrella of computer animation.

This section talks specificly about the area in computer animation called "rigging". Rigging a character is like designing the controls for a puppet. There are puppets that dangle from strings, puppets that sit on hands, puppets that move with metal bars, and of course, all types of animatronic robot puppets. Even in computer graphics, we still need tools to help us create motion. This is the world of character rigging. Some of our tools are:



chain

A series of connected bones is called a "chain", and these are the terms we use to refer to the elements in a chain.

- Object hierarchies
- Axis locks
- Degrees of freedom
- Object constraints (and)
    - Other types of object relationships

Just like real puppets, digital puppets need joints, and so we have skeleton-like, digital objects. We arrange these objects into hierarchies so that when a character's neck moves, his head goes with it. Or when the upperarm moves, the forearm is not left floating by itself, nor does it need to move on it's own to keep up. Because we make it the child of the upperarm in a hierarchy, it simply stays attached. And this goes for the upperarm as the child of the shoulder, and shoulder as the child of the spine.

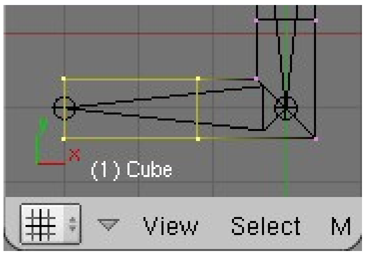
Sometimes we have an object that works as a control for parts of our character, but we want to limit the ways it can be manipulated. For these situations, we have axis locks. This allows us to make some bones unmoveable, unrotatable, or unscaleable. There are 3 axes for each type of transformation, making 9 axes total, and we can lock as many or as few of these as we like to suit our needs.

In some IK chains, we want to limit the range of motion for certain bones. One example would be the wrist of a person, it should only rotate 180º. In Blender we can do this using DoF's, **Degrees of Freedom**.

The single most used tool of a rig is the constraint. It allows us to define other types of relationships that hierarchies can't give us. Besides constraints, we also have a few other special tools, such as **IPO drivers(link)**, **pydrivers(link)**, and **python scripts(link)**.

The Tools

the Armature and its Features



In this image, the selected verts will belong to the bone they surround

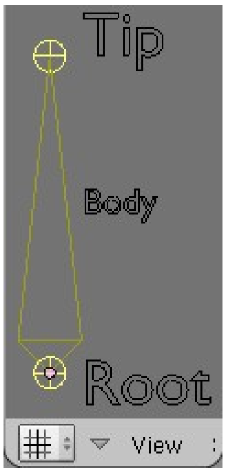
In Blender, skeletons are known as **Armature Objects**. Armatures are actually nothing like real-life skeletons. The best way to think of a bone in CG, is to imagine it as a center about which the verts of a mesh can exist as children.

The Blender armature object is a container for sub-objects, called bones. We're going to tour through all the features of the armature object, and you can try them as we go along. You can add an armature the same way you add any other object in Blender, through the **SPACE** menu. Any time you add an object to a scene, you should consider clearing it's rotation with **ALT+R** *while in object mode*.

the Armature and its Work Modes

Armatures have an object mode and an edit mode, just like the **Mesh Object(link)**, and you can toggle these with **TAB**, just like a mesh object. The state that you see your armature in while in edit mode is known as the **Rest Position**. Armatures are meant to be animated, so to fullfill this purpose, they have yet another mode: **Pose Mode**. You can toggle pose mode by pressing **CTRL+TAB**. Pose mode is different from edit mode in that you can change the selection to other objects outside the armature without first leaving pose mode. If you do change selection, when you return to select the armature, you will still be in pose mode. Bone selection is preserved when changing between edit and pose mode. You are placed into edit mode when a new armature is added to a scene, just like mesh, curve, and surface objects.

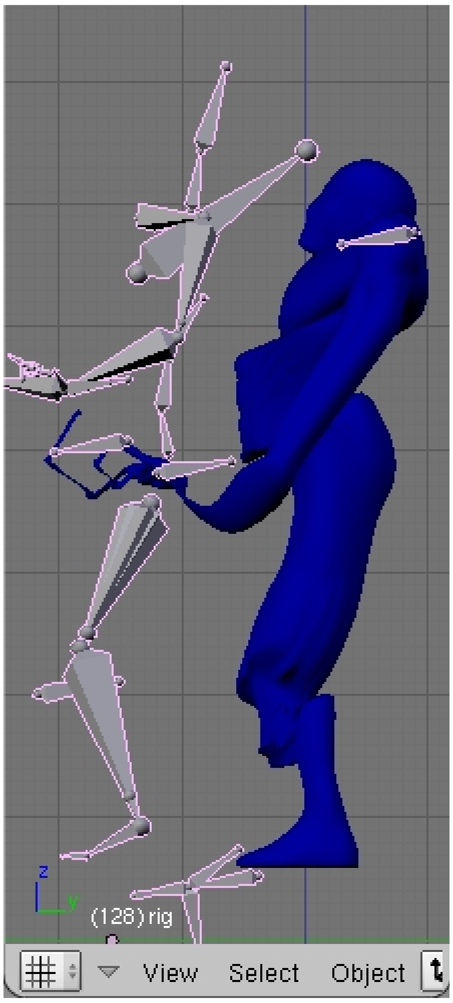
Armature Edit Mode



Bones are manipulated as pairs of points while in edit mode

Edit mode is where you add, remove, and place bones. In edit mode, bones are placed by moving points. Each bone has two points, a root and a tip. A new armature object has one bone by default, but more can be added via the **SPACE** menu, or you can select the tip of any existing bone and **extrude(link)** a new one from it with the **E** key, or **CTRL+LMB** click (not drag, that would do lasso select) in the 3D view. You can see the number of bones and bone points displayed in the top header. Extruding creates a bone that is a child of the bone from which it was extruded, and gives these two bones the special relationship of being *connected*. You can change this by toggling the **Con** button in the **Armature Bones** buttons panel while you have the bone in question selected.

Armature Object Mode



Bad things happen when an armature or it's mesh are transformed relative to each other

In object mode, an armature will behave just like any other singular 3D object (even though it contains multiple bones, which are objects themselves). Armatures can be moved about, resized, and rotated. It's good practice however, not to perform transformations on armatures. For armatures to deform a mesh object properly, the location, orientation, and size of the mesh--relative to the armature--***must*** stay constant. The only way ensure this is to constrain the mesh object's location, rotation, and size to the armature object. To make the mesh deformed by the armature, we have to either make it a child of the armature, or use a modifier. Modifiers have more helpful features, so it is the preferred method.

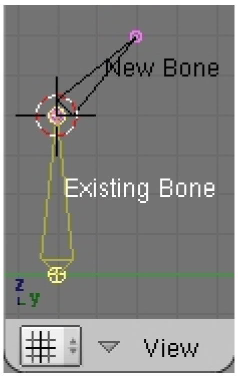
Armature Pose Mode

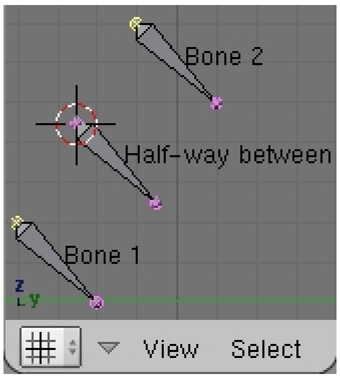
Pose mode is the mode where posing and animating of the armature bones can be done. Bones are technically sub-objects of the armature. The total number of bones, and number of selected bones is displayed in the top header. Besides posing and animating, you can also create and manage constraints for bones in pose mode. In edit mode, you define your armature's rest position. In pose mode, when you clear all bones of transformations (**ALT+R**, **ALT+S**, **ALT+G**), the rest position is what you should have, assuming you don't have any constraints preventing your bones from returning to the rest position. You can also make objects outside the armature children of a bone, by selecting the object with **CTRL+LMB**, and then parenting it to a bone with **CTRL+P**.

Building an Armature/Rig: Construction Techniques

Knowing how to add and move bones isn't everything you need to know to build an armature rig. A lot of work with armatures is done with **SHIFT+S** snapping, and cursor-as-pivot operations. You should understand the difference between using the **Bounding Box Center** as the pivot point (**,**), and using the **3D Cursor** as the pivot point (**.**). Throughout the rig tutorials in this section you will be required to do a lot of different snapping and cursor-pivoting operations.

Placing a New Bone

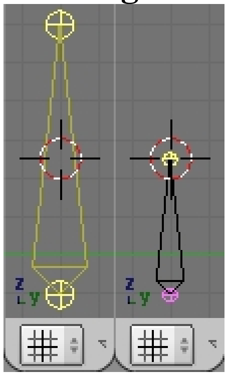




Often a new bone needs to be added in the same location as an existing root or tip point. To do this, snap the cursor to the existing point and add a new bone with the **SPACE** menu.

You can also place a new bone exactly halfway between two other bones. To do this, select the root points of two existing bones and snap the cursor to them both. It will then be halfway between them, and you can then add the new bone.

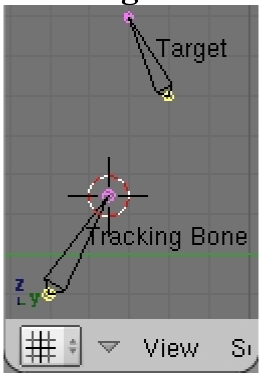
Resizing a Bone



There are two ways to resize a bone. The easy way is to select the root and tip points, snap the cursor to the selection, and then snap the tip point to the cursor. This makes the bone half as long as it was to begin with. In the tutorials, when you read, "resize the bone to 50%", this is what you want to do. If you see "resize to 25%", then just do this twice.

The other way is to snap the cursor to the root point, set the cursor as the pivot point, and then scale the tip toward or away from the cursor. This method allows you to make the bone any percentage of it's original size.

Pointing One Bone at Another Bone



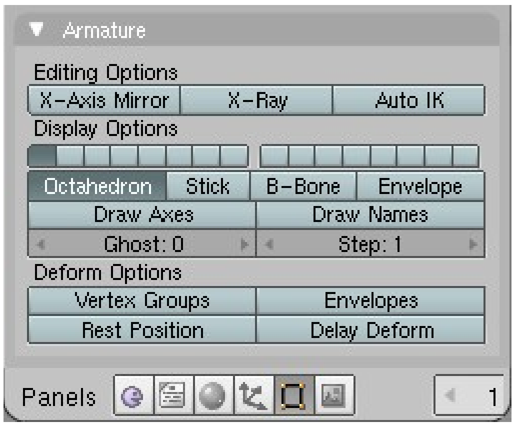
A lot of the rig designs require one bone to track (point at) another bone. This means that the bone that does the tracking needs to be pointing exactly at the target bone. To do this, you snap the cursor to the root points of both bones, and then snap the tip of the tracking bone to the cursor.

Armatures and Buttons

Many features for armatures and bones are accessable through the buttons window. Pictured below, are two panels from the editing buttons (F9), the Armature panel and the Armature Bones panel.

The **Armature Bones** panel changes based on a number of factors which include:

- The armature's current mode
- The bone or bones that are selected
- The status of the selected bone as part of an IK chain or not.



Armature

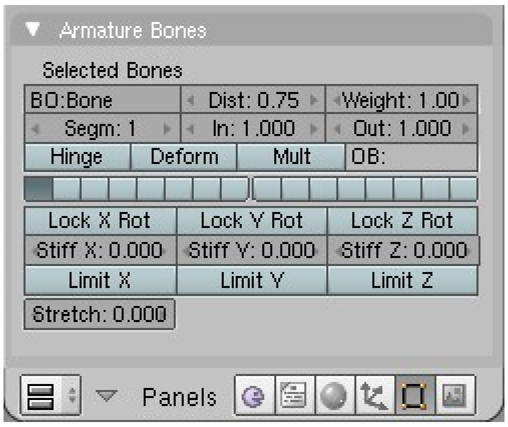
Editing Options: X-Axis Mirror | X-Ray | Auto IK

Display Options: Octahedron | Stick | B-Bone | Envelope; Draw Axes | Draw Names; Ghost: 0 | Step: 1

Deform Options: Vertex Groups | Envelopes; Rest Position | Delay Deform

Panels

These effect all bones of the armature



Armature Bones

Selected Bones

BO:Bone | Dist: 0.75 | Weight: 1.00
Segm: 1 | In: 1.000 | Out: 1.000
Hinge | Deform | Mult | OB:
Lock X Rot | Lock Y Rot | Lock Z Rot
Stiff X: 0.000 | Stiff Y: 0.000 | Stiff Z: 0.000
Limit X | Limit Y | Limit Z
Stretch: 0.000

Panels

These buttons effect the currently selected bones

The Armature Panel

Editing Options

- **X-Axis Mirror:** Enables mirrored editing for all mirrored bones. Use SHIFT+E to make mirrored extrusions, when X-Axis Mirror is active.
- **X-Ray:** Causes the armature to be drawn in the scene last, making it visible through all other objects, except for other objects that are also X-Ray active (see **Draw Panel** (F7)).
- **Auto IK:** This is a new feature and is still fairly undeveloped at this point. Automatic IK allows you to pose an armature by clicking and draging on any bone. Auto IK generally doesn't work with rigs, and should mostly be used as a posing tool for FK armatures.
- **Ghost:** Enables ghost drawing, which is visible if the bones are animated. The number here is the number of ghost copies Blender will draw.
- **Step:** The number here is the number of frames between ghost copies.

Display Options

- **Bone Layers:** These work in the same manner as the world layer buttons. Each button represents a layer, and when active, that layer is visible. Bone layers are a setting that is local to each armature.
- **Display Modes:**

These buttons set the armature display mode (explained below).
- **Draw Axes:** Enables axis drawing for all bones of the armature.
- **Draw Names:** Enables drawing of bone names in the 3D view.

Deform Options:

- **Vertex Groups:** Enables bones to deform a mesh via vertex groups.
- **Envelopes:** Enables bones to deform a mesh via envelopes.
- **Rest Position:** Forces the armature into rest position while in pose mode.
- **Delay Deform:** Disables deformation of children mesh objects.

The Armature Bones Panel

Selected Bones:

- **OB:** The name of the bone shows here. You can also use the Transform Properties panel in the 3D view (N).
- **Dist:** The envelope distance shows here. It is expressed in units from the surface of the bone (in envelope mode) to the outside of the envelope field.
- **Weight:** The weight that is used to calculate vertex influence amounts when using envelopes.

- **Segm:** The number of b-bone segments.
- **In:** The blend-in value. Effects the shape of the b-bone bezier curve.
- **Out:** The blend-out value. Effects the shape of the b-bone bezier curve.

- **Hinge:** This option allows bones to defy the rules of the parent/child relationship. When enabled, this feature causes bones to keep the appropriate location according to the parent's coordinate system, but not size and rotation.
- **Deform:** Enables bones to affect mesh vertices. Also includes bones for consideration during automatic vertex group creation.
- **Mult:** Causes vertex group weights to be multiplied by envelope weight. When enabled, a weight of 0 assigned to some verts will cause them to be uneffected by the envelope.
- **OB:** Specifies another object to be used as the visible bone geometry.

- **Bone Layers:** Indicates which layers this bone belongs to.

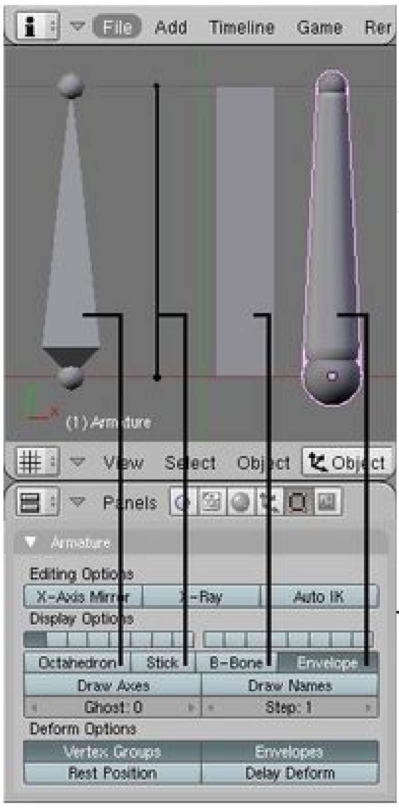
These buttons only appear for bones that are part of an *IK* chain:

- **Lock Axis Rot:** Allows you to prevent the bone from rotating on a specified axis.
- **Stiff:** Values for resistance to rotation.
- **Limit Axis:** Gives access to DoF (Degrees of Freedom). Allows minimum and maximum rotation values to be specified.

---”---

- **Stretch:** Allows bones to scale bigger or smaller in response to IK effector movement.

The Armature Display Modes

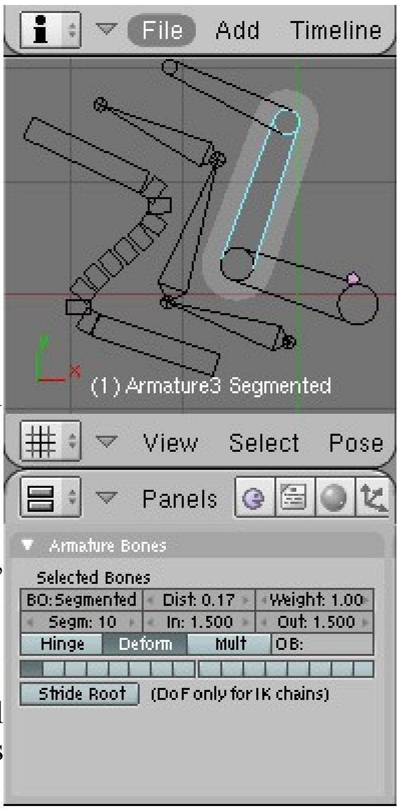


Armatures have four different methods of graphically representing bones, called **Display Modes**. Each display mode is only a different method of representing an armature. No matter which mode you use, no changes are made to the bones or the armature.

However, some display modes do give you access to bone properties and special visualizations that the other display modes don't.

(**left**) This bones are actually identical, just using different representations

(**right**) Here you see three identical armatures, each with only three bones



BO:Segmented | Dist: 0.17 | Weight: 1.00
Segm: 10 | In: 1.500 | Out: 1.500
OB:

Octahedron This is the classic display mode for bones. Before the other draw types existed, this is how bones looked (and it's a very professional looking way to draw them).

B-Bones The B-Bones display mode shows the shape that bones take when segmented (while in pose mode).

Stick For animators, this mode is perfect, because the bone objects don't hog a bunch of screen space which is better used for drawing the character. After all, it's the character that is being animated.

Envelopes The Envelopes display mode shows the influence areas of the selected bones.

Moving the Mesh

Armatures are pretty useless if they don't move the parts of our mesh that makeup our characters. There are two ways to do this in Blender:

- **Vertex Groups:** Vertices are assigned to groups with names. All verts in a group are effected by a bone with the same name as the vertex group *(assuming the armature is assigned to effect the mesh, by way of parenting or modifier)*.
- **Bone Envelopes:** All verts that reside within the limits of a bone's envelope field are effected by the bone *(assuming the armature is assigned to effect the mesh, by way of parenting or modifier)*.

Armatures and Hotkeys

Here are some of the hotkeys for working with bones in edit and pose mode. Some keys may not be listed, such as **CTRL+D**, which duplicates a bone in edit mode. It can be assumed that **CTRL+D** duplicates bones since it duplicates other types of objects, and Blender typically uses the same key for the same functions, even if the objects or modes are different.

Edit Mode

- **CTRL+N**: Recalculates the bone roll angles.
- **CTRL+P**: Makes the selected bone the child of the selected active bone.
- **W**: Opens a menu to either rename bones to the suffix of the opposite side, or to subdivide the currently selected bones (suffix mirroring works on duplicated bones, which will have a number suffix after the L/R suffix).
- **M**: Opens a menu of options to mirror the selected bones.
- **L**: Selects all bones connected to the bone currently under the mouse arrow.
- **ALT+S**: In b-bone display mode, this scales the size of the display thickness. In envelope display mode, it changes the outer envelope size (the **Dist** value from the **Armature Bones** panel).

Pose Mode

- **CTRL+ALT+C**: Give a constraint to the active bone, targeting one other selected bone.
- **CTRL+I**: Give an IK Solver constraint to the active selected bone, targeting the other selected bone.
- **CTRL+C**: Opens a menu to copy attributes of the selected active bone, to all other selected bones.
- **M**: Opens a menu of options to mirror the selected bones.
- **L**: Selects all bones connected to the bone currently under the mouse arrow.
- **ALT+S**: In b-bone display mode, this scales the size of the display thickness. In envelope display mode, it changes the outer envelope size (the **Dist** value from the **Armature Bones** panel).

Constraints and Axis Locks

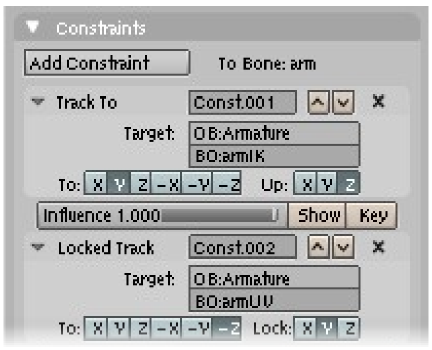
Introduction To Blender Constraints

Constraints are object features that define special relationships between objects, and are the standard method for controlling characters among all 3D animation packages that still implement a more traditional approach to digital character animation. In Blender, constraints can be given to any type of object or bone object. Constraints are accessed via the object buttons (**F7**). After you press the **Add Constraint** button and select from the menu, a constraint UI is added to the panel. Your active object now has this constraint, but you must enter the name of a target object in order to establish the relationship. In Blender, inverse kinematics are done with the IK constraint. When using a constraint on a bone object and targeting another bone (instead of an object from outside the armature), you must first enter the name of the armature object into the **OB:** (OBject) field. A new **BO:** (BOne) field appears, where you can place the name of the target bone.

Bones are given colors for various reasons:

- **Yellow:** A bone with an IK constraint.
- **Orange:** A bone with an IK constraint but no target.
- **Green:** A bone with any other kind of constraint.
- **Blue:** A bone that is animated.
- **Purple:** The Stride Root.

Universal Constraint Features


Constraints
Add Constraint To Bone: arm
Track To Const.001
Target: OB:Armature
BO:armIK
To: X Y Z -X -Y -Z Up: X Y Z
Influence 1.000 Show Key
Locked Track Const.002
Target: OB:Armature
BO:armUV
To: X Y Z -X -Y -Z Lock: X Y Z

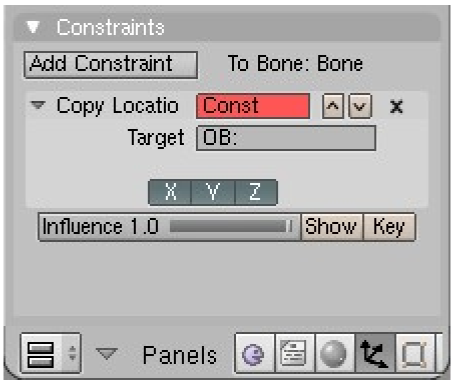
With the exception of the null constraint, All constraints have the following things in common.

All constraints have an influence value slider, which is linked to an **IPO Curve(link)**. All constraints have buttons next to this slider that allows you to set key frames into the influence IPO curve from the constraint UI.

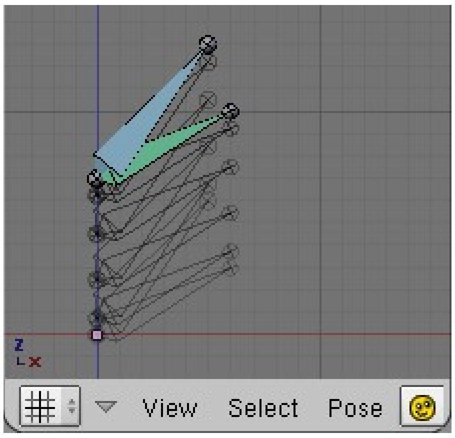
Constraints are added from the **Add Constraint** menu in the **Constraints** panel in the object buttons, F7. When you add a new constraint, it will go to the active object, or the active selected bone. When a constraint is added, it's name field appears red. This is because it does not yet function because it does not yet have a target specified. All constraints require a target object; either a world object, or an armature bone object. But not all constraints work with bone objects, and not all constraints work with world objects.

Constraints are evaluated in a specific order, and the order can be viewed and changed inside the **Constraints** panel. Each Constraint has a pair of arrow buttons in its top right corner, which are used to move the constraint up or down the constraint stack.

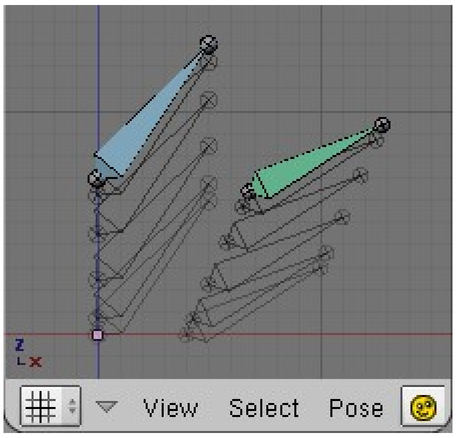
Copy Location



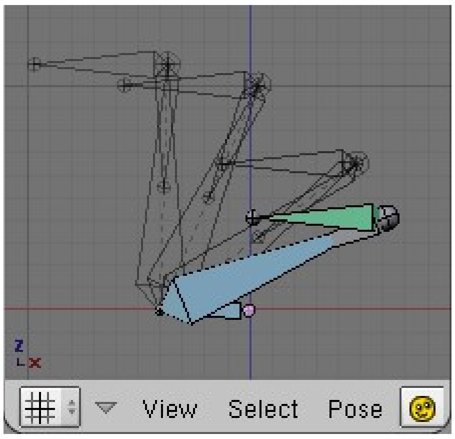
Copy Location forces the object to have the same location as it's target. When this constraint is used on a bone and another bone is the target, a new button--labeled **Local**--appears next to the **X Y Z** buttons. Using the local button causes the local translation values of the target to be given to the constrained object. In rest position, all bones are at the local location of (0,0,0).

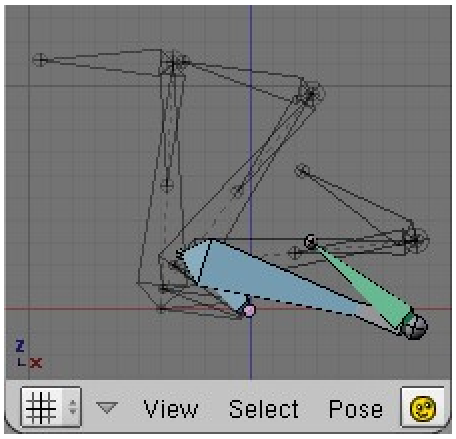


Using global space. This is the default behavior, it's what happens when the local button is not activated.



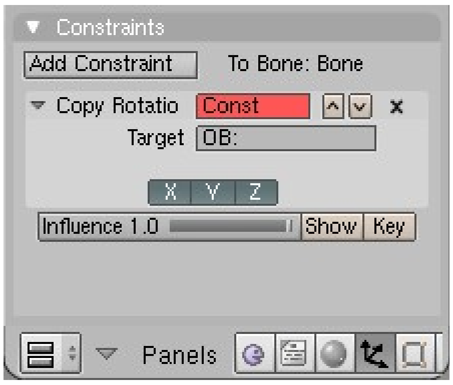
Using local space, local button activated.



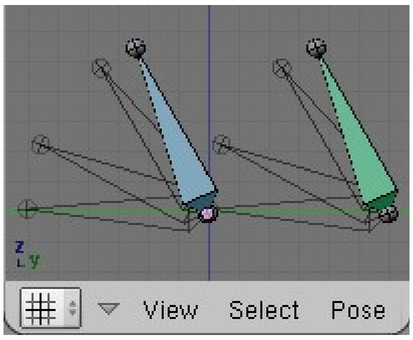


In these last two images, the constrained bone (shown in green) is actually the child of the root bone (the bone at the beginning of the chain). This demo shows possible uses for the location constraint in a rig. Note that the green bone still inherits rotation from the root because the root is it's parent. This is by design though, the green bone could be the child of any of these bones, or none of them.

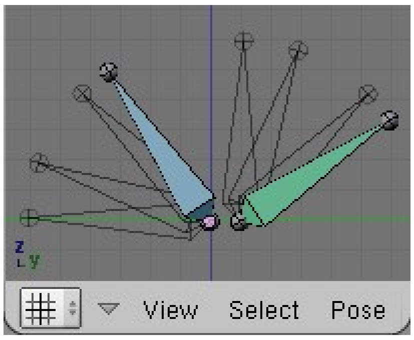
Copy Rotation



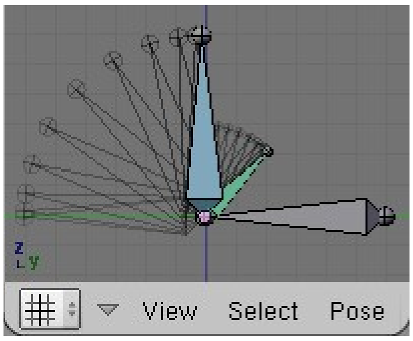
Copy rotation causes one object to match the rotation of a target object. For bones, a local option will appear, allowing you to use local space. Imagine you have a bone pointing up and a bone pointing down. If you use local space, each can point different directions, but when the target bone moves to its left, the effected bone will move it *its* left.



Using global space

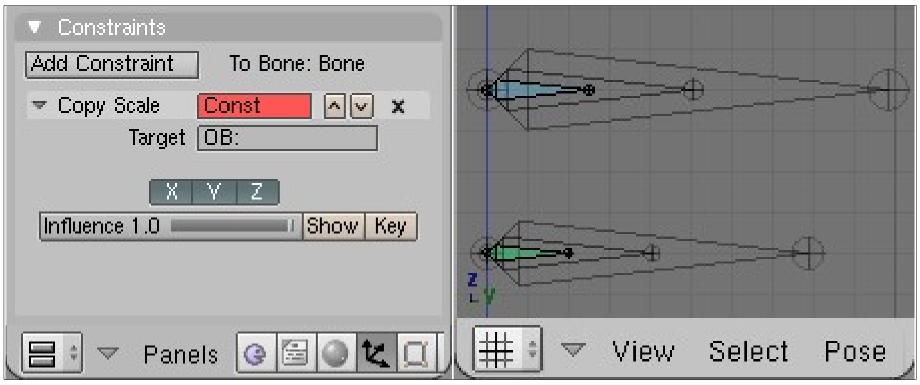


Using local space



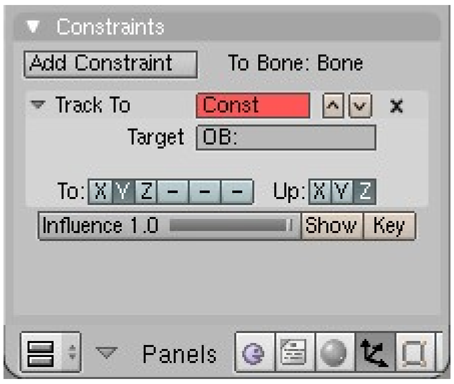
Here is one good use of the rotation constraint and local space. By setting the influence value to 0.5, the small bone will rotate half as far as the target. This is useful for character joints.

Copy Scale



Copy Scale forces the effected object to have the same size as the target. All bones have a (1,1,1) size in rest position. We can draw a bone that is 10 million times longer than the bone right next to it, but in pose mode, they both have a starting size of (1,1,1). You should keep this in mind if you're going to be using the copy scale constraint.

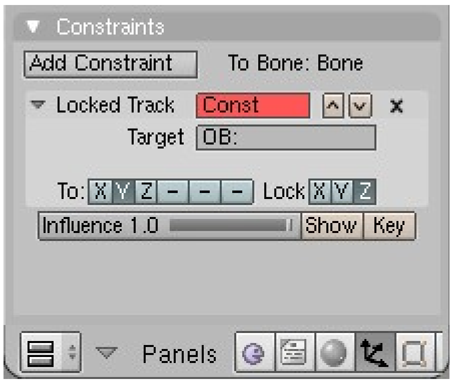
Track To



Track To rotates an object to point at a target object. This constraint also forces the object to be "right-side" up. You can choose the positive direction of any of the three axes to be the up-side. This constraint shares a close relationship to the IK constraint in some ways. This constraint is very important in rig design, and you should be sure to read and understand the page on tracking, as it centers around the use of this, and the IK, constraints.

To: The axis that points to the target object
Up: The axis that points upward

Locked Track



Locked Track is a difficult constraint to explain, both graphically and verbally. The best real-world example would have to be a compass. A compass can rotate to point in the general direction of it's target, but it can't point *directly at* the target, because it spins like a wheel on an axel. If a compass is sitting on a table and there is a magnet directly above it, the compass can't point to it. If we move the magnet more to one side of the compass, it still can't point *at* the target, but it can point in the general direction of the target, and still obey it's restrictions of the axel.

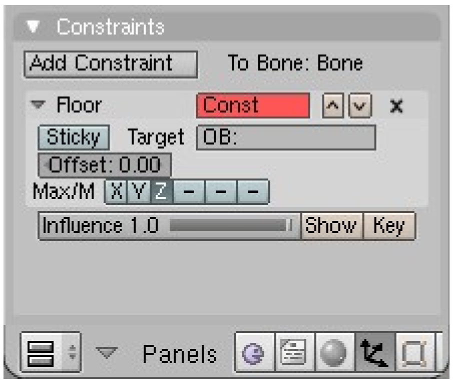
When using a Locked Track constraint, you can think of the target object as a magnet, and the effected object as a compass. One axis will function as the axel about with the object spins, and another axis will function as the compass needle. Which axis does what is up to you! If you have trouble understanding the buttons of this constraint, read the tool-tips, they're pretty good. If you don't know where your object's axes are, turn on the **Axis** button in the **Draw** panel, object buttons, **F7**. Or, if you're working with bones, turn on the **Draw Axes** button, **Armature** panel, editing buttons, **F9**.

This constraint was designed to work cooperatively with the Track To constraint. If you set the axes buttons right for these two constraints, Track To can be used to point the axel at a target object, and Locked Track can spin the object around that axel to a secondary target.

This is all related to the topic discussed at length in the Tracking section.

To: The axis that points to the target object
Lock: The axis that is locked

Floor



The Floor Constraint allows you to use a target object to specify the location of a plane which the affected object cannot pass through. In other words, it creates a floor! (or a ceiling, or a wall). This only works with planes of the global coordinate system. This means that if you rotate the target object, the floor plane will *not* be angled.

Animation Tip:

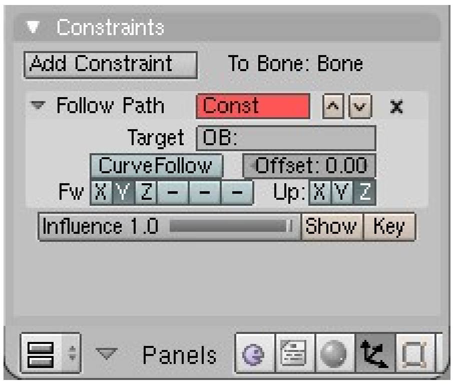
When you animate foot placement on the floor plane, ***always*** be sure to use the option **VisualLoc** from the **Insert Key** menu.

Sticky: Makes the effected object immoveable when touching the plane (cannot slide around on the surface of the plane), which is fantastic for making walk and run animations.

Offset: Offset from the position of the object center (can be positive or negative).

Max/Min: Will not pass below (positive axis) or above (negative axis) of target.

Follow Path



Follow Path places the effected object onto a curve object. Curves have an animated property that causes objects along the path to move. If you don't want objects on the path to move, you can give the path a flat speed IPO curve.

Follow Path is another constraint the works well with Locked Track. One example is a flying camera on a path. To control the camera's roll angle, you can use a Locked Track and a target object to specify the up direction, as the camera flys along the path.

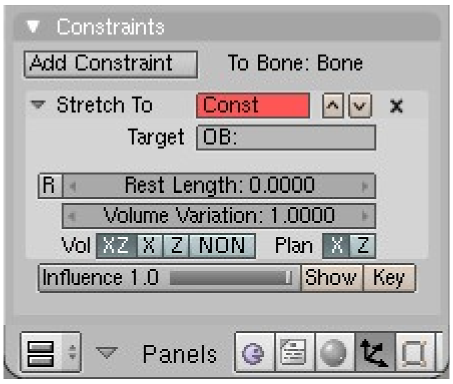
In order for this constraint to work, you have to have the **CurvePath** option activated (it's a property of the curve object). You can find it in the **Curve and Surface** panel in the editing buttons, **F9**.

This constraint does not work well with bones.

CurveFollow: Activate it to orient an axis of the object tangent to the curve.
Offset: Offset from the position corresponding to the time frame.

Fw: Axis to be oriented to the curve
Up: Axis to be upward

Stretch To



Stretch To causes the affected object to scale the Y axis towards a target object. It also has volumetric features, so the affected object can squash down as the target moves closer, or thin out as the target moves farther away. Or you can choose not to make use of this volumetric squash-n'-stretch feature, by pressing the **NONE** button. This constraint assumes that the Y axis will be the axis that does the stretching, and doesn't give you the option of using a different one because it would require too many buttons to do so.

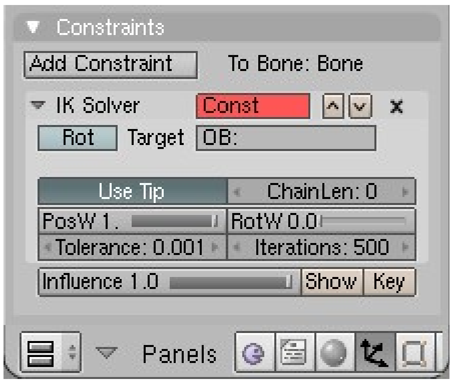
This constraint effects object orientation in the same way that Track To does, except this constraint bases the orientation of it's poles on the original orientation of the bone! See the page on Tracking for more info. Locked Track also works with this constraint.

R: Reset rest Length value.
Rest Length: Length of the object at rest position

Volume Variation: Factor between volume variation and stretching

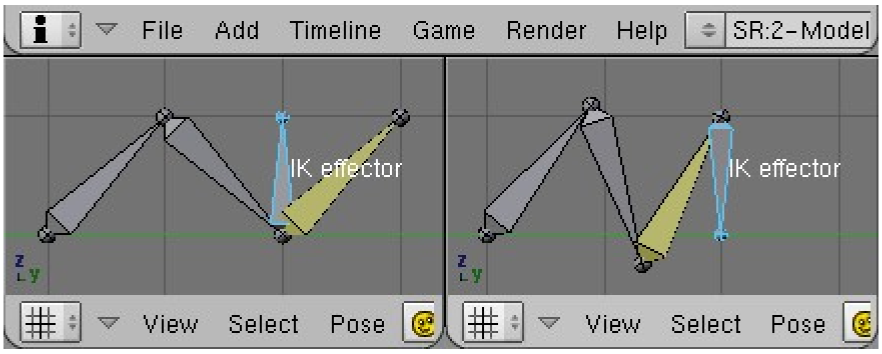
Vol: Wich axis to scale, keeping object's volume.
Plane:

IK Solver



The IK Solver constraint is Blender's implementation of inverse kinematics. You add this constraint to a bone and then it, and the bones above it, become part of the inverse kinematic solving algorithm.

Rot: Chain follows rotation of target



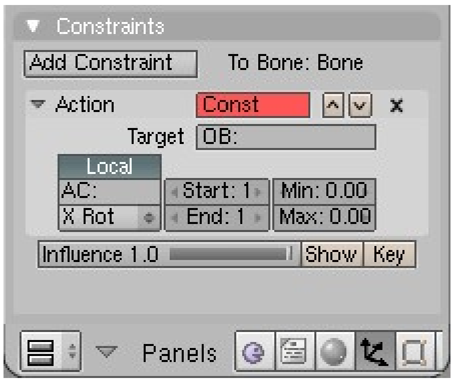
Left: Old behavior: wasted bones. **Right:** New behavior.

Use Tip: This option toggles between the old and new behaviors.
ChainLen: The number of bones above this bone that you want to be effected for IK. The default is 0, which means *all* bones above this bone will be used for IK.

PosW:
RotW:

Tolerance:
Iterations:

Action


Constraints
Add Constraint To Bone: Bone
Action Const
Target OB:
Local
AC: Start: 1 Min: 0.00
X Rot End: 1 Max: 0.00
Influence 1.0 Show Key

The Action constraint allows you to map any action to one of the rotation axes of a bone.

More will be added here.

Null

The null constraint doesn't do anything. It is an antiquated feature.

Preface: Rigging in Blender

This preface contains a lot of good info, so don't skip it! It has two sections: Some Guidelines, and Some Explanations. If you read about something in one of the rigging tutorials that appears to be assumed knowledge, then you can probably find more info on it here.

Some Guidelines

This section is intended to establish some common practices that are good to follow when working with armatures. The main reason being that they will improve the quality of your work and make it easier for others to understand and work with your rigs.

Mesh to Armature Assignment

In Blender there are two ways to attach a mesh to an armature. The old method is to make the mesh the child of the armature. Because the parent/child relationship then exists between the objects, we can transform the armature in any way we like and the character stays together and functional, unbroken and showing no odd effects. This is not the case if we transform the mesh while the armature is not in rest position.

With the addition of the modifier stack to Blender, we now have the armature modifier, which affords us some additional features. You can learn more about the modifier features in the Armature Modifier documentation (http://mediawiki.blender.org/index.php/Manual/PartII/Modelling/Modifier/Armatures). There are a number of reasons why you should use an armature modifier instead of a parenting relationship, but I won't go into all of that here. **Don't use a modifier AND have the mesh as a child of the armature**. Doing so is like having two armature modifiers, and it makes the mesh go all screwy when the armature is not in rest position.

The best option is to use an armature modifier, and give the mesh location, rotation, and scale constraints, all targeting the armature. This means that both the mesh and the armature should be in the same exact location and, as always, they should both have 0 rotation and a scale of [1,1,1]. If your mesh center is not in the same location as the armature, just snap the cursor to the armature object, select the mesh, and press the **Centre Cursor** button in the editing buttons (**F9**). If your scaling and rotation are off, then you'll have to clear the rotation and size while in object mode, and then reapply those transformations while in edit mode.

Using these three constraints simulates a parent/child relationship, but also locks our mesh in place so we can't move it. As shown on the armature intro page, moving a mesh relative to the armature is a bad thing. Using these three constraints prevents that completely, and from a rig design point of view, this is a wonderful thing! This elimanates yet one more way that someone could break the character.

The COG and the Armature Center

Basically every character has three bones in common: COG, body, and hip.

COG

COG stands for Center Of Gravity. All things, everywhere in existence, have a COG. This is why CG characters should have a bone to represent the COG. When tossed into the air, the COG is the point about which something rotates. The COG bone is the top most level of the rig hierarchy. Every bone in the rig is somewhere in the tree underneath the COG. This means that transforming the COG transforms the *entire character*. This is the bone we animate if our character is going to be falling large distances, flying through the air, performing aerial acrobatics, or the like.

Body

Basically all characters have a body. It's the part of the character that the limbs are attached to. By having a body bone, we can move the character's body while the feet or hands remain stationary, because of IK effectors. Generally speaking, all characters should have a top-most hierarchy like this:

- COG
  - body
    - hip
- legIK.l
- legIK.r
- handIK.l
- handIK.r
- any other IK effectors

Note:

To be clear, hip is a child of body. Body and all IK effectors are siblings. Exceptions to this structure can certainly be used, but you should only make them for a specific purpose.

Hip

The hip bone should be the parent of both upperleg bones. In humanoid characters, there is one hip bone for both legs, and effectively one hip bone for *each* arm (but we call them shoulders, or clavicles). It is foreseeable that you may choose to make an alien character that has two hip bones, just as humans have two clavical bones.

The bigger point though, is that this bone moves a *part* of the body that one or more limbs are attached to, and still allows the IK effectors to remain stationary. For some more insite into the usage of such a bone, see the Bend-O-Matic Spine tutorial (pag. 73).

The Armature Center

This creates a bit of a redundancy issue. All bones of the armature are children of the armature object, which has a center of it's own. Transforming the armature center has the same effect that transforming the COG has, so why bother creating a COG bone?

The reason is that animated bones can be included in actions. Armature object animations cannot. So if you want to make an action of your character jumping and doing an aerial flip, you can't! You can animate it, but it has to remain as two separate animations, one would be the character jumping and landing, and the other would be the armature center's animation of moving through the air and rotating.

So what do we do? We use a COG bone. If a situation arises where you need to use the armature center instead, you still can, but you might have to use a little sleight-of-hand to make it work. If you animate with a COG bone, it will probably be moving away from the armature center. This means that if your character walks 10 units forward, rotating the armature is going to spin the character around a point that is 10 units away!

Plan Ahead

Especially when building very large rigs, it will benefit you to think ahead and note which parts of the armature will be the non-deforming bones. This way, you can deactivate **Deform** for the appropriate bones early in the building process, so that all duplicate bones will inherit this setting. This applies to the **W** key subdivide command as well.

When In Doubt, Snap To Grid!

- You should not place bones haphazardly.
- Designing a rig requires care and precision.

The armatures in these tutorials are drawn snapped to the grid, or to the cursor. Not all the joints of your characters need (or should) be placed exactly on the grid, but using the grid will help you place matching points when the need arises. Basically, the point is this: don't be careless with the placement of your bones. If it looks like two or more points share an exact location in the images you see here, then they do.

Layer Usage

The bones of an armature typically fall into one of three categories:

1. Manipulable rig elements: Bones meant for transformation by the animator.
2. Deform bones: Bones set with Deform active, these move the mesh and often don't need to be dealt with directly by the animator.
3. Non-manipulable rig elements: Elements of a rig that work as part of a larger system to cause an effect, but don't need to be dealt with by the animator.

When an animator works with a character, he wants to see two things:

1. The character, in one form or another.
2. The character's controls.

We don't need to have a plethora of bones on the screen during animation if only two or three drive the whole system. By using bone layers, we can organize bones into groups however we see fit, and only display the section of the armature we need to work with at any given time.

You should try to assign priorities to your groups, and then place them accordingly. For example, you will probably want to put all manipulable rig elements into layer one (the farthest left layer button), since these are the bones that the animator is going to be working with. If the animator is going to need multiple sets of manipulable rig elements, then place these into the far left layer buttons so they have the appearance of higher status.

The tutorials in this chapter will assume you can manage bone layers with this concept in mind, as you follow along, or whenever you feel it's a good time to do some organization. The tutorial images will be meant to present rigs in a way that either helps you understand them better, tries to show all the components, or just makes them look cool.

The IK and Constraint Hotkeys

When building an entire character rig, the IK hotkey **CTRL+I** comes in very handy for assigning IK constraints. Remember to make use of it always, because it's much faster and easier than the alternative.

The same should be said for the constraint hotkey, **CTRL+ALT+C**.

More goes here

- the dreaded bone roll angle
- bones internal vs objects external

- never parent to the IK effector

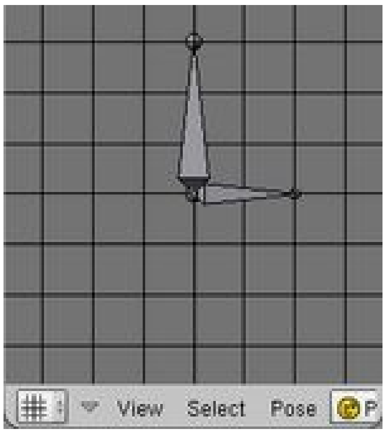
Some Explanations

If there some areas of rigging that still perplex you, then give these topics a good read. Hopefully you'll find the answers you're looking for here.

Understanding Hierarchies

Character rigging is completely based on objects hierarchies. If you don't understand how 3D object hierarchies work, then you are going to have a lot of trouble learning how to rig characters.

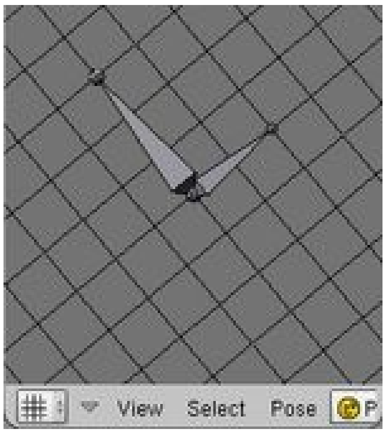
A Hierarchy is a way of arranging persons or things. In 3D computer graphics, all objects are children of the CG world. The CG world has a center and coordinate system, just like all of the objects in it.



Here we see two bones. The bigger bone is the parent, and the smaller one is the child, and the grid lines around them show the coordinate system of the parent bone.

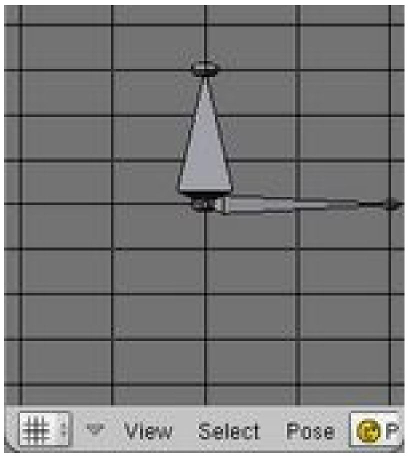
Let me elaborate. Every object has a coordinate system. You can think of this as being like that object's own special little world (and believe me, it is special!). In our example, the smaller bone is the child of the bigger bone--the parent. This means that the child bone exists inside the parent bone's special world.

- parent
  - child

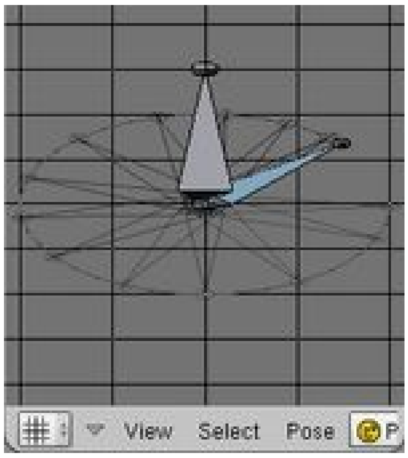


A child object is like a person standing on earth. Not only is earth spinning around its axis, but it's also orbiting the sun. As you sit reading this though, you probably don't perceive any of that. That's why people use to think the world is both flat, and the center of the galaxy (because at the center, it would be stationary, while everything else appears to be moving).

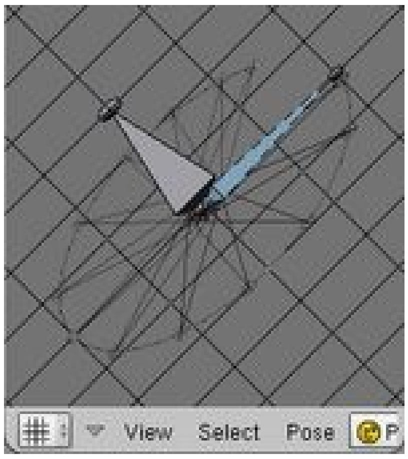
Our child bone is naive like this. In this image, the parent bone has rotated, but the child doesn't know a thing about it. As far as he's concerned, he's still standing in the same place, facing the same direction.



In this image, the parent is back to its original orientation, but now it has been scaled on the Y axis. Does the child know about this? Nope. The world he's living in may appear to be warped to you and I, but as far as he's concerned, nothing has changed.



So what happens if the child runs around in a circle? Nothing special. Just like the world he's living in, it looks warped to us, but looks normal to him!



For fun, lets see his world both rotated and warped:

You have to keep this in mind if you are going to be parenting other bones to stretchable bones.

Terminology Overview

note for editing: this section could go into a wiki glossary.

- floating bone: Any disconnected bone, but usually a single bone placed visibly outside the inner areas of the armature.
- root bone: The first bone of a chain.
- tip bone: The last bone of a chain.
- chain: A series of connected bones.
- parent: An object that has one or more children objects.
- child: An object that has a parent object.
- sibling: An object that shares a parent object with other children objects (like a sister/brother).
- IK effector: A bone that moves around in space to define the target location of an IK chain.

The Design-and-Test Two-Step

Character rigs have functional aspects and structural aspects. You design the structures in edit mode, and you implement the functions in pose mode. As you are building a character rig, both of these aspects should be cycled and tested so that each area of the character can be assessed at a local level.

For example, build and test one finger, then do the other fingers and test them together, then the hand, then the arm, then the shoulder... Build and test the foot, then the leg. Build and test the spine, then the hips, then connect the legs to the hips and test the functioning of the two together, seeing how one effects the other, or both effect each other.

This process involves constant changing between edit and pose mode. You can even include animation as a testing procedure. You can move, add, or remove bones from the armature while it still has animation. You can add, remove, or otherwise edit constraints between bones that may or may not have animation.

The idea is that you learn to manage each area of the armature individually, so that you are not overwhelmed with too many objects and connections later on. This will also help you find and eliminate small problems before you build more systems on top of them.

The tutorials here will assume that you can make use of this work concept at any point, as needed.

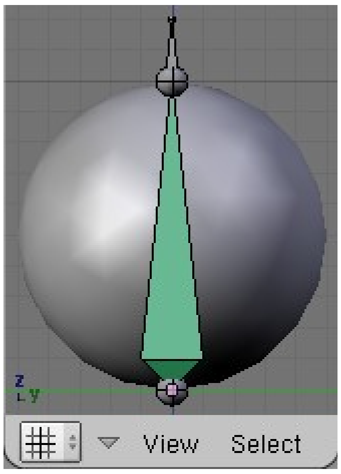
More goes here

- stretching vs reaching
- weight painting == vertex groups
- mechanical vs organics
- building armatures on planes

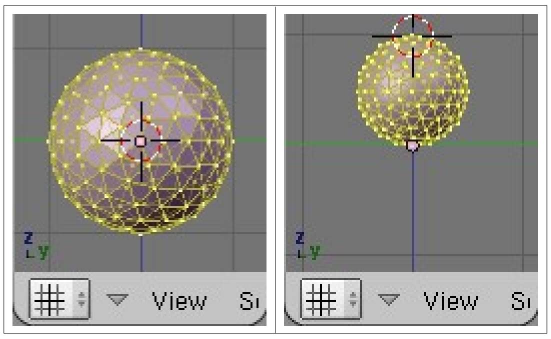
Some Beginner Rigs

This page will assume that you're still new to working with armatures and snapping operations, so these tutorials will be more specific about how to do each step. All of the following tutorials will assume that you know when to use operations like snapping.

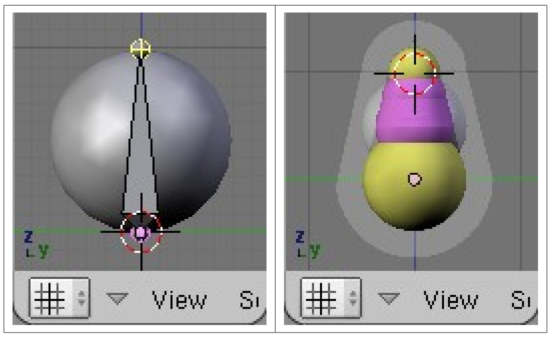
The Squash n' Stretch Ball



This is a great first-time rig. We're going to use the prinicle of squash n' stretch to make this animatable bouncing ball.



Start off with an empty scene and set the cursor to the origin, **CTRL+C**. Add a sphere. I'm going to use an icosphere because I like 'em. Set smooth on the sphere so it looks nice and not all faceted. Exit edit mode, clear the rotation with **ALT+R**, and then reenter edit mode. Place the cursor 1 unit above the origin and snap it to the grid with **SHIFT+S**. Press the **.** (dot) key to use the cursor as the pivot point. Now scale the sphere down to 0.5, holding the **CTRL** key. Exit edit mode.



Place the cursor back to the origin and add an armature. Turn on **X-Ray** in the **Armature** panel. In object mode, clear the rotation of the armature. Enter edit mode and place the tip of the bone as shown.

Snap the cursor to the tip point and add a new bone. Select the new bone and scale it down to 0.3, while holding **CTRL**.

Name the big bone, "stetcho", and the little bone, "target".

Change the display mode to **Envelope**. Select the bone and scale it up, as shown. Select target and turn **Deform** off.

Exit edit mode and select the mesh. Give it an **Armature Modifier(link)**, and type "Armature" (no quotes) into the **Ob:** field. Give the mesh three constraints: Location, Rotation, and Scale. Put "Armature" (no quotes) into the **Ob:** field for each constraint.

Now for one last step. Select the armature, enter pose mode, select stretcho, and give it a Stretch To constraint. Put "Armature" in the **OB:** field, and put "target" in the **BO:** field.

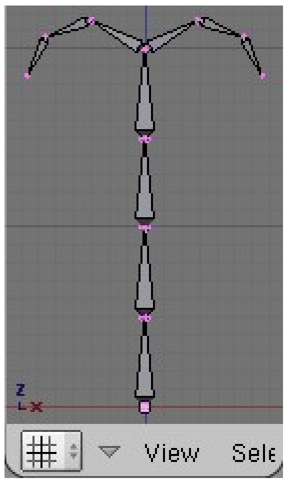
To make the rig easy to work with, you should put it back into octahedron display mode.

Some Extra Info

The reason why you want to use Stretch To for this character is that it automates the squashing and stretching for you. If you animated the rotating and scaling by hand, then you would have to create the squash n' stretch (expanding and thinning) effect manually (very inefficient).

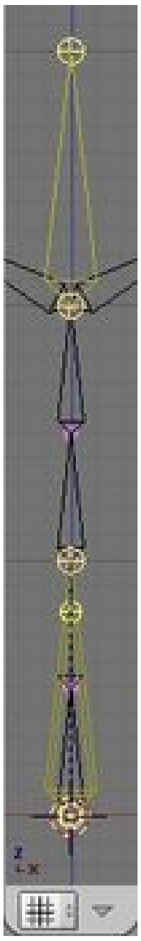
You can probably animate the armature object to make the ball bounce around. If you want, you could instead use a COG bone, where both bones would be the children. See the Rigging preface (pag. 26) for info on COG bones.

The Dancing Palm Tree

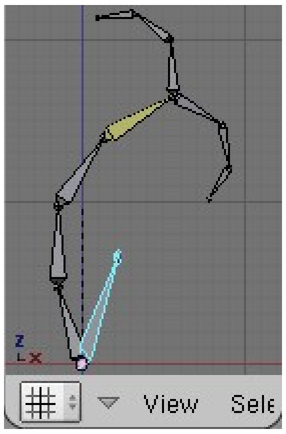


Reset the cursor with **CTRL+C**, add an armature, and clear it's rotation in object mode. In edit mode, make the bone 2 units tall, standing parallel with the Z axis. Use **SHIFT+S** where needed. Select the bone and subdivide it (**W**) two times. Enable **X-Axis Mirror**, select the tip point at the end of the chain and mirror-extrude it with **SHIFT+E**, then disable **X-Axis Mirror**. Extrude twice more with **E** to make the leafy-looking chains (these are just for show; after all, it *is* a palm tree!).

Snap the cursor to the tip again, add a new bone, and name it "target". Snap the cursor to the root point at the bottom (beginning of the chain), add a new bone, and name it "warped". Add another new bone, and move the tip down 0.2 units (holding **CTRL**), just so we can see it. Name this bone "widget".



Select widget, and then **SHIFT** select warped, and press **CTRL+P**, and choose "Keep Offset" from the **Make Parent** menu. Select target and then **SHIFT** select widget, and use **CTRL+P** again, and choose the same option. You've now got a hierarchy like this:



- warped
    - widget
        - target
- other bones...

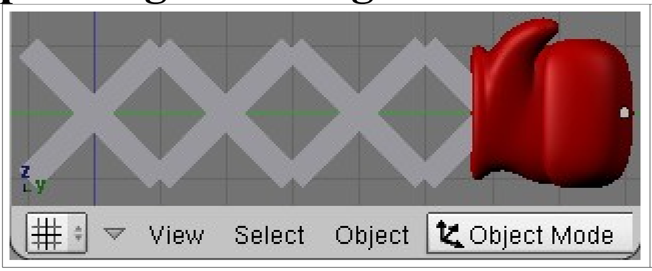
In pose mode, select target and then select the last of the vertical bones in the chain, press **CTRL+I** (and then confirm) to IK constrain the chain to target.

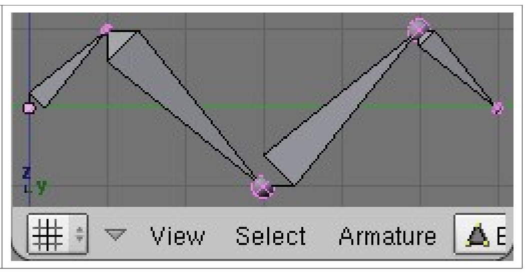
In pose mode, select warped and press **S** to scale, followed by **SHIFT+Z** to scale on X and Y only, and take it down to 0.5. Place warped and target into layer 3, and place widget into layer 2. You should only have layers 1 and 2 visible. Now rotate widget and watch it dance!

Some Extra Info

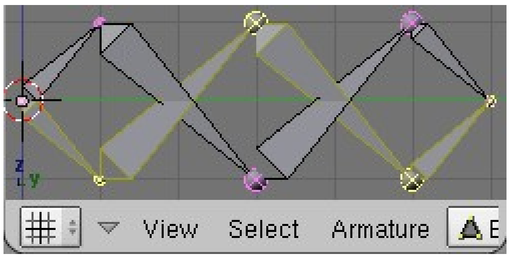
You could also do something like this with a Propagating Rotations Spine Rig (pag. 71), but you would have to have a rotating parent bone at the root of the spine to make the base rotate side to side.

Expanding Punching Glove Arm





Add your armature and--as always--clear its rotation while in object mode. In side view, draw some bones as shown. Be precise with your bone placements, especially for this rig.



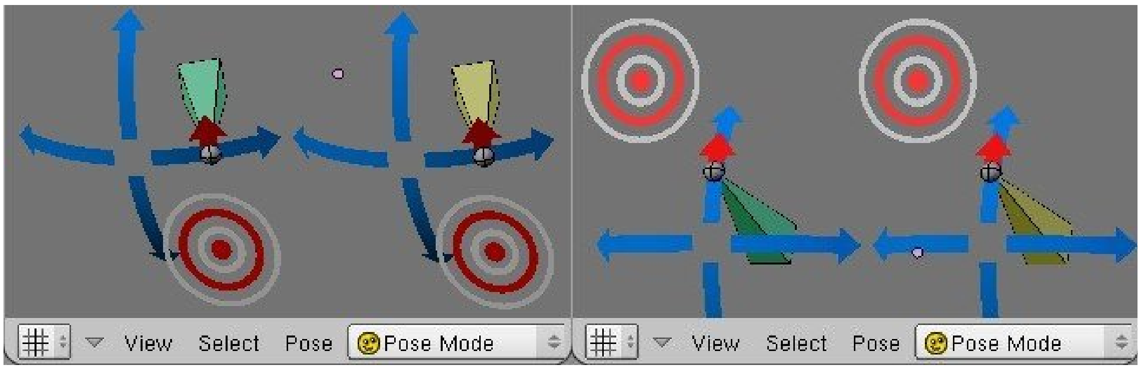
Place the cursor back at the origin (**CTRL+C**), select all the bones, set the cursor as the pivot point (**.**), press **CTRL+M** to mirror the bones, and select the "Z Global" option from the pop-up menu.

Sorry, I gotta shift my focus to the other pages right now, but I will be back!

Understanding Target Tracking

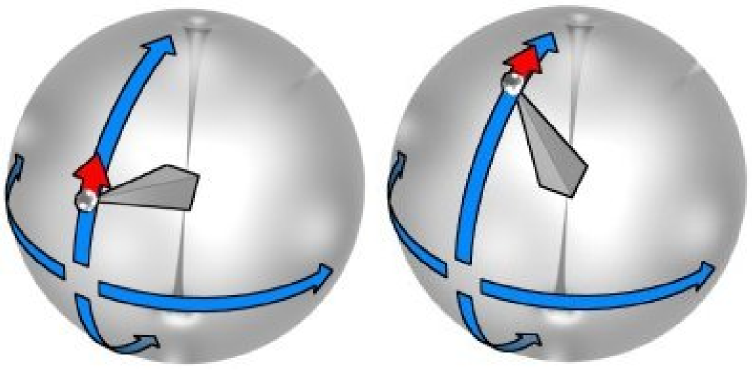
In many rigs, it's necessary to have one bone target another, free-floating bone. This relationship is the critical component that makes the last three spine rigs work, and is the foundation for the arm and both leg rigs shown here. Suffice it to say, this section is important, even if it is a little boring.

In this section we're going to learn about the behavioral difference of the **Track To** and **IK** constraints. Both of these cause the assigned bone to point to a target, but as that target moves the bone farther and farther away from it's original position, the bone's rotation around it's local Y axis takes on a life of it's own. We need to understand how these two constraints behave so we can try to tame them, and bend them to our will!



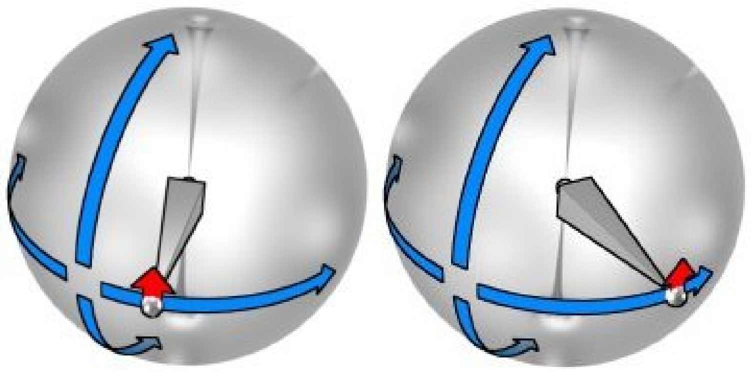
At first glance, Track To and IK seem to do the same thing

Track To Constraint

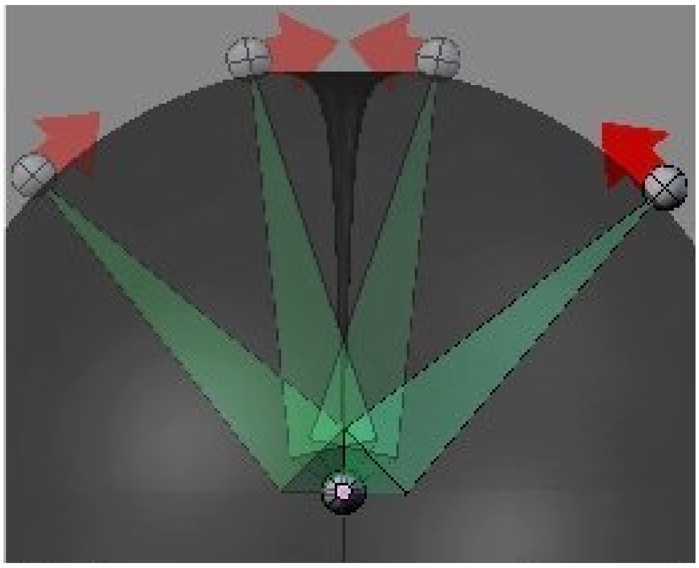


The Track To constraint does two things:

- It directs your object to point at a target with one of it's axes
- It also forces the bone to stay 'right-side up'.

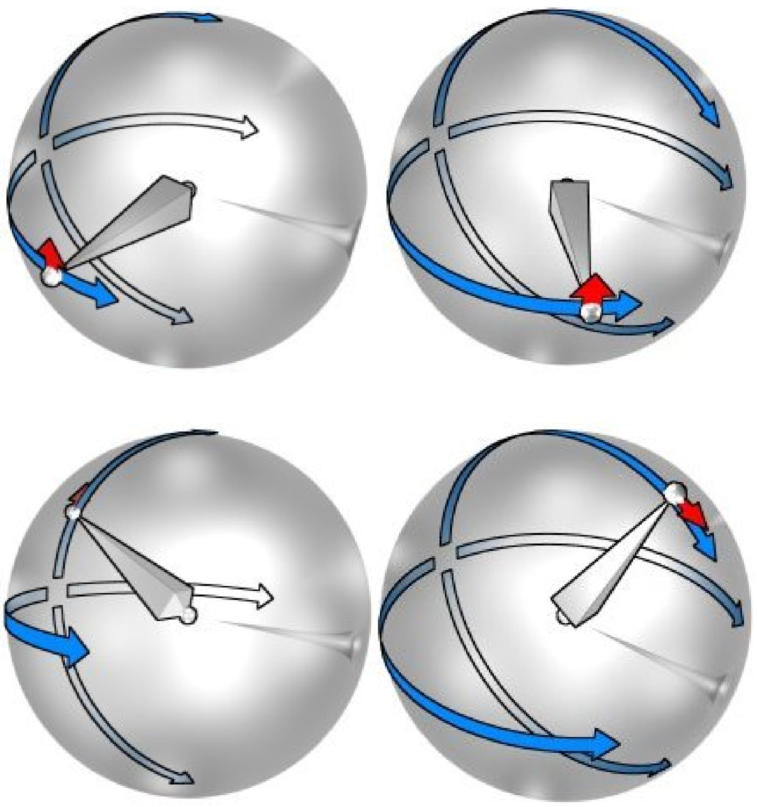


The direction that is considered 'up' is the global +Z direction.



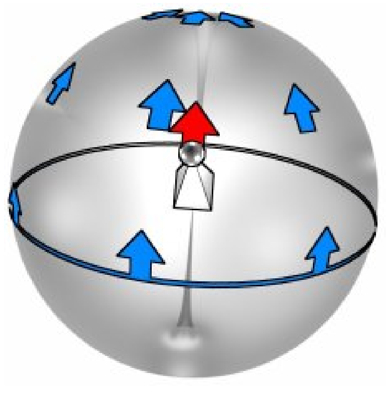
If the target crosses directly above or below the effected bone, it must flip over because--as dictated by the design of this constraint--the bone must always be right-side up. The side that is the 'right' side however, is your choice, but it has to be one of the three positive axes of the effected object. See *Track To* constraint at page 21.

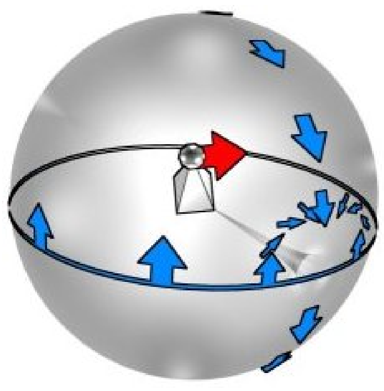
IK Constraint



The IK constraint can be used on a single bone to do the same thing as the track to constraint, but the IK constraint has a different behavior. It doesn't base part of it's orientation on the global Z axis. Instead, it keeps the rotations based on the orientation of it's rest position, relative to its parent. This is critical to the proper functioning of an IK chain. Imagine the implications if you use IK on the spine of your character, a spine with bones pointing *straight up!*. If IK worked like Track To does, your character's torso would be spinning in circles all the time. You could never have a character reach toward the sky or the ground, you couldn't allow your bipeds to stand up straight, arms at the sides. Character animation would be virtually impossible.

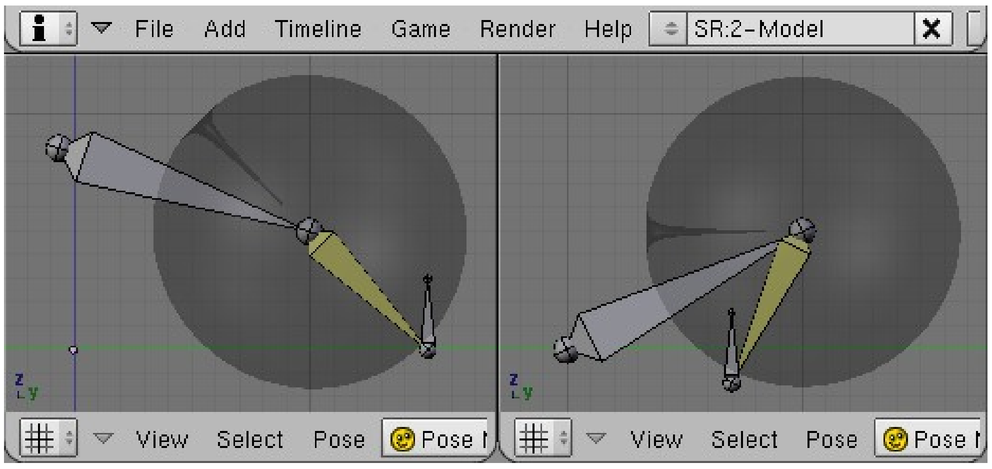
Making the Comparison

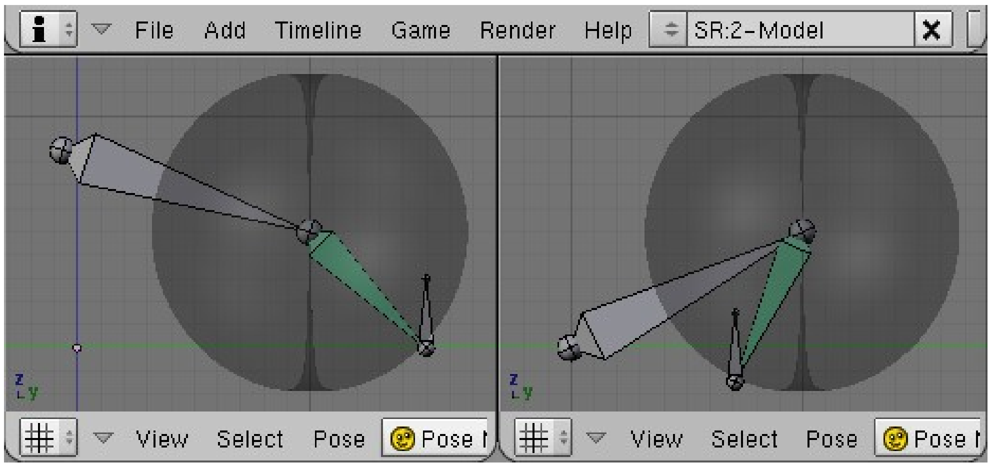

Track To

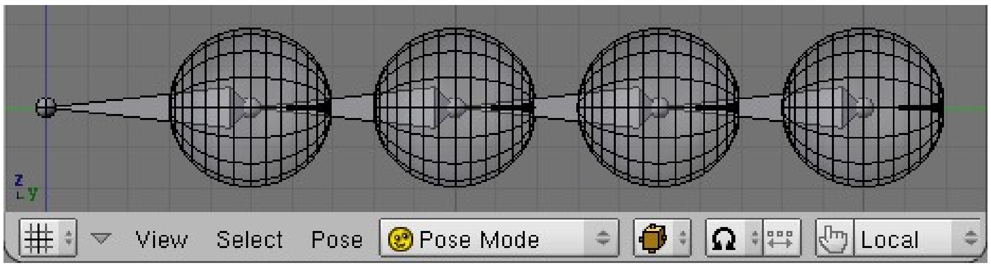

IK Solver

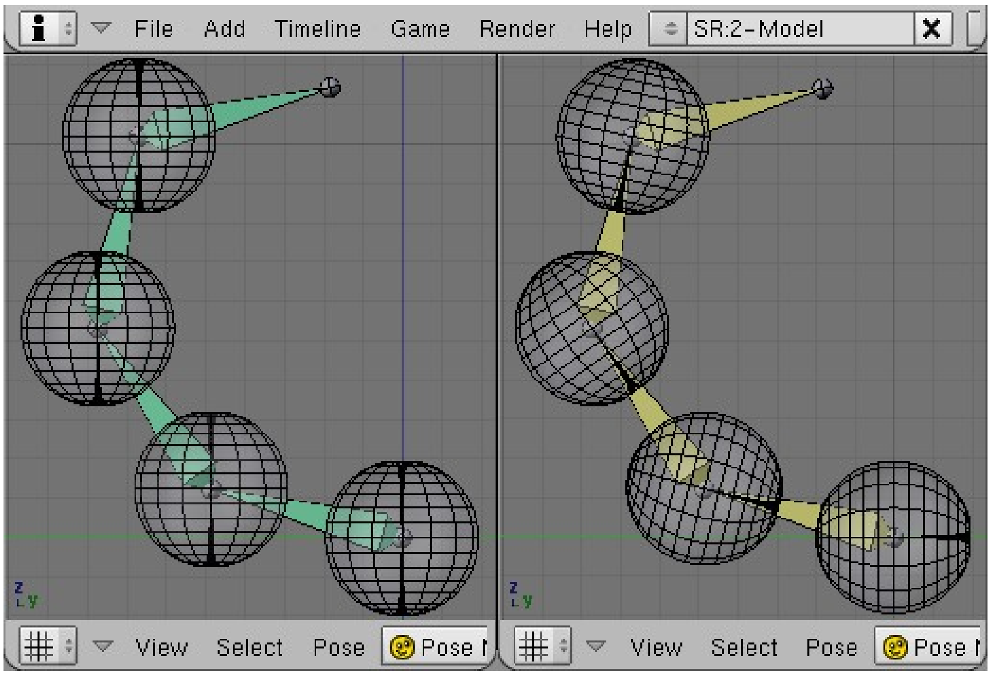
In these images, the pits in the spheres represent rotational poles. These are the locations where all possible Y axis rotations come to a single point. Track To has two poles, and the IK solver only has one.

Keep in mind that not only are the location and number of the poles different, but they are also based on different coordinate systems.

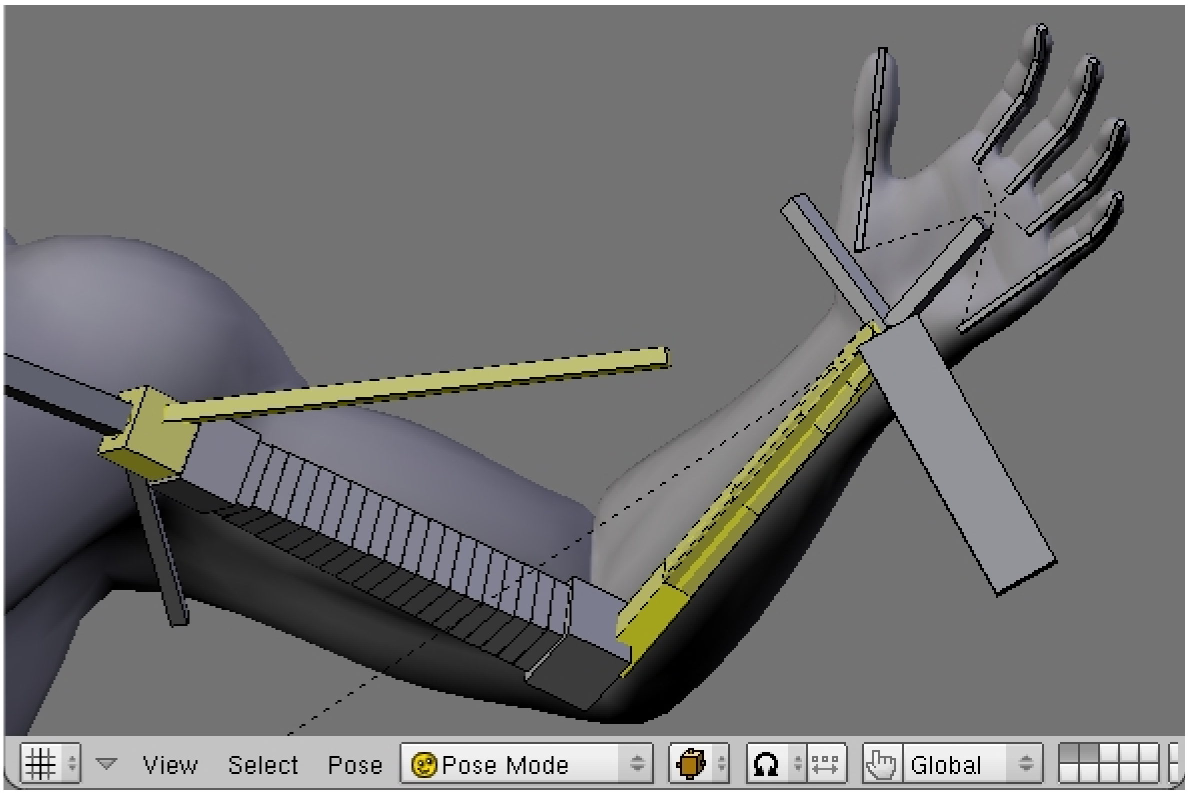





View
Select
Pose
Pose Mode
Local


File
Add
Timeline
Game
Render
Help
SR:2-Model
View
Select
Pose
Pose
View
Select
Pose
Pose

Arm Rig Design

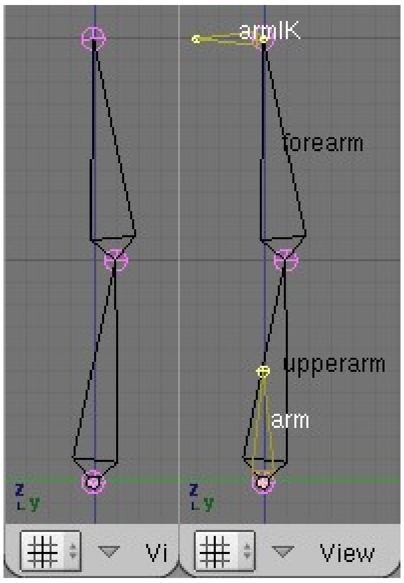


When designing a rig for a character's arm, it's imporant to consider the differences between computer bones and real bones. In the human forearm, there are two bones: the ulna and the radius. The wrist can rotate 180º, and it does this by moving the end of one around the other. With CG bones however, none of this matters. What matters is that the skin twists and the CG bones have to create that twist. The upperarm can also rotate--though not quite 180º--and it does this by rotating at the shoulder socket. But this also doesn't matter.

What this means is that the CG bones of both the upperarm and the forearm need to create a twisting motion. We can do this easily with a segmented b-bone, or if you want more control over the amount of twist at each section of the arm, you can use multiple bones.

Besides the issue of twisting, there is also the issues of elbow direction control, wrist rotation, and IK/FK blending. We will attempt to address all of these needs with the following tutorial about arm rigs.

Lets Build It!



In top or side view, draw your bones as shown. Notice the slight bend. This is extremely important. If you don't build it this way, Blender won't know which way the arm is supposed to bend.

1. Add a new bone at the root position that points exactly to armIK, then size it down 50%.
2. Add a new bone at the end of the chain. This will be the IK effector.
3. Now is a good time to do the naming.
4. Then go into pose mode and IK constrain forearm to armIK, and set the ChainLen value to 2.

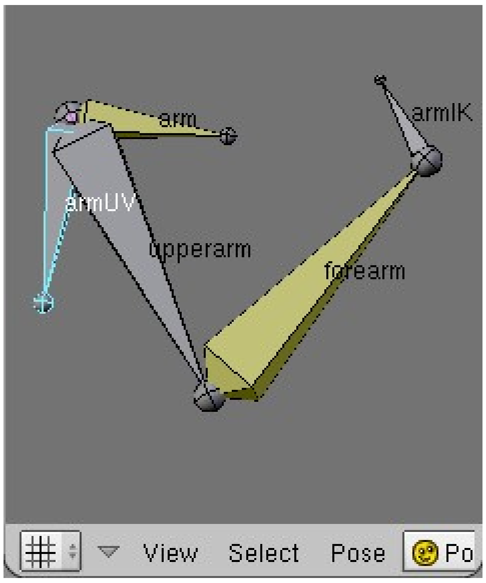
Elbow Direction Control

Ever seen someone do a dance called "the funky chicken"? In case you haven't, place your hands near your arm pits and flap your arms as if they were wings. When you do this, your hands are--for the most part--motionless. But your elbows are moving up and down. When you use an IK solver, it controls the placement of the character's hand, but it doesn't give you any control over the direction the character's elbow is pointing.

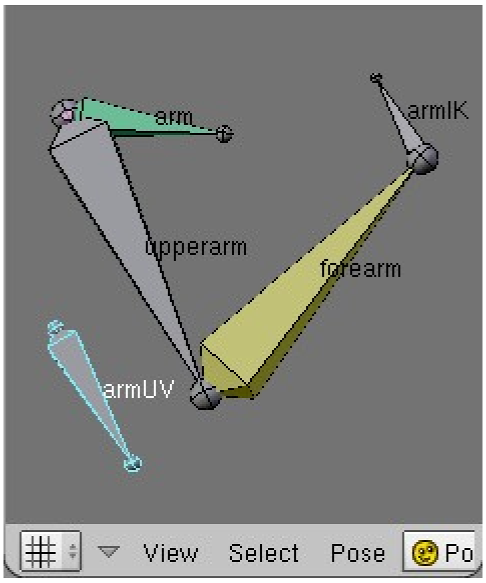
One solution might be to rotate the root bone of the chain. This *can* work fine for character posing, but it doesn't work for animation. Between keyframes, you are trusting Blender to choose the correct way to interpolate the rotation of the root bone, but the math behind IK chains and rotated root bones is not predictable. In short, it will lead to spastic, uncontrolable arm rotations. This is why we created an arm bone. We can control it's direction and roll, so it provides a good basis upon which to build an IK chain.

Finding Some Direction

For this next step, you have to choose which system you think will be the best to animate with.



Option 1: Using IK tracking, and a 1D rotation control bone



Option 2: Using Track To tracking, and Locked Track to give the up vector

But How Do I Choose!?

Basicly, the first option is nicer to work with because you don't have more bones floating around the character. The second method is more precise though, because the input that determines the elbow direction is not based on the rotation of another bone in the arm, but is instead a direct child of the COG.

Note:

Please see the page on Tracking (pag. 43) to be sure you understand what is happening here.

If you're not sure, here's some advice:

Option 1: Use this for most of your characters. It's compact, easier to use, and can handle most all situations just fine.

Option 2: Lets say you have a character that is going to be armwrestling, and then doing push-ups. In a scenario like this, you might want to use this design. I would also say that you might want to use FK to animate the armwrestling and then blend to IK for the push-ups, but it might be easier to use IK for the whole thing. This rig lends itself to a situation like armwrestling because the elbow is planted on a surface. This design allows you to set the target for the elbow on the table, so you know the elbow won't slide around on the table. *But*, because you're using IK, you'll have to use more keyframes on the hand to keep the elbow from going through the table, or lifting away from it.

Animation Tip:

In a situation like this, you can snap the cursor to the armUV bone, and rotate the armIK bone around with the cursor set as the pivot point. This will allow you to easily keyframe the IK effector traveling through a perfect arc.

Building Option 1

In option 1, your hierarchy should be like this:

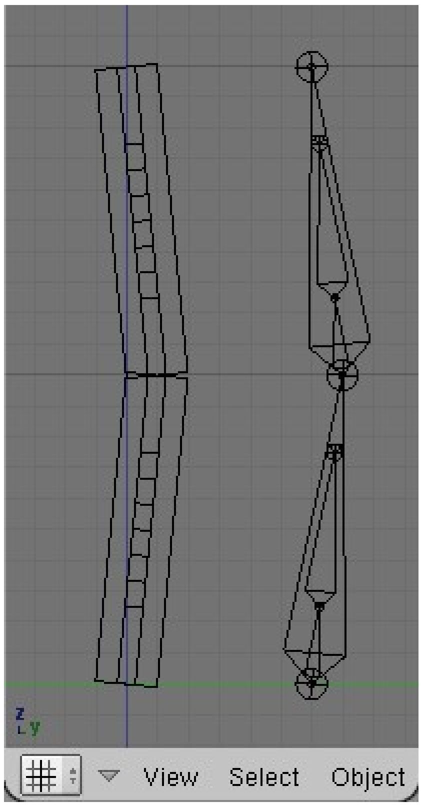
- armIK
- arm
  - armUV
    - upperarm
      - forearm

The key to making option 1 usable, is to lock all the axes of armUV, except for the rotation axis that aligns with the Y axis of arm. You can orient armUV anyway you like, but ideally, one of it's axes aligns with the Y axis of the bone arm.

Building Option 2

In option 2, your hierarchy should be like this:

- armIK
- armUV
- arm
  - upperarm
    - forearm



After you decide which technique you want to use to control elbow direction, you can then build more bones on top of the existing bones. If you decide to use FK and IK in the same rig, it is desireable to have one bone for the upperarm, and one bone for the forearm. To make the arm bones twist though, we my want to have many more bones for each part of the arm.

Shown on the right are two instances of the same armature, but in two different draw modes. In b-bones display mode, we see that the longer interior bones are segmented.

IK/FK Blending

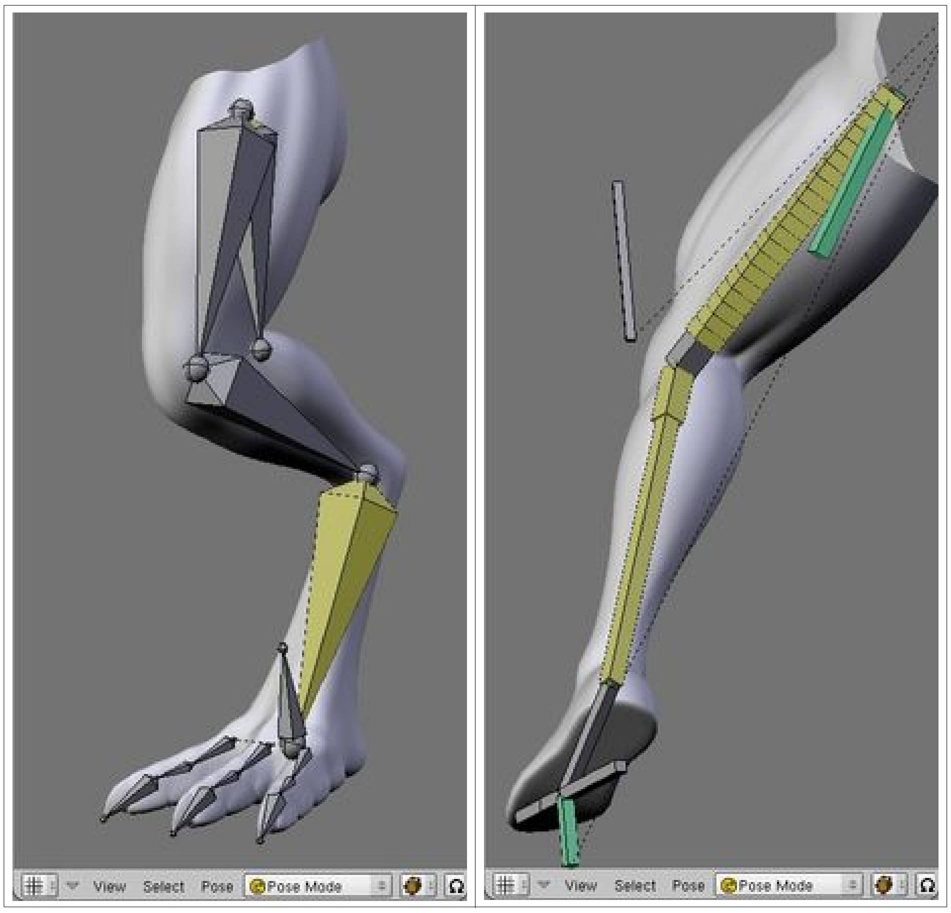
(TO BE WRITTEN)

Hand Rig Design

Hand rigs are mostly just a matter of choosing what method gives you the most amount of control, while still being easy to work with during animation. The rig on this page is taken from the Elephants Dream characters, and is a very good rig for most all puposes.

(TO BE WRITTEN)

Leg Rigs



Leg rigs are very similiar to arm rigs, and in some cases, they are identical. Legs and arms are both limbs, and they both interact with the world around the character, but legs spend a lot more time doing this. As a result, legs are animated using IK almost all of the time. If you have a character--let's say a dancer, and she's going to have lots of arm movements but little interaction with the environment, then FK is the way to go. But if your character is going to walk down a staircase with her hand on the railing, or climb a rock wall, or lift heavy objects, then you want to use IK instead.

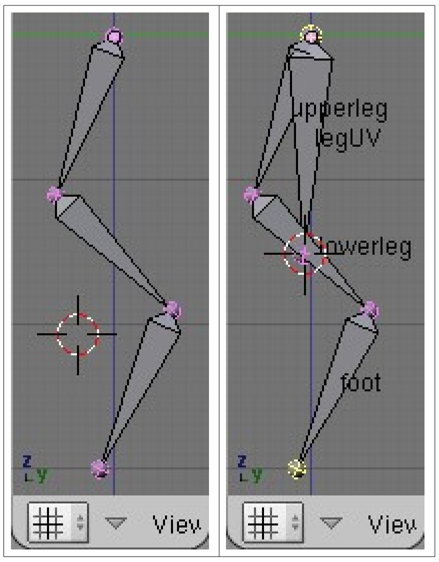
Another factor is that legs have feet (usually), and arms have hands (usually). These are also rather similiar to each other, but still different enough that they can change the way the rig of the leg or arm must be designed.

We're going to look at two rigs in this section. Hopefully building these will teach you enough that you can modify them--or create completely different rigs--to suit your characters' needs.

The Funky Chicken

This is the rig you want to use for characters that have bird-like legs and feet. Examples might be a dinosaur, an ostrich, a kangaroo, or a giant mech sporting the reverse-joint leg design. In the real world (the world outside of a computer), legs like this are an evolutionary change where the foot is elongated and the animal walks on it's toes. In the CG world, we have to treat the ankle as being no different than a knee: it's part of the leg. This means that--for us--the "foot" is where the limb touches the ground.

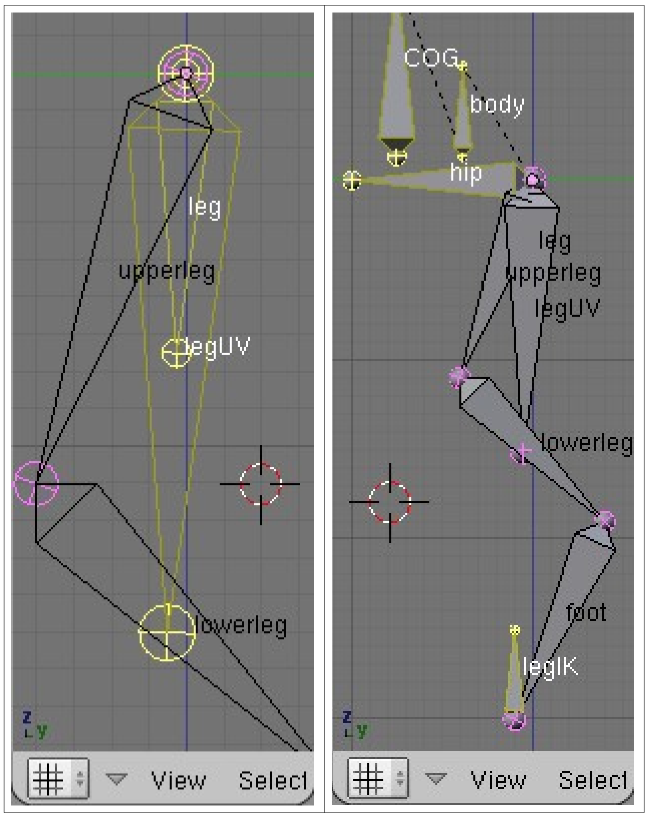
Let's Build It



Start by drawing the three bones of the leg. Draw these to fit your model.

Snap the cursor to the root bone's root point, then add a new bone. Select the tip and root of the chain--as shown--and then snap the cursor to the selection. Then snap the tip of the new bone to the cursor--as shown.

Now is a good time to do some naming. The new bone can be called "legUV". UV is our abbreviation for the term "Up Vector", which means up direction. This bone is a rig element, so turn **Deform** off. Name the other bones "upperleg", "lowerleg", and "foot" (remember, this bone is part of the leg rig, we just call it 'foot' because real birds have foot bones here).



Duplicate legUV, snap the *cursor* to the selection, box-**de**select the root point only, and then snap the *selection* to the cursor. If you do this right, you should have a new bone called legUV.001 that is half as long as leg. Name this new bone "leg".

Snap the cursor to the tip of the chain and add a new bone. This is going to be our IK effector, so name it "legIK".

To show you where these bones fit in the hierarchy of the entire character, I've also added hip, body, and COG bones. Remember, the location of these bones depends on the design of your character. Make your hierarchy like this:

- COG
  - legIK
  - body
    - hip
      - leg
        - legUV
          - upperleg
            - etc...

Note:

Be sure to notice that legIK and body are siblings on the same level

And of course we need constraints to make it do stuff! IK constrain leg to legIK, and set **ChainLen** to 1. Also IK constrain foot to legIK, but set **ChainLen** to 2.

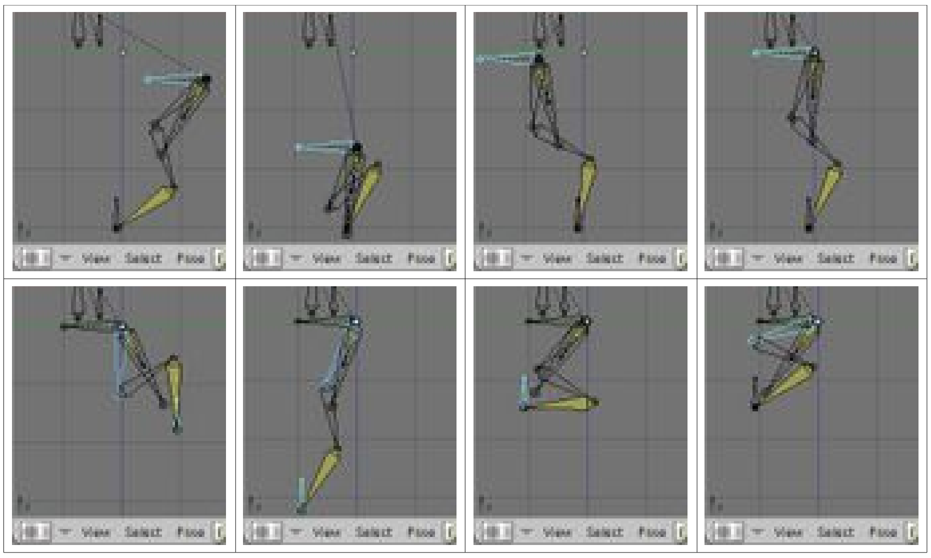
Locking Things Up

Final steps: Lock the appropriate axes of the bones, and organize your layers.

Usage

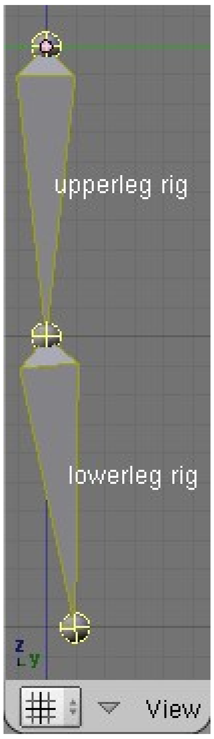
When animating with this rig, you have three primary inputs:

- Foot placement with legIK.
- Knee direction control with legUV.
- Knee flexion and extension by rotating upperleg.



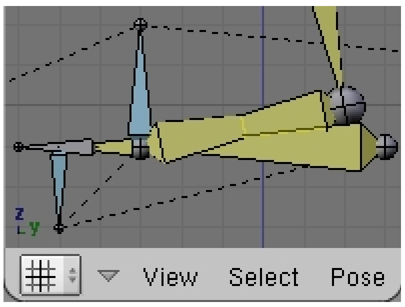
The Not-So-Funky Non-Chicken

page under construction



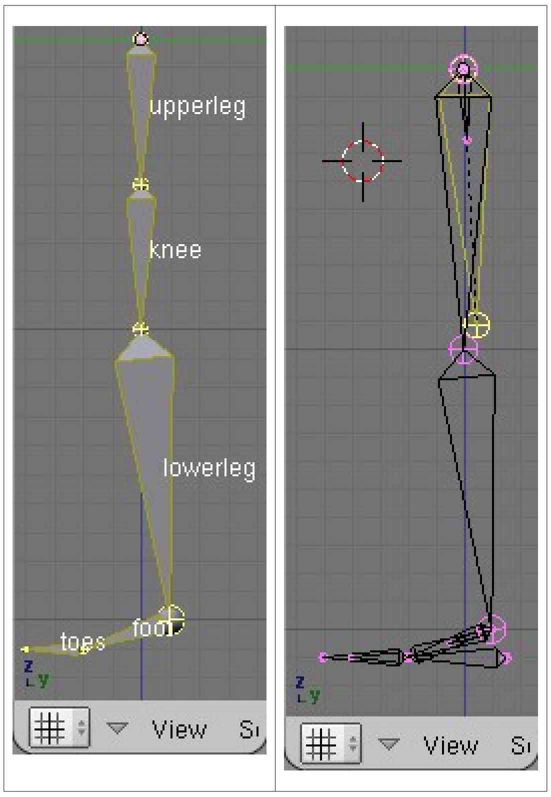
Now we'll look at a rig for human legs.

Draw the bones of the leg as shown, and name them "upperleg rig", "lowerleg rig", "foot", and "toes".



You'll need a foot rig, two of which can be found on the Foot Rigs page (pag. 64). We won't see the foot rig in this tutorial, but you will learn what parts are constrained to a part of the foot rig.

The concept behind this rig is that we can use a simple leg rig that is easy to control and animate, to control a second set of leg bones which does the mesh deforming.



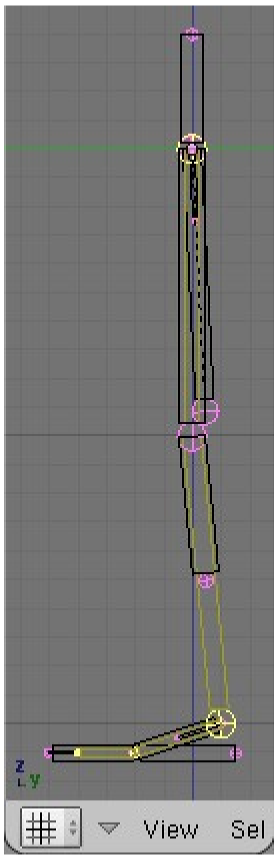
Remember your hierarchy should be like this (right image above):

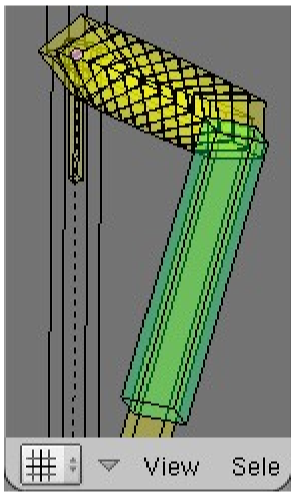
- leg
  - legUV
    - upperleg rig
      - etc

1. IK constrain leg to legIK and set ChainLen: to 1.
2. IK constrain lowerleg to legIK, and set ChainLen: to 2.
3. IK constrain foot to toe rig and set ChainLen: to 1.
4. IK constrain toe to toeIK and set ChainLen: to 1.

Now add a new bone about 0.4 units above the leg root point, and draw some new bones as shown.

These will be "leg null", "upperleg", and "lowerleg". Set upperleg's **Segm** value to about 10 (you can use however many segments you like). Also set its **blend-in** and **blend-out** values to 0. Note that lowerleg is half as long as lowerleg rig. This is because these bones are overlapping and we don't want to make them the same length if they don't need to be.

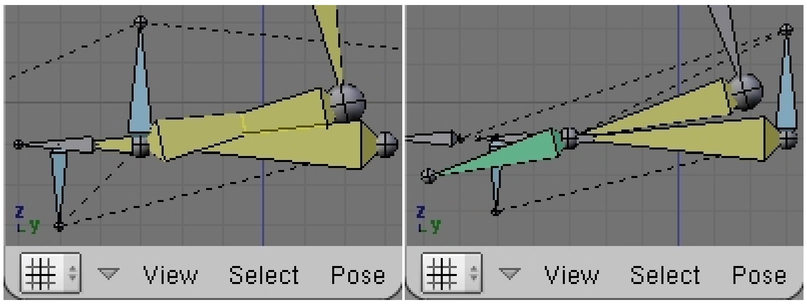




Now is a good time to switch to b-bone mode and use **ALT+S** to resize the b-bones so you can work with them more easily. You don't want overlapping bones to have the same draw size.

IK constrain upperleg to lowerleg rig. Track To constrain lowerleg to foot. Then give lowerleg a Locked Track constraint, use **Z** for the **To:** option, use **Y** for the **Lock:** option, and use legUV as the target.

Foot Rig Design

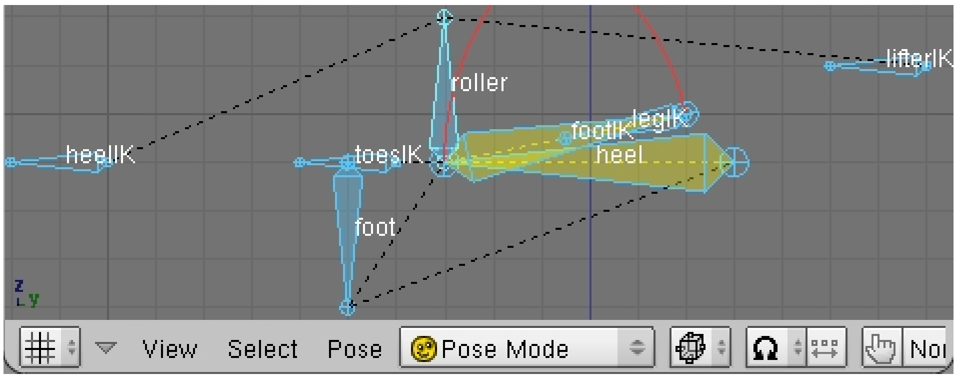


Rig 1 Rig 2

Human-like feet are a tough thing to animate because they can have as many as four different rotation points, they must interact with the ground surface, and on top of all that, the foot bones need to move in a way that doesn't cause the character's knee to experience irregular changes in speed, or even stutter back and forth.

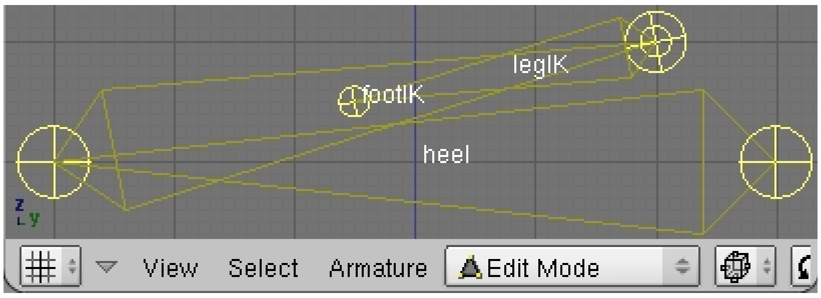
In this section we'll look at two different designs to automate the roll of the foot as it goes from heel to ball to toe. We'll also talk about how this effects the movement of the leg and knee, and what more we might be able to do to make the animator's job easier.

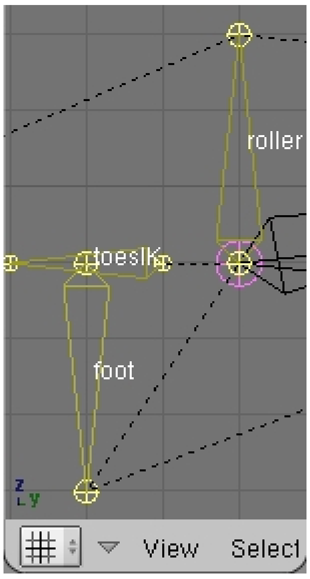
Building Rig 1



The rotations from the ball of the foot and the heel are both driven by IK, and we take advantage of DoF (degrees of freedom) to stop them from rotating down into the floor.

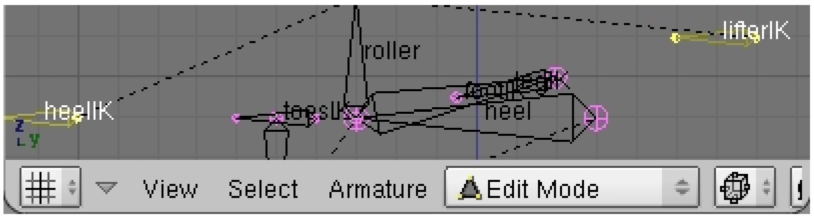
Draw out a three bone chain like this, and name the bones accordingly.





Draw and name these as well.

We're gonna need some IK effectors, so add some bones, place, and name them, like those shown. Be sure that heel is pointing at heelIK and footIK is pointing at lifterIK.



So now we've got all the bones we need, all we have left to do is edit the hierarchy, add some constraints, and set some rotation limits.

IK constrain heel to heelIK and set **ChainLen** to 1. Then do the same with lifter and lifterIK. The lowerleg rig bone of your Leg Rig (pag. 57) should be IK'ed to legIK with a **ChainLen** of 2. Also IK constrain foot to footIK and IK constrain toes to toesIK (both using a **ChainLen** value of 1).

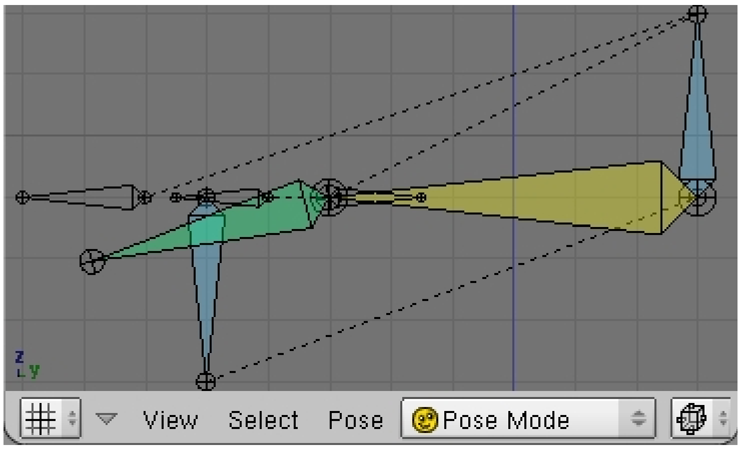
In the **Armature Bones** panel, lock Y and Z and enable X axis limits for both heel and footIK. Set both bones to have the X limit values: **Min:0.0, Max:80.0**
Setup your hierarchy like this:

- foot
  - heel
    - footIK
      - legIK
  - roller
    - heelIK
    - lifterIK

Locking Things Up

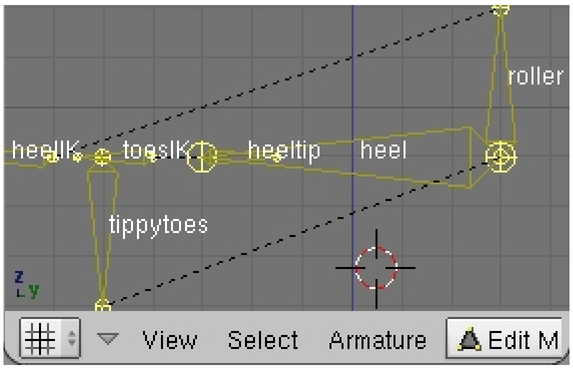
Final steps: Lock the appropriate axes of the bones, and organize your layers.

Building Rig 2

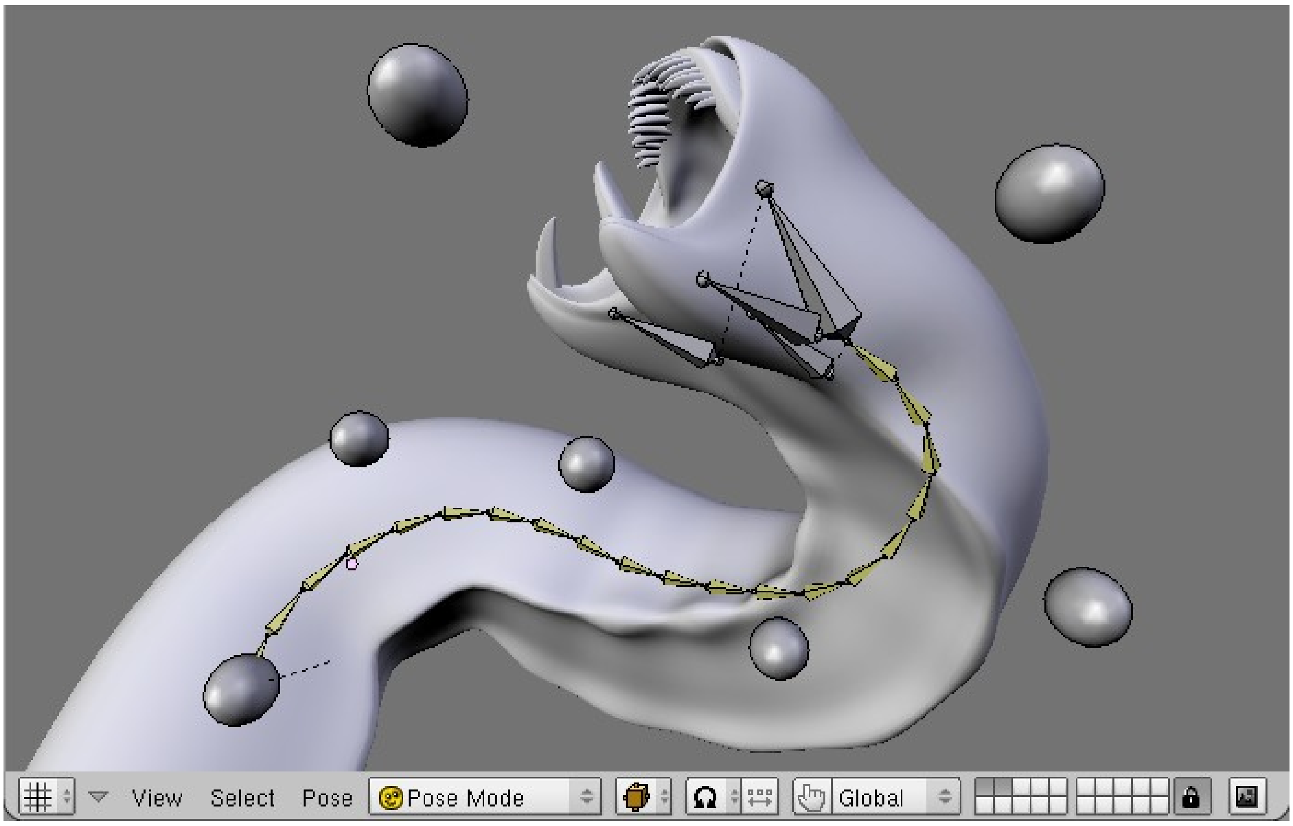


This rig makes use of the IK Solver's target rotation following, which does almost all of the work for us.

Draw some bones like these here crazy thing deals.



Spine Rig Design

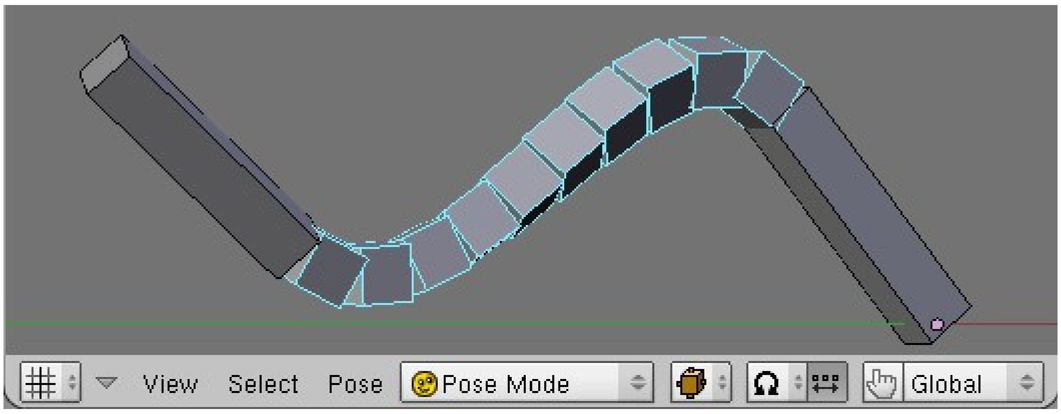


For the purpose of discussing character rigging, we can consider any lengthy chain a spine. This could be the spine of a human, the neck of a giraffe, the tentacle of a giant squid, the trunk of an elephant, the tail of a T-Rex, the spine of a chinese dragon, or even the long serpentine tongue of a certain giant black alien symbiote!

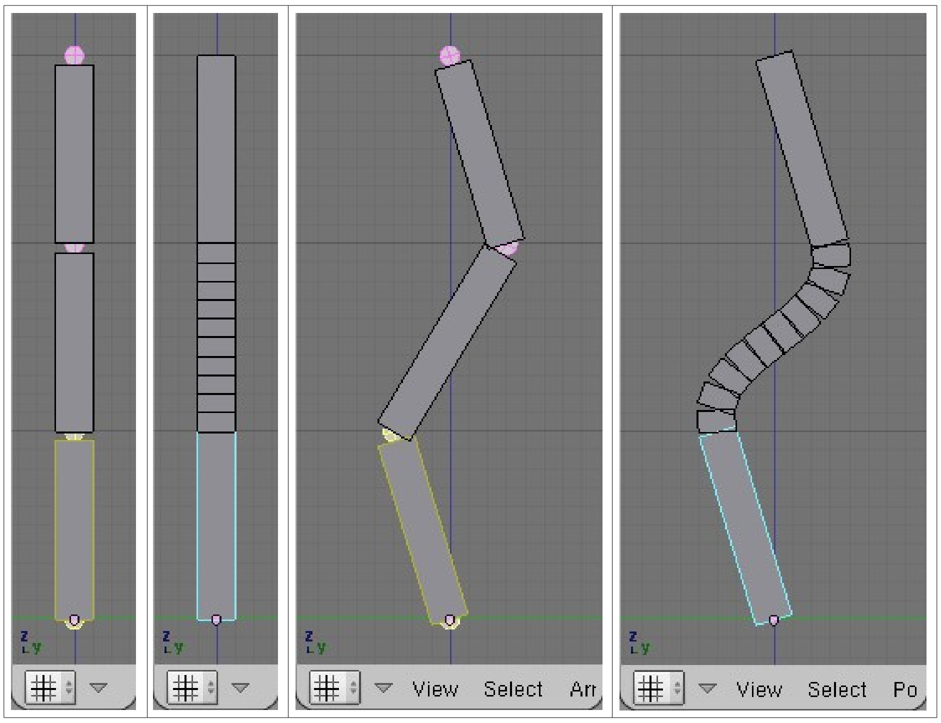
We'll look at a number of different designs and I'll cover how to build them, their uses, and their individual pros and cons.

1. **The B-Bone Spine**
2. **The Propagating Rotations Spine**
3. **The Bend-O-Matic Spine**
4. **The Bones-on-Curve Spine**
5. **The Linear Curve Tracking Spine**
6. **Making A Spine Twist**

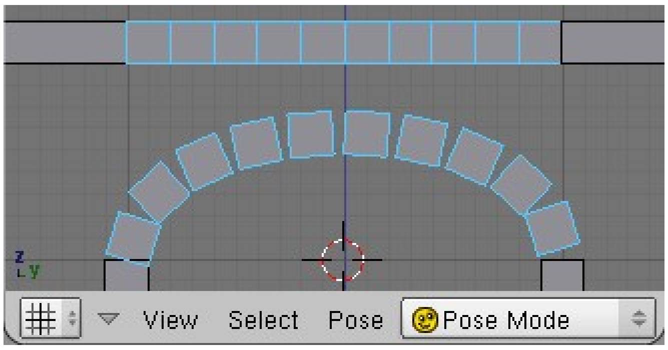
The B-Bone Spine



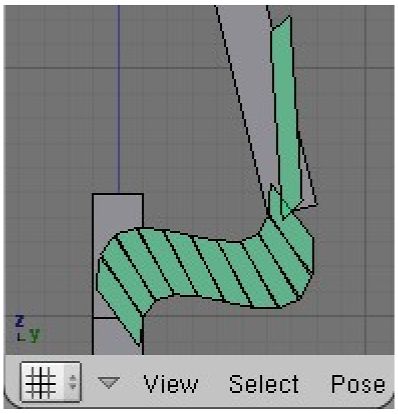
This is probably the easiest way to make something curvy and animated. B-Bones have to be perfectly in line with both the parent and child bones in order to be straight in rest position.



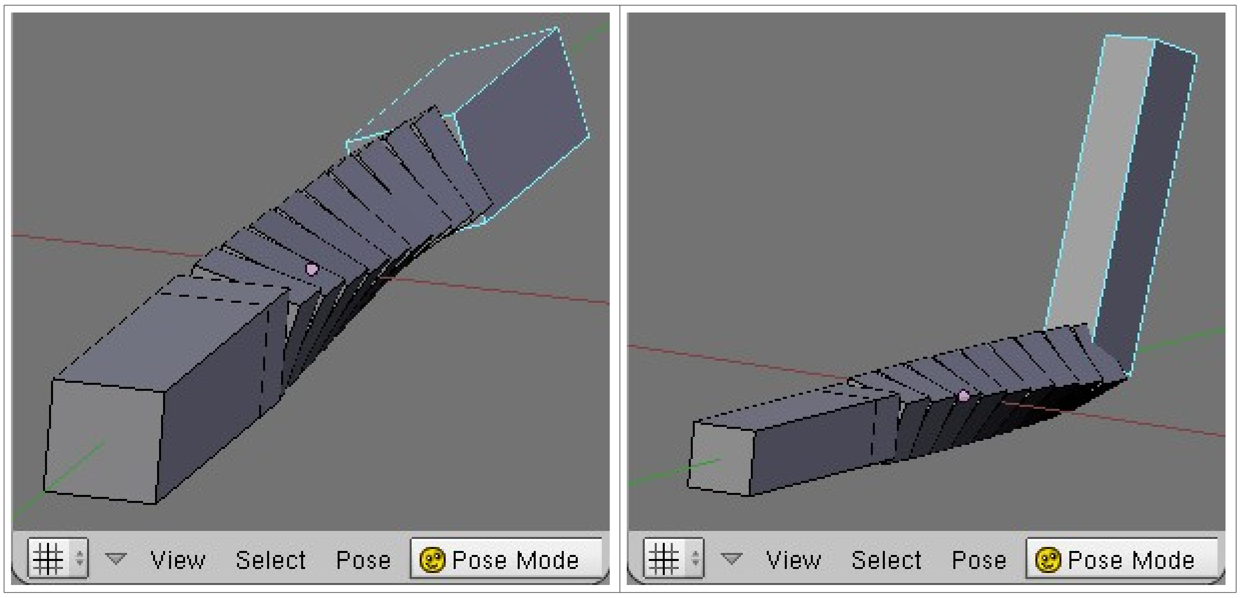
Segmented B-Bones are best used for cartoon-like characters, because b-bones change length as bend angle increases.



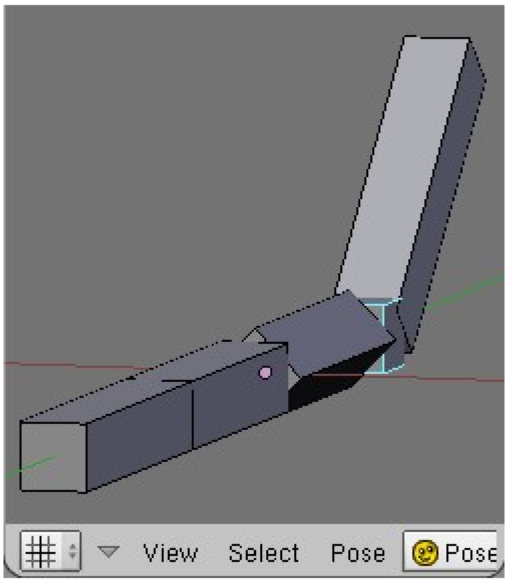
You should never change the scale length of a bent, segmented b-bone. If you do, it will sheer the bone segments and thus sheer the area of the mesh those deforms. Of course, you can do this if that's the effect you want.

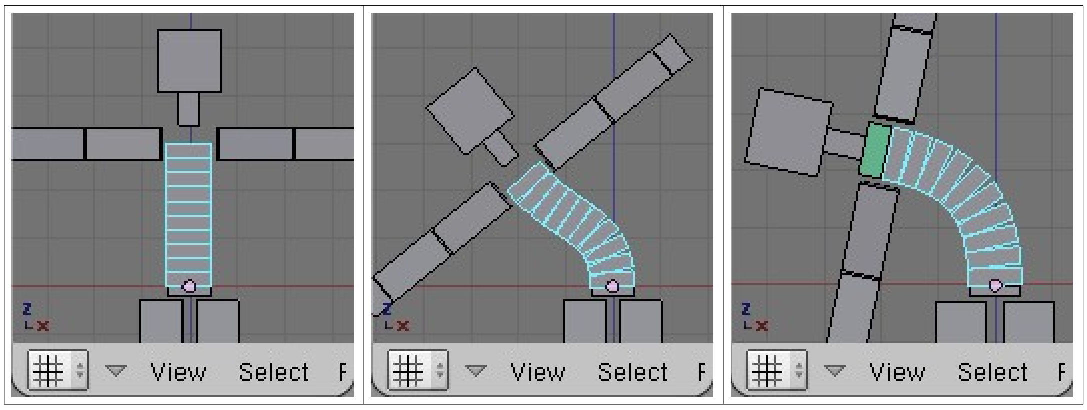


B-bones are very useful in rigs to do things such as twisting. If you set the blend-in and blend-out values to 0, it will cause the bone to remain completely straight, but still twist.



As you can see in the image on the left, the last segment is never quite as far rotated as the child bone. For this reason, it is sometimes a good idea to have an extra bone after the segmented bone.





In the last image of this sequence, you see a small green bone. It's copying the local rotation of it's parent, which is why the b-bone is arched nicely in that image and not in the second image. The arms and neck are children of the green bone. If you parent them to the b-bone, they won't have the orientation of the last segment of the b-bone, and will be as they are in the second image.

This Rig in Review

Now lets consider the benefits and drawbacks of this design.

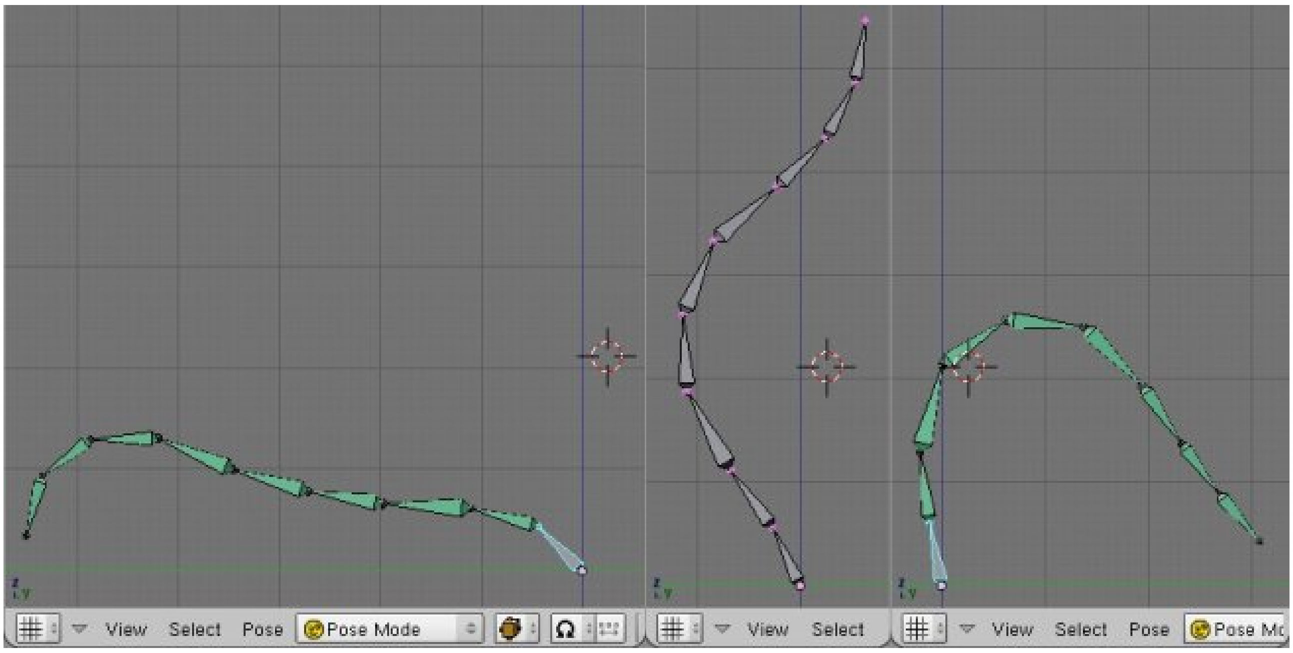
The Pros

- Way easy to use.
- Very useful for twisty areas, like human forearms and upperarms.

The Cons

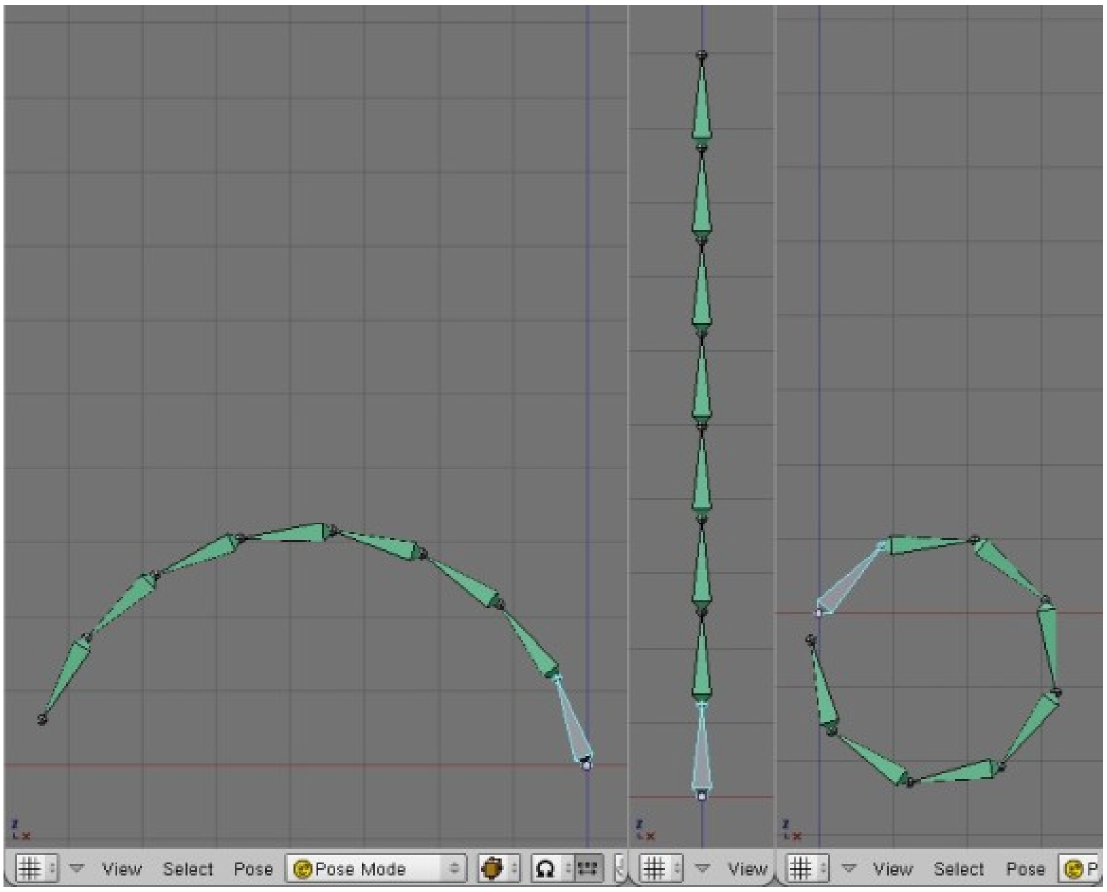
- B-Bones change length when bending and are therefore, not a good choice for realistic character spines.
- Input is limited to two rotations and so the possible curvatures are limited.
- Movement is unnatural because the segments stretch to reach around the rotation points. It's more like a slinky than a spine.

The Propagating Rotations Spine



Here's another really simple design. All you have to do is draw your spine--it can be any shape or size you want--and then give every bone in the chain a copy rotation constraint, using local space, targeting any bone that you want to use as the bending control object.

Remember that **CTRL+C** can be used to copy the constraint from the active bone to all other selected bones.



This Rig in Review

Now lets consider the benefits and drawbacks of this design.

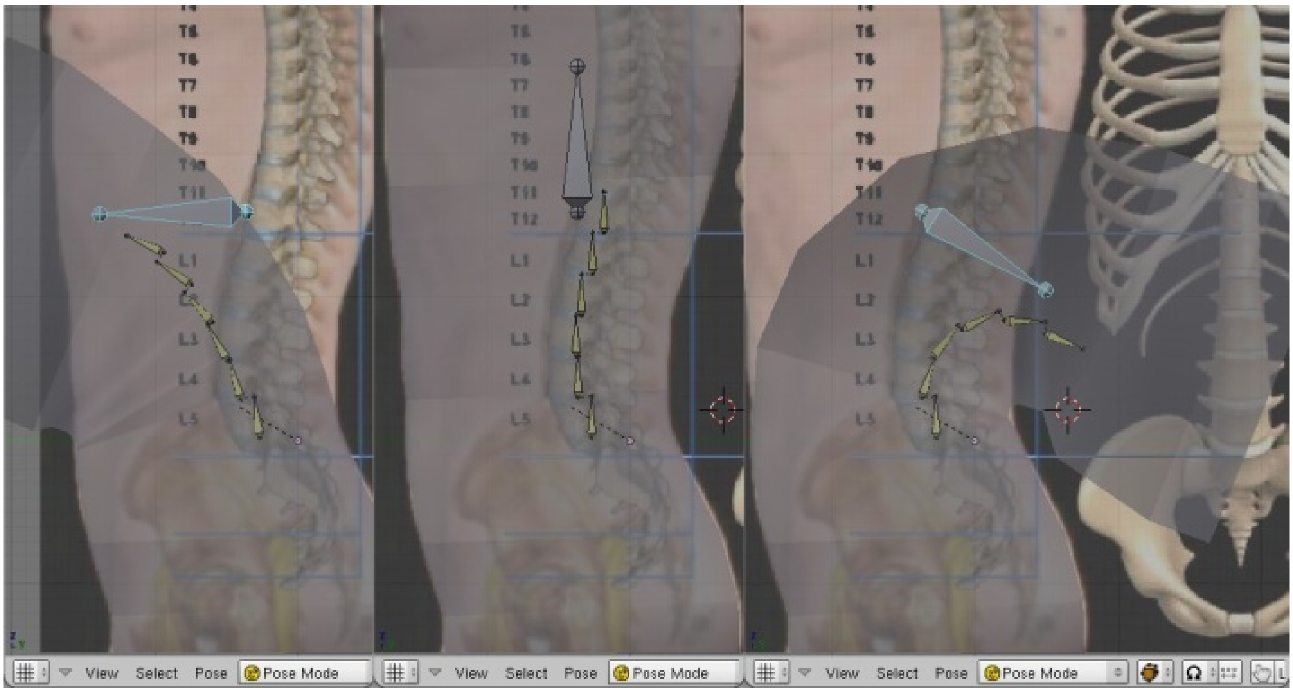
The Pros

- Very quick and easy to make.
- Gives very nice arches and bends.
- Induces twisting, but doing so effects the bent shape.

The Cons

- Inflexible, the shapes it makes are all you get.
- The system is based on FK, so control is very limited.

The Bend-O-Matic Spine

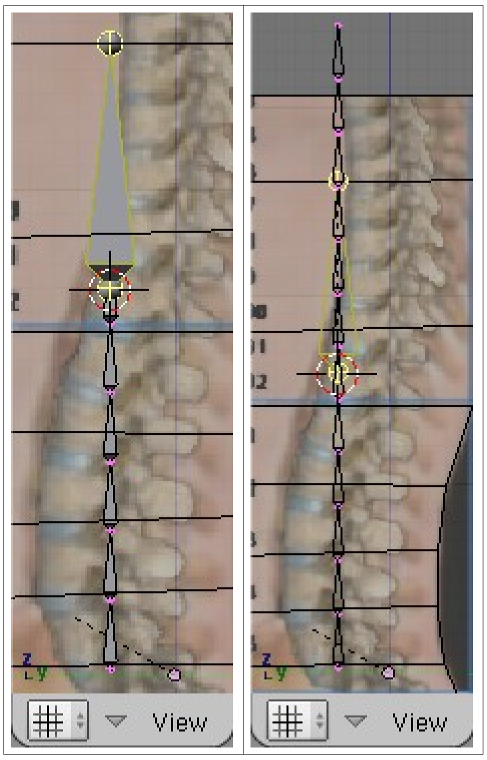


reference image

"Bend-O-Matic Spine" is just a term given by this author to one of his rig designs. This design should work for most athletic human-type characters, but it's probably not quite adequate for some other types of characters, such as a contortionist. This rig should also work good for making a dancer, because this rig has two controls for hip movement that cause the spine to bend, and the upper body to stay relatively still. The control that causes the spine to arch also induces twist as well.

Keep in mind that this particular example is designed specificly for use as a human-like spine, but you could produce many different types of variations on this rig for lots of different uses.

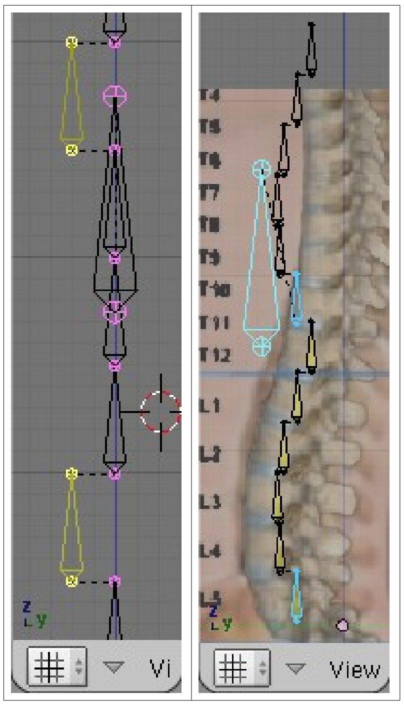
Let's Build It



Start out with a single bone placed near the center of the fist disc and extrending vertically to the disc between L1 and L2. You want the bone to be perfectly vertical. From this bone, we will create all the deform bones for this rig. Subdivide it twice. Name the bones spine1 through spine4, starting at the base. Duplicate spine3 and spine4, and move them up to the end of the chain, and name them spine5 and spine6. Make spine5 a child of spine4, connected (for now). Those are all the deform bones for this rig. Add a new bone at the center of spine6, and name it torso. Now select all of the spine bones, duplicate them, and translate them on the Z axis to the end of the chain. These don't need good names, but if you want to name them anyway, I would suggest using spineIK1 through spineIK6. Make each one of these a child of torso.

IK constrain each spine bone to it's spineIK duplicate. Change the **ChainLen** value to 1 after each constraint is added. Now rotate torso in pose mode and watch her go!

"But my spine is curved!"



The spine is probably useable as it is now, but if you want to give it more of a curve like a human spine, then we need to move the bones. We could move the points of the spine to make it match the arch in the image, but if we do that, our bones won't point straight up, and that means they won't be pointing at our IK targets. We could try moving the targets, but then the arch backwards or forwards will change.

Instead, disconnect all of the bones. Select spine2 and it's target bone, and move them both until spine2 sits about where the center of the disc between L4 and L5 should be. You shouldn't have to move the bone up or down very much to make this adjustment, and the less you do, the better. Now do this with the rest of the bones and IK target pairs, for each corresponding disc of the spine. You should also move the torso bone forward 0.1 or 0.2 Blender units. Doing this changes the shape of the arch, so you might want to do a test:

- Rotate the torso bone forward in pose mode.
- Create a duplicate of the armature.
- Change the position of the torso bone in edit mode for this new armature.
- Reenter pose mode to see the difference between the two arches.

The more that you offset the bones from the originally vertical column, the more effect it will have on the arch that the spine forms when it bends, but arranging the joints this way makes it easier to do even more tweaking of the arch. Here's another experiment to try:

- Rotate the torso bone forward in pose mode.
- Select any of the target bones and translate them on local Y.
- If you decide you want to keep some changes made to the target locations, then make the same changes to edit mode and clear the locations in pose mode.

Let's Make it Do the Twist

I've been told that twisting of the human torso occurs near the middle of the spine (T11 and T12) and not at the bottom (L1 through L5, or the lumbar region). Whether or not you want your rig to be as anatomically correct as possible is completely up to you. You can make the spine twist at any bone you choose. You can also use the torso bone as the target of the rotation constraints. For more in-depth info, let's see the Making A Spine Twist.

This page discusses ways to make any of the four chain spine rigs twistable. Not all of the options here work for each rig, and results may vary for those that are compitable. The propagating rotations spine is different from the latter three because it doesn't use IK tracking. Because of this difference, option 1 doesn't apply for this rig. These options apply for the bend-o-matic rig, but keep in mind that it will behave differently if you offset the bones as described in the latter part of the tutorial.

2. The Propagating Rotations Spine
3. The Bend-O-Matic Spine
4 . The Bones-on-Curve Spine
5 . The Linear Curve Tracking Spine

Option 1

Give all the bones of the spine--except for the root--a copy rotation constraint using local space, and target some bone from outside the spine (a bone you would create as a rig element to control the twist). It would also be good to turn off X and Z in the rotation constraint of each bone, because we only use the Y axis for twisting, but this is not really necessary. I would suggest making the twist control bone a child of the head bone (at the tip end of the chain), and of course you should lock the axes of the control bone so it can only rotate on it's Y axis.

This option is great because you can make the spine twist around many, many times. The reason is because each bone is the child of the one before it. If the first bone rotates 10 degrees, the next bone rotates 10 degrees on top of the previous bone, making for a total of 20 degrees of rotation. If you have a ton of bones in the spine, this method might create too much twist to control easily. If you want the spine to twist slower, then lower the influence value of all the copy rotation constraints to 0.5 or even 0.25, depending on the number of bones in the spine. However, if you do that, your maximum amount of twist will go down. If you want, you can instead hold the Shift key while rotating the control bone to make it rotate slower.

Option 2

This option is a considerable amount of additional work, because it involves writing a script. If you have a very long spine and you're going to animate it being tied into and/or out of knots, then you might want to consider this option.

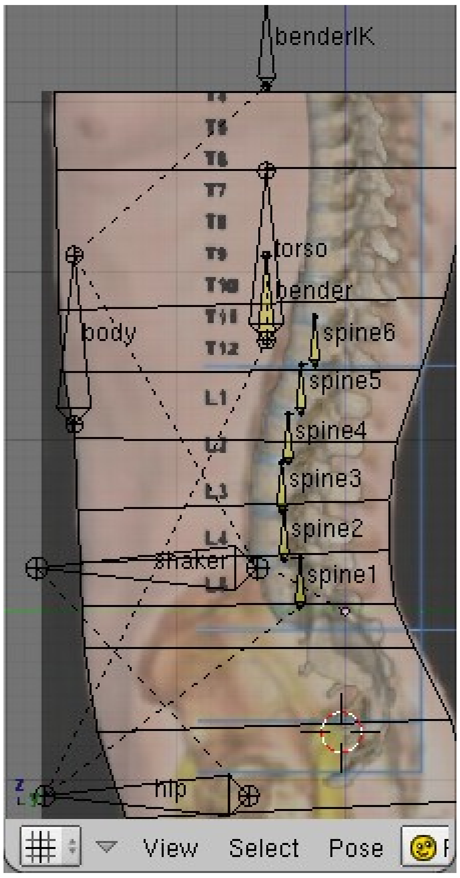
You would need give each bone of the spine a child with an orientation matching it's parent. You would then give the last child bone--at the tip end of the chain, not the root end--a Locked Track constraint. You would also need a new floating bone to act as the target.

Extreme amounts of bending (like tying the spine into a knot) will cause the end of your chain to spin. The only way to make the end of the spine orient itself indipendantly is to use a Locked Track constraint. This will force the end bone to align with a target and not spin out of control. You might need to have this kind of control if you want the spine to interact with it's environment, like picking up objects.

The issue then is making all of the other children bones rotate a portion of what this bone rotates. At the moment, copy rotation constraints can't copy the rotation of a locked track constrained bone, so the only option we have left is to write a python script to do the job. If you decide to take on this task, you'll probably need to use the bone's matrix to get it's rotation result, and you'll need to run the script with a script link.

What you decide to do should be your own decision, based on your own experiments, and what you think works best for your character.

Adding the Hips



What we have made so far is a good rig on it's own, but we can do a little more to make it even better. We can add extra pivot points to make for easy and quick-to-animate hip movements. These extra pivot points go in as additional hierarchy levels between the spine and the body bone.

Duplicate torso, resize to 50%, and name "bender". Duplicate bender and move it straight up (constrained to Z) about 1.5 units and name it something like "benderIK". IK constrain bender to benderIK, and set **ChainLen** to 1. Add a new bone, "hip", and place it over the leg ball sockets in the image. Add and place a new bone, "shaker", half way between hip and torso. Setup the hierarchy like so:

- body
  - benderIK
  - shaker
    - hip
      - bender
        - torso
      - spine1
        - spine2
          - spine3
            - spine4
              - spine5
                - spine6

Note:

To be clear, benderIK and shaker are children of body, and bender and spine1 are children of hip.

Locking Things Up

Final steps: Lock the appropriate axes of the bones, and organize your layers.

This Rig in Review

Now lets consider the benefits and drawbacks of this design.

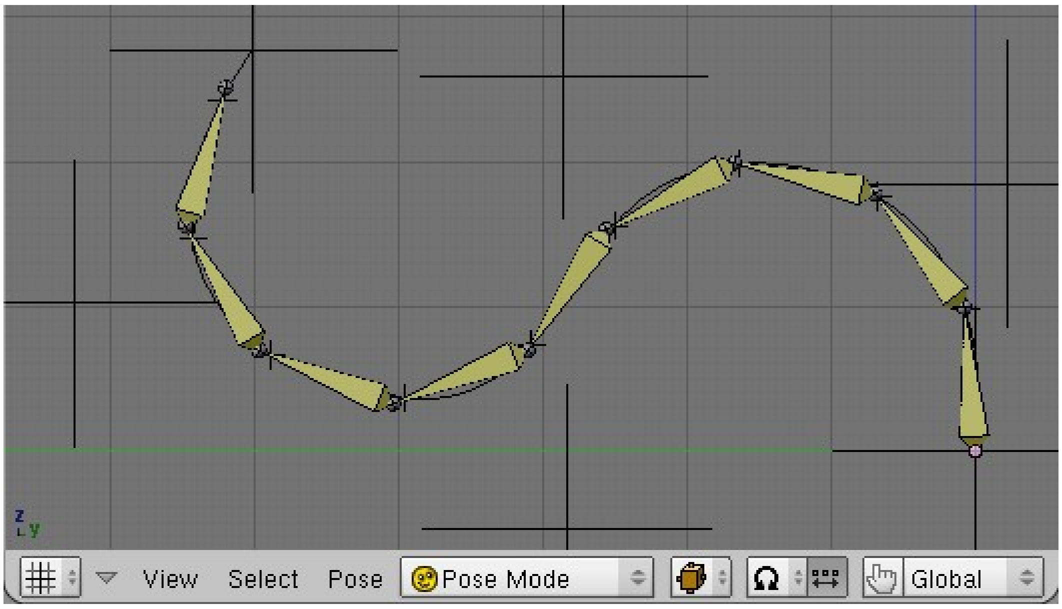
The Pros

- Creates great hip and torso rotation that is very quick and easy to use.
- Induces twisting, not just bending.
- Allows for additional arch shapes during animation by translating or scaling the torso bone.
- Allows for some hip movement while the upper body stays relatively motionless.

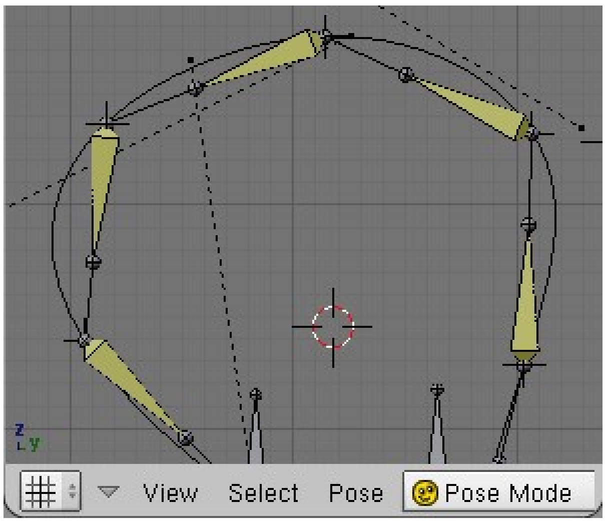
The Cons

- Slightly lengthy setup period.
- Works well for bending, but can't do wavy movements.

The Bones-on-Curve Spine



This is one of the most flexible spine rig designs. You could use this rig to animate a snake, a chinese dragon, a belly dancer, any type of tentacle, etc, etc, etc.



There are two kinds of behavior that we can hope for when using curves in our rigs:

- The rig stretches and squashes to the full length of the curve.
- The rig always stays the same length, no matter how long or short the curve may become.

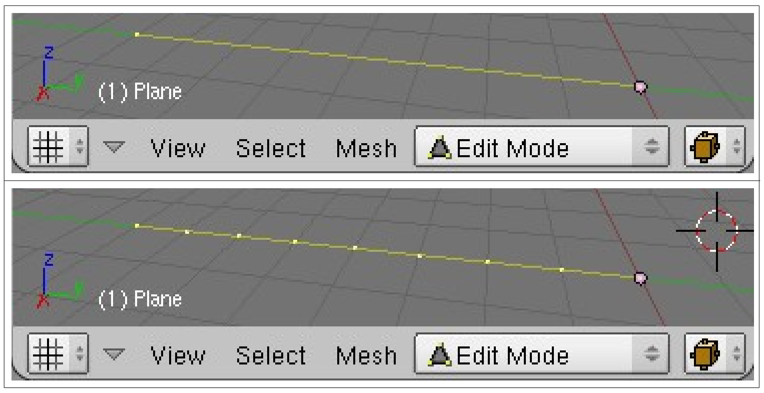
We will build a rig that does the latter. To create a rig more like the former, you simply enable the **CurveStretch** option for the curve object, and location constrain the bones to the appropriate targets.

Let's Build It

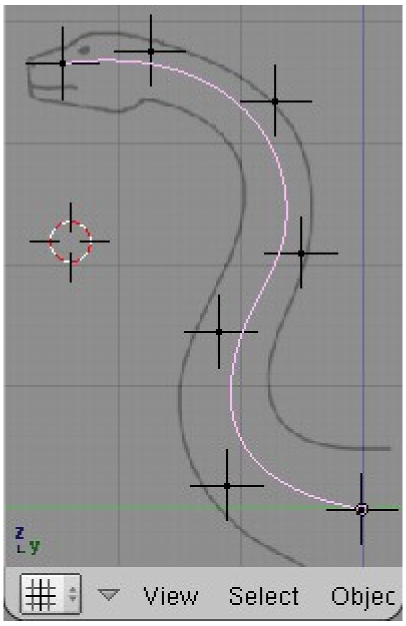
The easy way to build this rig is to use a straight curve and a straight line of bones. If we did that, your character--whatever that may be--would need to have a straight spine. Chances are it doesn't though, so we'll build this rig to fit a character mesh that is already curved.

This rig requires the use of objects outside the armature to make it work. This first subsection is for the curve, mesh, and empties we need to setup before we can draw the bones of the armature.

Meshes and Curves and Empties Oh My!



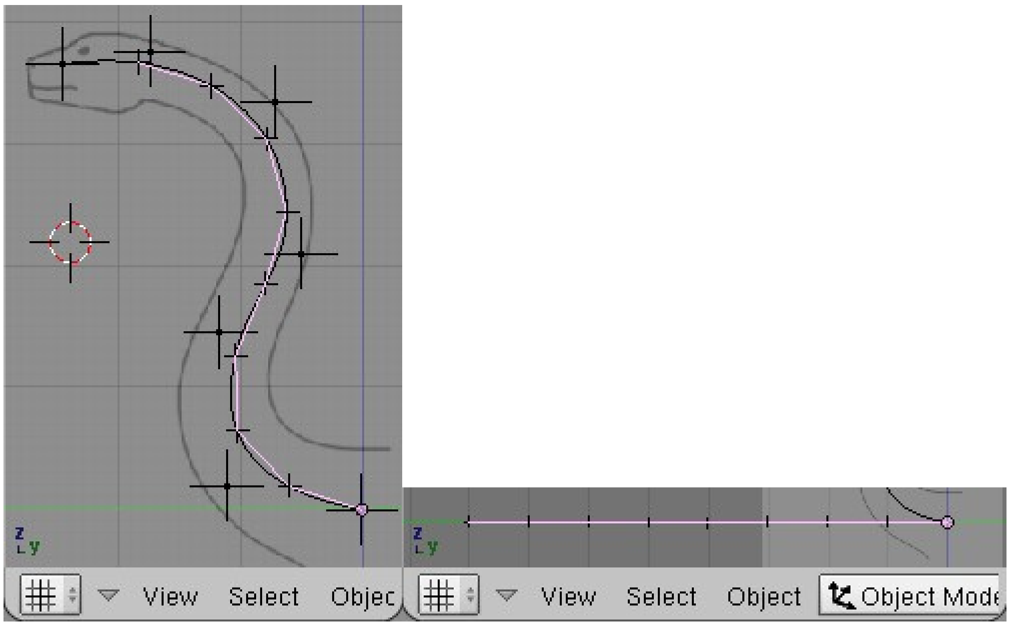
Add a mesh object. You can make it a plane if you want, but it really doesn't matter what mesh primitive you start with. Name it SpineMesh and clear it's rotation. Delete **all** but one of the verts and snap that remaining vert to the world origin. Extrude out 8 units on the Y axis, then subdivide the edge three times.



Now we need a curve. It would be best and easiest to start with a straight curve and build the spine straight, but you can build it to fit a curvy character mesh. For this demo, we'll use a reference image. Add a NURBS Curve (**Add->Curve->NURBS Curve**), clear it's rotation, and name it SpineCurve. In the **Curve and Surface** panel (**F9**), activate the **3D** button, and in the **Curve Tools** panel, press the **Endpoint U** button. Now place your points--and add more as needed--to make your curve fit the spine shape you have. Make sure you don't use too many points.

Give each point of the curve a hook (**CTRL+H**). You might want to change the display mode of the new hook empties to "Plain Axes" (**F9**).

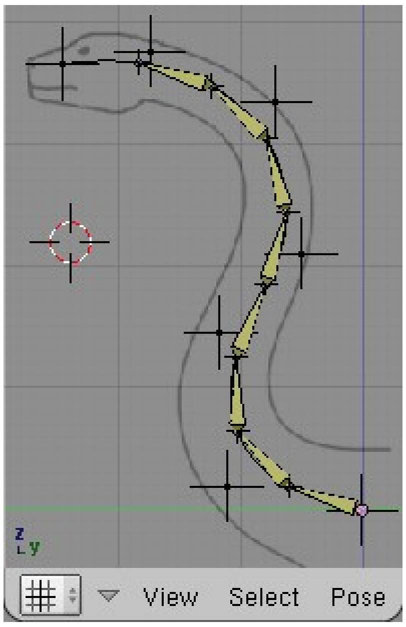
Add an empty. As with the hooks we added to the curve, you might want to change the display mode to "Plain Axes". You'll want this one to be just big enough that it can still be seen when covered by a bone, so scale it down a bit. You can resize these later if the bones need to be bigger than you figured. Duplicate and place one empty for each vert of the SpineMesh object, matching the vert locations exactly.



In edit mode, make each vert of SpineMesh a vertex parent of the corresponding empty (select the vert, then select the empty with **CTRL+RMB**, then press **CTRL+P** to make vertex parent).

Select the SpineMesh object and give it a curve deform modifier (http://mediawiki.blender.org/index.php/Manual/PartII/Modelling/Modifier/Curve_deform), using the SpineCurve object as the deformer. The mesh object will have its base at one end of the curve, and you want that end to be the base of the spine, so you might have to switch the direction of the curve (edit mode, **W**). After that you'll probably need to scale the mesh bigger or smaller until it is the appropriate length. Notice in the image that the last empty is near the base of the skull.

Enter the Skeleton



Now we can add the armature object (remember to clear its rotation first). Extrude and snap the points of the bones to the empties on the curve.

IK constrain each bone to the empty at its tip. You can do this quickly by selecting the empty first and then **SHIFT+RMB** selecting the bone, and then pressing **CTRL+I**. Make sure you change the **ChainLen** value to 1 after each IK assignment. Also, give the first bone of the chain a location constraint targeting the first of the larger empty hook objects. The first of the smaller empties serves no purpose, so you can delete it.

Optionally, you can place floating bones at each hook object, and then make each hook a child of the corresponding floating bone. This makes it possible to include spine animations in armature actions and to perform mirrored or copied posing, but it also causes a cycle in Blender's dependancy graph which results in slightly delayed responce time from the armature spine. When clearing the location of these floating bones, you must clear twice.

Locking Things Up

Final steps: Lock the appropriate axes of the bones, and organize your layers.

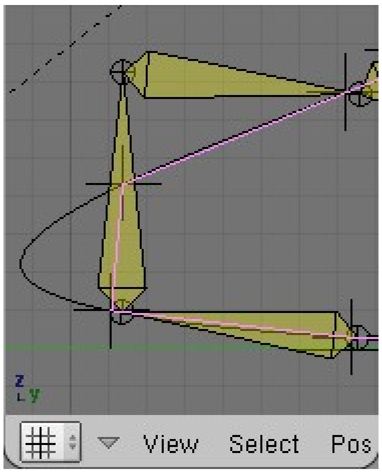
This Rig in Review

Now lets consider the benefits and drawbacks of this design.

The Pros

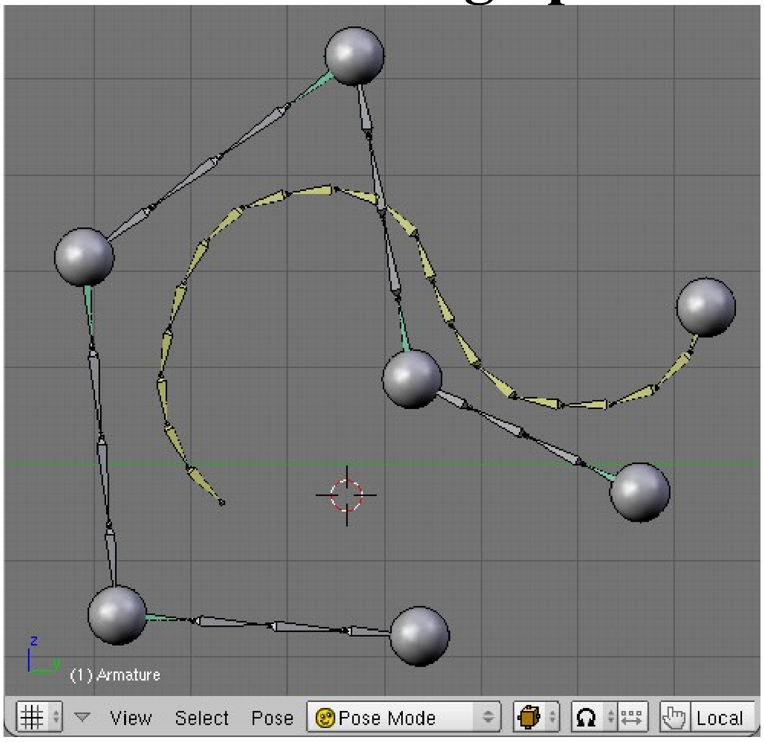
- Can be used for practically any type of movement; bendy, wavy, twisty, etc.
- Makes nice flowing curvy spines easy to animate.
- The design can be expanded to twist anyway you like.

The Cons



- The design requires the use of outside objects, and can have refresh issues.
- Because the curve is not straight between bone joints, the mesh is made shorter than the chain as the curve becomes more and more curvy. This can be seen in rigs with many bones. This limits the maximum number of spine bones the rig can have.

The Linear Curve Tracking Spine



The spheres in this image are bones with a sphere mesh object given as the custom draw type.

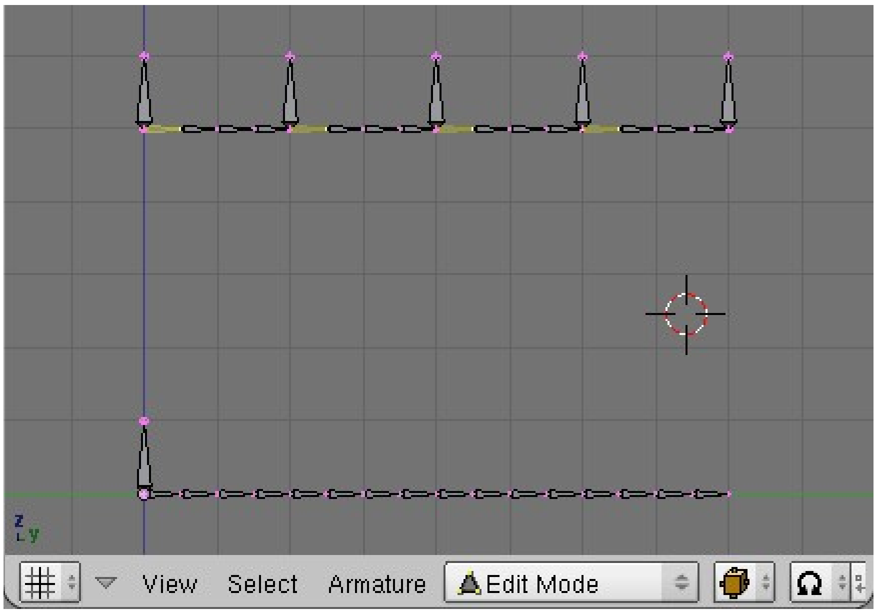
This rig is a great alternative to the bones-on-curve system.

Let's Build It

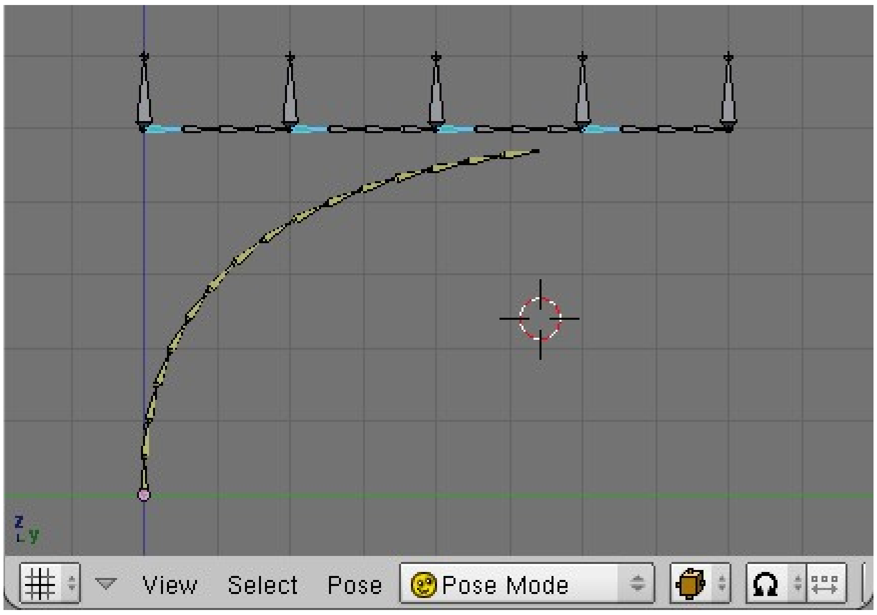
I'm sure you could build this rig in a curved shape to match a curved model, but you're much, much better off saving yourself the trouble and making the model straight to begin with.

Add an armature and clear it's rotation. Make a bone 8 units long, lying on the Y axis. All of the deform bones will come from this bone. Subdivide it four times.

Select all of these bones, duplicate them, and move them 5 units up on the Z axis. These duplicates are rig elements, not deform bones. If you want to give names, name the deform bones spine1 - spine16, and name the rig elements spineIK1 - spineIK16. I will use these names here, but it only matters that you understand which bones I'm refering to.

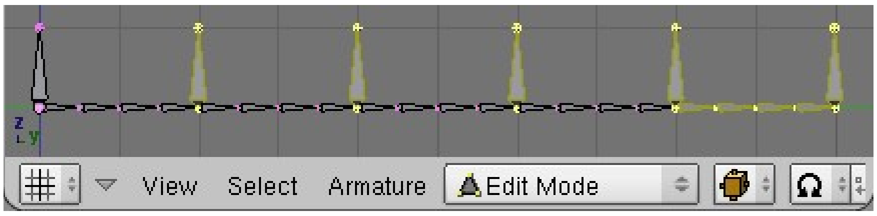


Add a new bone at the root of the lower chain. Name this bone CP1 (for Control Point). Create duplicates and place them as shown. Name these CP2 - CP6. Make each of the shown selected bones the child of the closest CP bone. Also make spine1 the child of CP1.



IK constrain each spine bone to it's spineIK duplicate. Change the IK constraint **ChainLen** value to 1 for each spine bone.

Give each of the shown selected bones Stretch To constraints, targeting the closest CP bone that they point to (e.g. spineIK1 targets CP3).



In edit mode, move all of the shown selected bones down and over, as shown.

Locking Things Up

Final steps: Lock the appropriate axes of the bones, and organize your layers.

This Rig in Review

Now lets consider the benefits and drawbacks of this design.

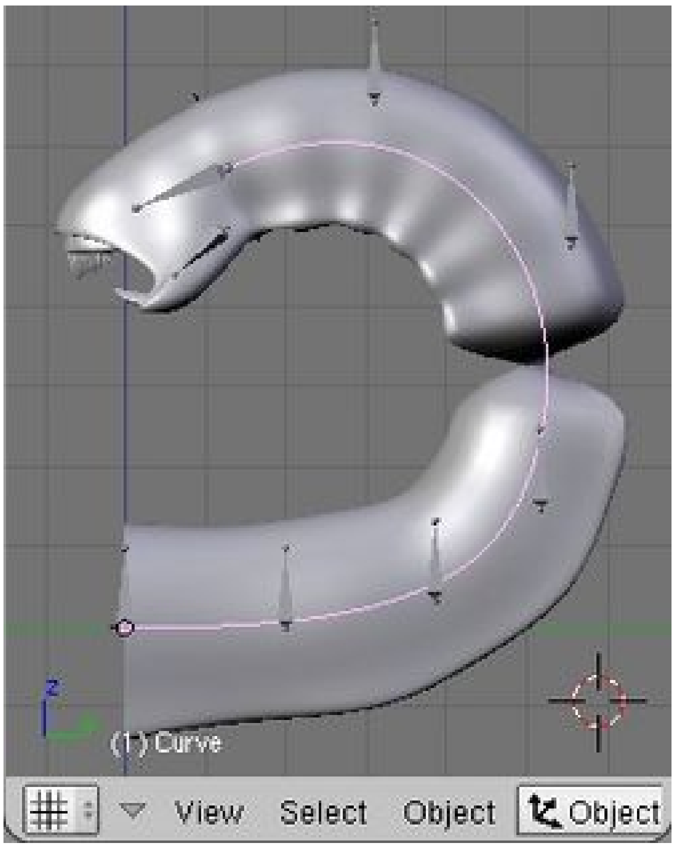
The Pros

- Very flexible.
- Allows for as many or as few spine bones as you need.
- Works without the use of outside objects, unlike bones on a curve.

The Cons

- Control inputs are tied to a specific section of the spine, so the spine is not automatically made smooth.

Spine Rigs vs Curve Deform

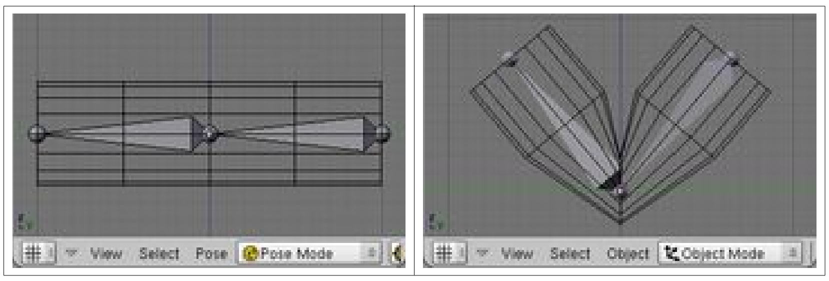


For some characters, it might be more practical to use a Curve Deform Modifier (http://mediawiki.blender.org/index.php/Manual/PartII/Modelling/Modifier/Curve_deform) instead of a spine rig. There are advantages and disadvantages to both spines and curves, and you should understand these so you can make an informed decision over which one to use for your characters.

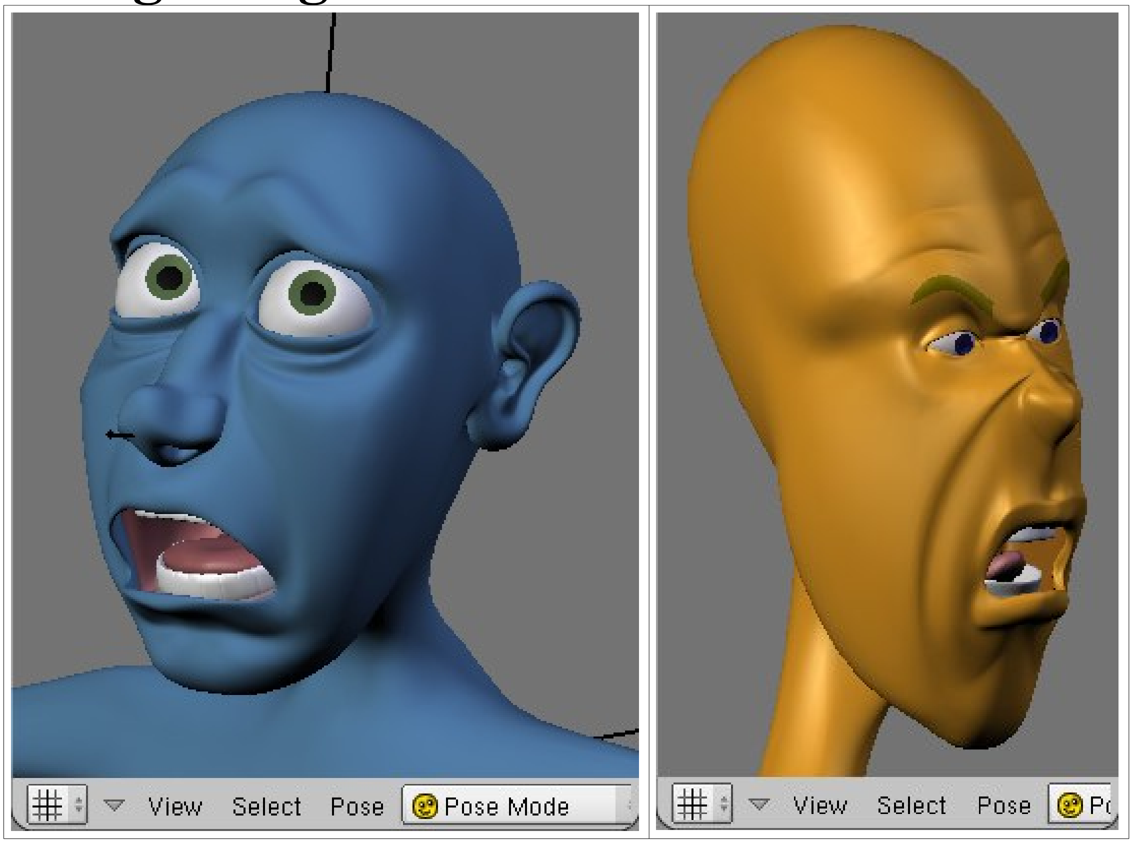
The first issue with curve deform is that it shares the same Z-axis flipping issue as the Track To constraint. The difference is that it uses the curve's local Z-axis as the up/down direction. Curves have to do this because they have to have a way to determine orientation at any given point on the curve.

Curves don't provide a way to easily parent objects on the curve and keep them at a constant distance from one end of the curve. This means adding appendages along the curve would be difficult to manage, at best.

Using a curve will give nicer deformations, especially in sharp corners. The blended weight groups that armatures use compress the mesh around the joint. This is caused by the way that vertex locations are calculated when the influence of one or more bones are blended.



Face Rig Design



I originally did not propose to cover face rigs for the BSoD, so I will finish this page after BSoD is finished.

Widgets and the Wizzbangs

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Dodads and the Dohickies

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

This Rig In Review

Now lets consider the benefits and drawbacks of this design.

The Pros

- It rocks.
- It rocks.
- It rocks.

The Cons

- It sucks.
- It sucks.
- It sucks.

Stride!!

I originally did not propose to cover the stride bone for the BSoD, so I will finish this page after BSoD is finished.

Widgets and the Wizzbangs

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Dodads and the Dohickies

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

This Rig In Review

Now lets consider the benefits and drawbacks of this design.

The Pros

- It rocks.
- It rocks.
- It rocks.

The Cons

- It sucks.
- It sucks.
- It sucks.

the Rig and the Mesh

Rig! Mesh!
Rig Mesh!
RigMesh!

Ya!!!

JointDeformers.blend is in zip file of this offline version.