

"ARM RIGGING FOR ANIMATORS" (STRETCHY CHAIN)

RIGGING TUTORIAL
SOFTWARE: SOFTIMAGE XSI
LEVEL: ADVANCED/EXPERT

Written by Christoph Schinko, 2006

1. INTRODUCTION

When setting up a character, it's easy to dream about all the great things it will be able to do and how simple it will be to control and tweak. Along the rigging process though, many of these ideas are discarded – due to time constraints and/or technical difficulties.

One of the things an animator typically dreams of is the ability to animate squash & stretch, an integral part of classical animation that is often professionally ignored in 3D setups. Of course, the principals of S&S can be implemented in the poses of the character and the behaviour of its extremities, but let's see how the wish list below can be implemented into an arm rig that is functional, fast and easy to control.

Squash & Stretch:

- Ability to over-stretch a skeleton chain and toggle this behaviour on/off
- Ability to reset any stretching without losing the pose (along the eff/root-axis)
- Ability to individually stretch the upper and lower arm or both of them together (eff stays in place)
- according volume changes in the arm and toggle this behaviour on/off
- Manually offset and hand animate those volume changes
- Offset the elbow in any direction (totally independent of the rest of the chain, for final pose adjustments and the ability to "break" the joints as described in "The animator's survival kit")
- Switch from IK to FK (customizing XSIs' excellent FK/IK blending abilities for this rig)
- Quickly lock down the hand in place during animation (for holding onto a table etc.)
- Roll divisions for upper and lower arm for even rotation distribution on the mesh

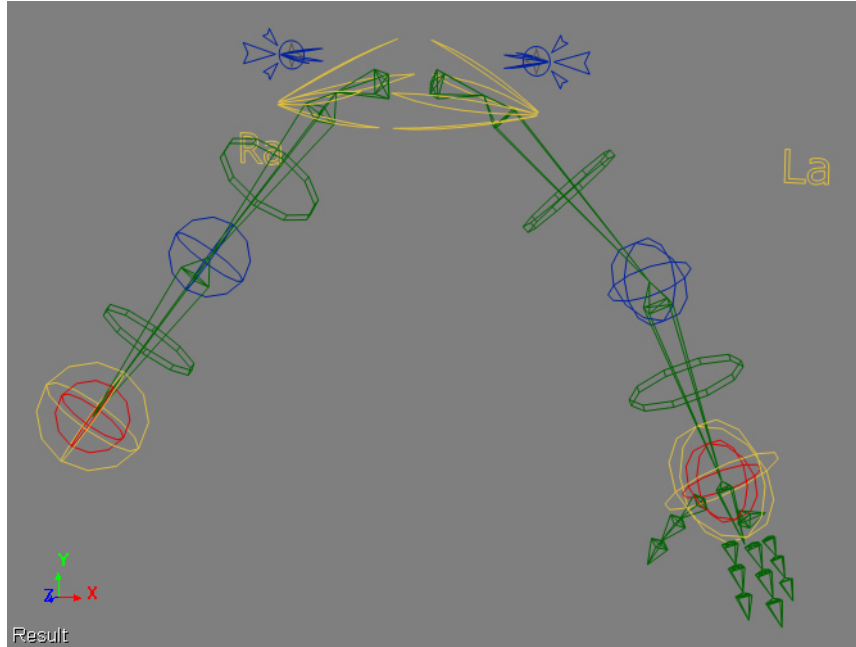


Figure 1: The finished setup

2. STRETCH

1. Create a 2-bone chain in the right viewport and name the root "r_arm", the bones "b_upArm" and "b_loArm", and the effector "e_arm". This naming convention makes it easy to follow this tutorial and allows for quick selection from XSIs' MCP, using "b_*" etc.

2. In the ppg of each bone, set the length of the upper arm to "3" and the lower arm to "4", to be able to work with round numbers (but be able to distinguish between them).

When moving the effector, those lengths always remain the same, until the arm is stretched and the effector cannot be moved away from the root any further.

By changing those length values, the arm can be stretched – and again returned to its original length. These changes will be dependent on the effectors' position, or better: the distance from the effector to the chain root.

This distance can be measured via an expression and further be used on the length parameters of the bones. To keep a better overview, it might be helpful to create a separate object that keeps track of this distance measuring.

3. Create a null (name it "dist") to measure and at the same time display the distance between the eff and root.

4. Set an expression on the local y translation of the null (kine.local.posy), enter the following expression and hit apply.

```
ctr_dist( r_arm.kine.global.pos,  
e_arm.kine.global.pos )
```

When moving the effector now, the null will move along its y-axis (you can move it a bit to the side in z) to the same value as the distance between the root and effector.

5. Apply an expression on the length of the upper arm, telling it that when the distance from the effector to the chain root is bigger than the sum of both original bone lengths (hence, a stretched arm), the bone will gradually be stretched (to meet the effector). The sum of the original bone lengths (3+4 = 7) will be referred to as straight arm distance.

```
cond(dist.kine.local.posy > 7, 3 +  
(dist.kine.local.posy - 7) / 2, 3)
```

This means: When (the distance is > bigger than the straight arm distance (3+4), THEN set the bone length to its original value (3) + HALF of the

difference (as it's 2 bones) between the actual distance (dist) and the straight arm distance ($\text{dist} - (3+4)$).

6. Apply the same expression to the lower arm, taking into account that its original length was 4, not 3

```
cond(dist.kine.local.posy > 7, 4 +
(dist.kine.local.posy - 7) / 2, 4)
```

Now the effector can be pulled from the root as far as you like, and the bones will stretch accordingly and also compress back to their original lengths and behave normally.

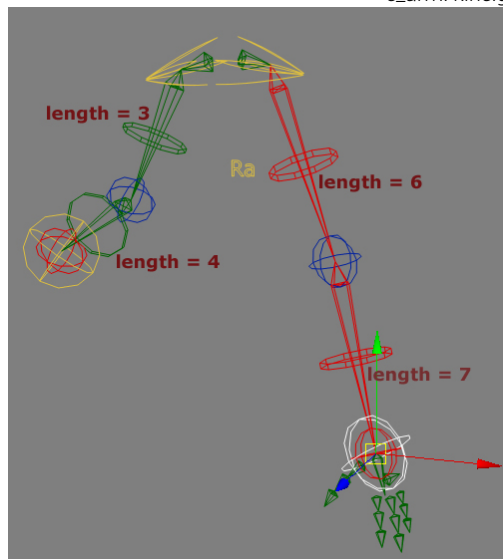


Figure 2: Overstretching the arm chain

When testing this, the need for the following 3 things will immediately arise:

- The ability to lock the chain to behave normally, at its original length,
- The ability to bend the arm while being overstretching,
- And the ability to squash the arm smaller than its original length.

3. RESET AND LOCK

As it might be better to block in any animation with a "locked", fixed-length chain and only animate overstretching later on, the ability to disable the chains stretchiness is important.

Also, when posing a character, it will be nice to be able to reset any accidental over-stretching, making the arm snap back to its original length without losing the pose (along the eff/root-axis). A condition in the length expressions

can take care of that, only allowing a change of bone lengths when a new reset-and-lock parameter is 0. As the bone lengths both rely on the measured distance, it's easiest to go in there to control the bones' behaviour.

7. Create another null ("resetAndLock"), then

8. Change the expression of the "dist"-null to the following

```
cond( resetAndLock.kine.local.posy == 0, ctr_dist( r_arm.kine.global.pos, e_arm.kine.global.pos ), 7 )
```

This will cause the dist-null only to measure the distance between the root and effector when the resetAndLock-null is at 0. At any other value, the distance will be set to "7", the original straight arm distance of the chain – resulting in the same normal chain that was created in the beginning.

4. STRETCH OFFSET

Other than resetting and locking the

stretch of the chain, it might be nice to individually stretch each bone separately, or both at the same time – with the effector locked in place, though. Note that these offsets can also be used to have the chain bend while being overstretching. And they also provide the ability to hand-animate actual squashing of the bones.

This will be controlled by 3 parameters, respectively setting the stretch-offset for the

- Upper arm
- Lower arm
- Both arms at once

9. Create 3 objects to provide those parameters ("stretchUp", "stretch" and "stretchLo").

This time, their scaling will be used to stretch the bones. Their scaling values will simply be added to the length-expressions on both bones.

10. Edit the expression on upArm.bone.length.

Before:

```
cond( dist.kine.local.posy > 7, 3 + (dist.kine.local.posy - 7) / 2, 3 )
```

After:

```
cond( dist.kine.local.posy > 7, 3 + (dist.kine.local.posy - 7) / 2, 3 ) + stretch.kine.local.sclx + stretchUp.kine.local.sclx - 2
```

The "-2" is to compensate for the control objects' initial scaling values (of 1 each), so a scaling of "1" means no offset (0).

11. Do the same for the loArm.bone.length (remember that it's original length was different)

```
cond( dist.kine.local.posy > 7, 4 + (dist.kine.local.posy - 7) / 2, 4 ) + stretch.kine.local.sclx + stretchLo.kine.local.sclx - 2
```

But now the effector will still follow the bones, as it is not constrained to a rig control object yet.

12. Position constrain the effector to a control object ("c_arm").

5. NULLS

By now, there's a bunch of different nulls in the scene, to help with the expressions. Of course, those expressions could refer to different parameters of a single null, or a custom parameter set, with the necessary parameters stored in there (be aware that XSI doesn't like a CustomPSet with certain parameters to be stored under any object of the chain – it will report a dependency cycle).

Also, all expressions could be driven by actual control objects of the character rig, implementing all functionality right into the 3D view,

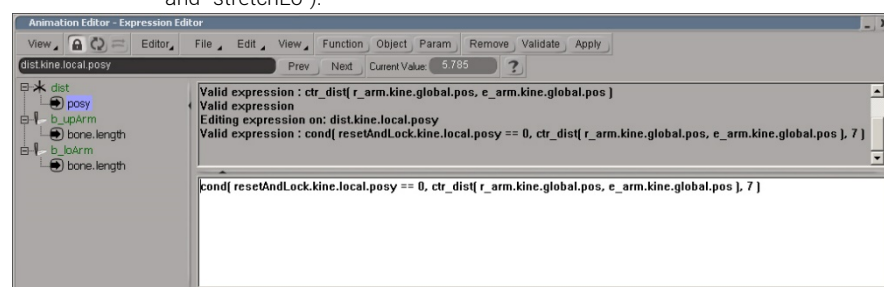


Figure 3: Expression editor

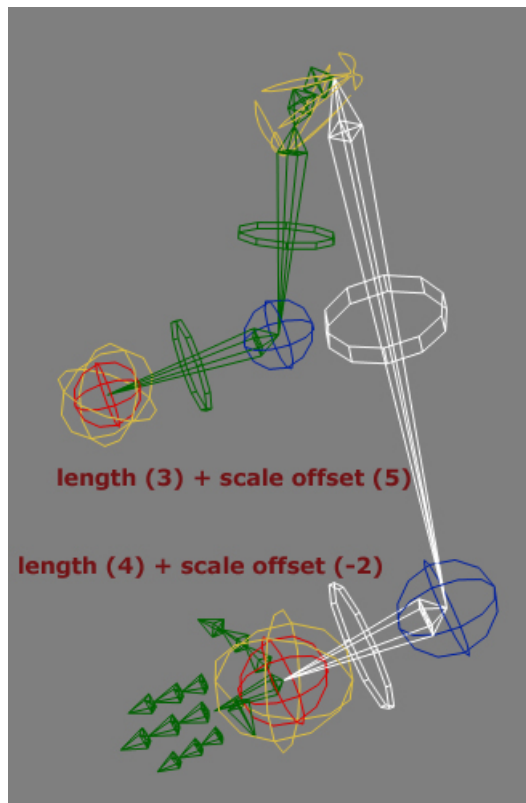


Figure 4: Individual bone stretch offset

which is most desirable for the animator. However, it is advisable to use separate nulls for setting it all up, as they give a good overview and are easy to track down in case of problems or unexpected behaviour.

6. ELBOW OFFSET

Let's move on to a different part of the setup now, a control object that follows the elbow of the arm and can offset it through simple translation. The root and effector stay in place, but the elbow can be freely moved to any desired position in any direction – so this is very different to an upvector constraint.

This feature is useful for perfecting a pose that would look just a bit better if the elbow could be tweaked in some anatomically incorrect way, even if just for a few frames. In the legendary book of Richard Williams, "The Animator's Survival Kit", he often talks about "breaking" a joint to achieve a better-looking animation. With this elbow offset, it can easily be realized in 3D.

To set it up, a second chain will be created to follow the original chain. It will actually be two one-bone chains, rather than a single chain, and will simulate a 2-bone chain that is defined through the elbow offset object.

13. Create a control object for the elbow ("c_elbow") and match its translation with the b_loArm bone.

14. Parent it to that b_loArm bone, so that it always stays at the elbow, and can easily be animated from there.

15. Draw two 1-bone chains, exactly over the two original bones. This is best accomplished using the snapping (to center) tool. Name them offsetUp (r_offsetUp, b_offsetUp,...) and offsetLo.

16. Position constrain the first new bone to the

original root and the elbow control object, and the second new bone to the elbow control object and the original effector. Like this:

- a) r_offsetUp -> r_arm
- b) e_offsetUp -> c_elbow
- c) r_offsetLo -> c_elbow
- d) e_offsetLo -> e_arm

17. Hide the new roots and effectors to get a better overview.

You can see that the elbow offset is already working, and it can easily be reset to 0 by hitting Ctrl+Shift+R. But the new bones need to adjust their lengths properly, as the geometry will later be enveloped to them (to follow the elbow offset). This can be done just like before, using the distance between the elbow control object and the original root, etc.

18. Apply the following expressions

a) b_offsetUp.bone.length:
`ctr_dist(r_arm.kine.global.pos, c_elbow.kine.global.pos)`

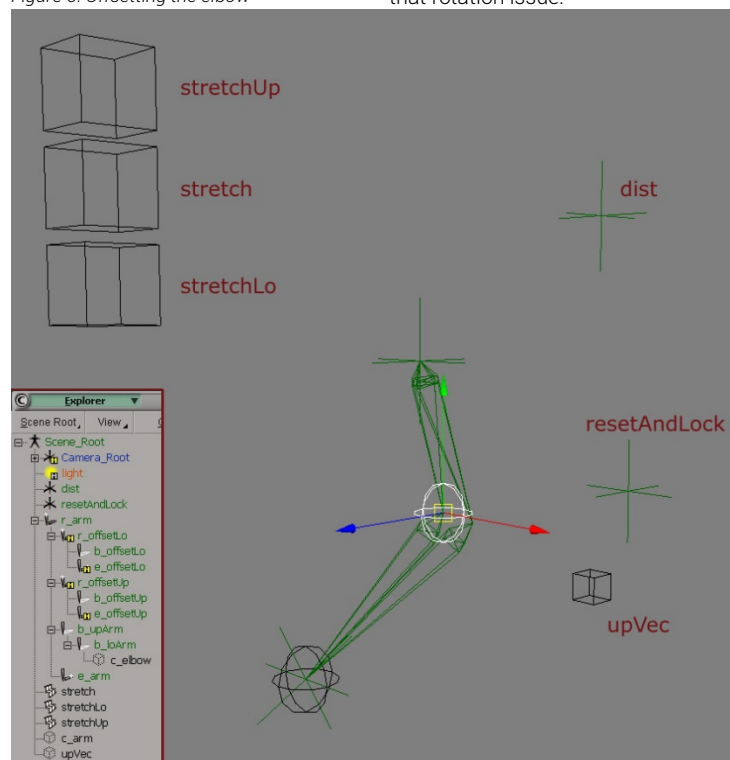
b) b_offsetLo.bone.length:
`ctr_dist(e_arm.kine.global.pos, c_elbow.kine.global.pos)`

That looks a lot better now. But when you set an upVector on our original chain, you'll notice that the new offset bones are not following the x-rotation of the original bones. This is easy to fix for the offsetLo bone.

19. Create an upvector object ("c_upVec"), select b_upArm and set its upvector, using *Create>Skeleton>ChainUpVector*.

Move c_upVec around to see the rotation problem with the offset chains. Instead of position constraining r_offsetLo to c_elbow, use a pose constrain to take care of that rotation issue.

Figure 5: Offsetting the elbow



20. Delete position constrain of `r_offsetLo` and pose constrain it to `c_elbow`

Now it will always have the same rotation as the original lower arm bone. For the `offsetUp` bone it's not that simple, but let's leave it for now, as a more convenient solution will emerge later on in the process. Before moving on, parent the newly created offset chains under the root of the original arm chain, just to clean things up a bit.

21. Drag and drop `r_offsetLo` and `r_offsetUp` under `r_arm`

7. VOLUME

When stretching out the chain now, an enveloped arm would follow and grow longer, but it actually wouldn't retain its volume, it will keep its thickness – unless the y and z axis of the bones are scaled down proportionally to the growing length.

22. Apply an expression to the y scaling of the upper arm – the `b_offsetUp` bone, as the arm will be enveloped to the offset bones. To get an indirect proportional relationship between the length and the scaling, the scaling will be 1 divided by the length of the bone.

```
1 / ( b_offsetUp.bone.length - 2 )
```

When the chain is not overstretched, the bones' scaling should be 1. An outcome of "1 / 1" will do that, so 1 will be divided through the original bone length (3 - 2).

23. Set the same expression on the `ScIz` parameter. Do not just drag and drop the `ScIy` parameter onto the `ScIz` parameter for XSI will report a dependency cycle.

Now the upper arm retains its volume when overstretched. When animating, you will find though, that it's not always nice to have "proper" volume scaling. It might look much better without it, or cheated manually.

24. Create a new null ("volScale") and use its local `PosY` to turn the volumic scaling on or off.

25. Change the `ScIy` expression of the upper arm to turn off volume scaling by setting the `volScale` null to `y=0`.

```
cond( volScale.kine.local.posy == 0, 1, 1 / ( b_offsetUp.bone.length - 2 ) )
```

26. Use the x-translation of that same `volScale` null to offset the volume scaling manually, and be able to cheat, cheat, cheat.

```
cond( volScale.kine.local.posy == 0, 1, 1 / ( b_offsetUp.bone.length - 2 ) ) + volScale.kine.local.posx )
```

27. Apply the same changes to the `ScIz` expression.

28. Do the same procedure for the lower arm `b_offsetLo` (`ScIy` and `ScIz`)

```
cond( volScale.kine.local.posy == 0, 1, 1 / ( b_offsetLo.bone.length - 3 ) ) + volScale.kine.local.posx )
```

8. WRIST SETUP AND CONTROL

Let's quickly setup a wrist chain and think about how it is best to be controlled for animation. There are two basic principles: either the wrist rotation follows the arm effectors', so the hand is always in line with the lower arm bone, when moving the arm around. Or the wrist rotation remains fixed in global space when moving around the arm, meaning it will always point in the same global direction and not follow the lower arm in rotation.

Both methods have their advantages and disadvantages. When posing a character, or animating a swing of the arm, it is more natural if the wrist follows the lower arm rotation, and any additional animation will be easy to control by f-curves, as we know its 0,0,0 rotation always points straight away from the arm.

However, this behaviour is not so welcome when the arm is animated and its upvector adjusted later on (the whole hand animation would change with it) or if there's interaction with another scene object (when lifting an object straight up for example).

Therefore, to get the best of both worlds, let's try and set up a wrist control that can to both, and the animator can switch back and forth between the two behaviours.

29. Draw a 1-bone wrist chain in the right viewport and name the new objects `r_wrist`, `b_wrist` and `e_wrist`

30. Match the rotation of "`r_wrist`" to the original effector "`e_arm`", and zero out the rotations of the wrist BONE "`b_wrist`", so that its z-rotation will rotate the hand towards and away from the body, with the arm in its natural resting pose.

31. Copy the `c_arm` control object ("`c_wrist`") and set it a bit smaller. It will control the rotation of the hand.

32. Create a null ("`wristBuffer`"), match its translation to the "`c_wrist`" object and its rotation to "`e_arm`".

33. Parent "`c_wrist`" to "`wristBuffer`" and zero out its rotations.

34. Orientation constrain the wrist bone "`b_wrist`" to the wrist control "`c_wrist`".

35. Position constrain "`r_wrist`" to "`e_arm`"

36. Position constrain "`wristBuffer`" to the arms effector "`e_arm`"

37. Now constrain the orientation of the "`wristBuffer`" null to "`e_arm`" and lock the constraints ppg.

Its active-flag will be used to switch between the two behaviours – either the control objects (and therefore the wrist) will follow the effectors orientation or not.

38. Link the activeness of the constraint to the `ScIX` parameter of "`c_wrist`", to be able to switch behaviours right from inside the viewport

```
cond( c_wrist.kine.local.scIx < 1, 0, 1 )
```

Play around with it to see if this setup suits you. It's quite comfortable to quickly pose the character with the wrist following the arm and be able to switch this following off and still keep the f-curves simple, local to the effectors rotation.

Note: Be aware that the wrist could cause gimbal lock during animation. You can try different rotation orders within the kinematic properties of the wrist bone (YZX works quite well for me), or follow other tutorials on how to get around gimbal lock – like the common approach to split up the rotation axis onto different objects.

9. ARM & HAND LOCK DOWN

Another thing that is always needed during animation is the ability to quickly lock the hand in place, when holding on to a table for instance. This is easily realized by position constraining the arm control and orientation constraining the wrist control.

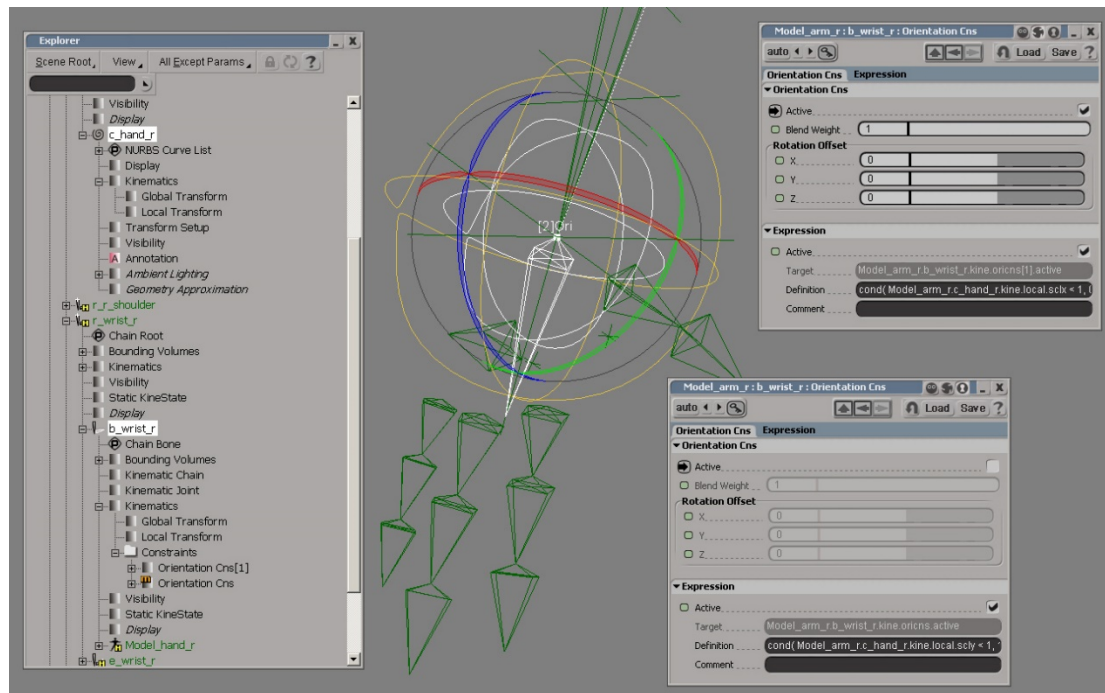


Figure 6: Wrist setup

39. Create an object that the arm and hand will snap and be locked to ("handSnap"). It should give a visual cue as to which way the hand will point when constrained.

40. Save a translation key on "c_arm"

41. Position constrain "c_arm" to "handSnap"

42. Set an expression on the "active" parameter of the position constrain

`cond(c_arm.kine.local.sclY < 1, 1, 0)`

This will use the scaling of the control object to switch behaviours. When "c_arm" is scaled down in y, the arm snaps to the "handSnap" object and stays locked there. Let's do the same for the orientation of the hand.

43. Orientation constrain "c_wrist" to "handSnap"

44. Set the same expression on the "active" parameter of the orientation constrain

`cond(c_wrist.kine.local.sclY < 1, 1, 0)`

By animating the scaling, the two constraints can be set independently of one another, depending on the situation at hand. Also, the blending slider of the constraints are still available for animation, enabling the animator to blend these switches over time if necessary.

A convenient resting place for the "handSnap" object should be found – one where it's not in the way, but easily accessible when animating. It could follow the overall arm root, together with other objects, that control many of the parameters we've set up so far (see next paragraph).

10. VISUAL FEEDBACK

There's a lot of functionality and different options in this rig right now, which might be confusing for the animator. Therefore, it is advisable to make things a bit clearer, provide information and possibly visual feedback about what's going on right now.

Add annotations to all control objects that have special functionality, for example when scaling it in y.

45. Select "c_arm" and apply *Get>Property>Annotation*

46. Write, for example "sclY < 1 activates POSITION constraint to handSnap object". Do the same thing for all control objects that might be confusing or unclear. The annotation property can be brought up at any time when the object is selected, just hit "Selection" in the MCP or look in selection-mode in the explorer – you'll find the reddish annotation icon.

Another easy way of clearing things up is by changing the colour of control icons or bones when stretched. This is done by setting an

expression on the display colour of each object, and telling it that, "when bigger than 1, turn red", for example.

47. Select "b_upArm" and isolate it in the explorer. Notice how its "Display" settings are spelled in italic, that means that they're shared with the rest of the scene and need to be made local to this very object before changing anything.

48. Click on the "Display"-icon and hit "Yes" (make local) in the dialog box. Now, "Display" is not written in italic anymore.

49. Expand the display options in the explorer and set an expression on the "colour" property

`cond(b_upArm.bone.length > 3, 15, 0)`

"15" is a nice bright red, so every time the bone is longer than its original length (3), its wireframe display colour will switch to red. This would also be good for the stretch offset control objects, to be able to see if there's a stretch offset applied or not. Here, it might even be better to make it turned red when it's bigger OR smaller than 1 ("!=" is the expression command for "not equal to" – look under "Function>Conditions" in the expression editor for more). Maybe even set it up with two conditions: turn red when bigger than 1.1 or smaller than 0.9, to have a bit of room in between – that way it is easier to find back to zero when not working numerically.

50. For the elbow offset bones it is nice to be visible only when the elbow is actually offset. Set an expression on their view visibility,

```
cond( c_offset_Larm.kine.local.posx == 0 &&
c_offset_Larm.kine.local.posy == 0
&& c_offset_Larm.kine.local.posz == 0, 0, 1 )
```

and maybe set their colour to a bright yellow ("126") to always be able to spot them (check the history of the script editor to find out the exact colour-values you're setting).

Last but not least, when the rig is working the way you want it to, simplify it by slowly getting rid of all those helper-nulls and replacing them with more visual controls, implemented into the rig. It might take a while to edit all expressions to work with the new objects, but it pays off as soon as you start animating!

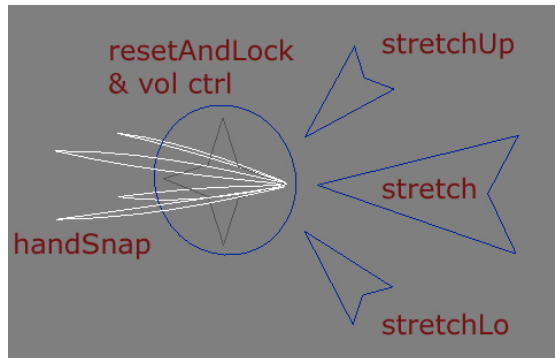


Figure 7: Chain option control icons

11. FK/IK (forward kinematics / inverse kinematics)

Another important decision when animating arms especially is whether to use FK or IK. Now, this arm has obviously been set up for IK, but with Softimage XSI's excellent implementation of FK/IK blending, it would be a shame not to adjust our rig to be able to take advantage of these features.

XSI's FK/IK controls can be found in the "Kinematic Chain Properties" under the FIRST bone of each chain. The "FK/IK blend" slider allows to effectively blend between the two animation methods.

The only thing preventing our arm to be animated in FK right now, is the position constraint of its effector to the c_arm control object. However, if we drive the activeness of this constraint with an expression, it can be de-activated by, for example, scaling our c_arm object down, and

at the same time drive the FK/IK blend property to tell XSI that it's an FK chain now.

51. Set an expression on the blending of the "e_arm" effectors' position constraint

```
c_arm.kine.local.sclx
```

52. Set the same expression on the FK/IK blend property of the chains' Kinematic Chain properties.

Now, whenever the c_arm control object is scaled down in x, XSI will blend from an IK to an FK chain, and the effectiveness of the effectors' position constraint will proportionally grow weaker, until – at a scaling of x=0 – the system will be forward kinematics.

Simply animate the scaling to blend from IK to FK and vice versa!

The scaling of the control icon at the same time visualizes the behaviour, but let's also activate shadow icons for the arm bones, so that they can be grabbed easily when animating in FK mode (these can be driven by

expressions as well, to only "appear" when in FK mode).

12. WEIGHTING HELPERS

To finish the arm rig off, some weighting helpers will be added. Usually, it can be quite tedious to properly weight the lower arm to show a nice roll when the wrist is rotated in x. This is due to the nature of the enveloping algorithms that do not interpolate rotations within one enveloped bone. There's numerous approaches on how to overcome this problem, from a 2nd bone along or the opposite way to the lower arm bone, to well thought out, anatomically correct rigs of the lower arm structures. However, in this tutorial, helper-objects will be positioned along the lower arm, to dampen the wrist rotation towards the elbow.

The same problem applies for the upper arm and its behaviour when it is rotated in x. The part close to the shoulder stays pretty much in place, but down the upper arm the flesh rotates successively more with the

bone underneath. The same approach can be utilized here.

Those helper nulls are usually called "roll divisions", and can be created automatically with the rigs that are implemented in XSI. They're really cool, by the way, all set up with super-fast scripted operators and what not, so be sure check them out! Now, however, we will manually set up them roll divisions (yes, I've lived in the UK for a while), if only to deepen our understanding and come closer to rigging-nirvana.

THE LOWER ARM

53. Create a null ("h_loArmRoll") and parent it under the "b_offsetLo" bone, the elbow-offset bone that will actually be enveloped.

54. Set up a 2-point-constraint and pick the "c_elbow" and "e_arm". Lock the constraints' ppg.

The "distance percentage" will be used to distribute a number of helping nulls along the lower arm

55. Switch OFF the active flag in the upVector tab (otherwise the object will not be able to fulfill the expression set next)

56. Set an expression on RotX of the helper null

```
b_wrist.kine.local.rotx / 2
```

From now on, the x-rotation of "h_loArmRoll" will be half the x-rotation of the wrist bone.

57. Copy "h_loArmRoll" three times

58. Adjust the "distance percentage" in the 2-point-constrain ppg for each new null, evenly distributing them along the lower arm. 25%, 50%, 75% and 100% (to have one at the wrist itself).

To do this quickly, select the first null in the explorer, open the constraints' ppg ("select" in the MCP) and set this ppg to "similar" (the "two eyes"-icon beside the lock-ppg icon). Set the distance and select the next null in the explorer – the ppg will update to the new nulls' constraint ppg.

59. Select all "h_loArmRoll" nulls and hit "0" to open the expression editor. Adjust each expression to ensure even roll distribution. The null at the wrist could be /2 the rotation of the wrist (if the character is wearing a shirt which doesn't need to follow the wrist 100%). The next null /4, then /8

and /16 of the wrists' rotation, as we're getting closer to the elbow.

`b_wrist.kine.local.rotx / 16`

When rotating the wrist in x now, the nulls show how evenly the rotation will be distributed onto the mesh.

60. To be sure to avoid unexpected behaviour when resetting one of the arms control objects, set the Y and Z rotation of all "h_loArmRoll" nulls to 0 (by an expression on each parameter).

61. Also, as the geometry will be enveloped to those new nulls, the volume scaling of the arm needs to be implemented into those nulls. Here it is enough, however, to set the right expression for SclY and SclZ on ONE of the helper nulls, and simply link the other ones to it.

unsure, enable "relation" display under the eye-icon above your viewport.

65. Now set the RotX of the helper null to be / 2 of the original upArm bone rotation ("b_uparm", not the elbow offset upArm bone).

66. Lock its other rotation axis to 0 using expressions, like before.

67. Copy the helper null 4 times, offset the distance percentage of the 2-point constraints and edit the RotX expressions to gradually subdue the arms rotation towards the shoulder – like you did for the loArm earlier.

This "h_upArmRoll" nulls don't need to be parented under the "b_upArmOffset", and in fact, the "b_upArmOffset" isn't needed at all anymore! So, go ahead and delete it if

to make the elbow come out a bit – and so on and so forth...

13. MIRRORING THE ARM RIG

When mirroring the arm rig onto the other side of the character, be sure that "freeze negative scaling" in the *Skeleton>DuplicateSymmetry* dialogue box is OFF. You do want to duplicate the constraints, yet sometimes it happens that a control object the snaps to an awkward position. In this case, simply reset it back to 0.

Having said that, of course, I'm assuming that the control objects are zeroed-out in their default position. You can do that easily by selecting *Create>Skeleton>SetNeutralPose* for each control object.

Also be sure to adjust the expressions for the roll divisions to work correctly on the other, mirrored side! (eg. set them to "– wrist rotation / 2" etc)

14. SCRIPT

Obviously, it's quite a bit of work to transform a chain into this setup. It can be quite time consuming for let's say both arms and both legs of a character – especially when later on the chains need to be moved and the rotation points adjusted. Then all expressions for the bone lengths, the distances etc. would need to be re-adjusted – quite tedious.

To avoid that, make sure that the positioning of your bones is final, or use a script to automate at least a part of this process.

15. CONCLUSION

We now have completed a quite advanced arm rig for a character. It is especially tailored to the needs of animators and should be able to fulfill most of their needs. Naturally, different animators like different rigs or approaches, but I hope that this tutorial has given you enough information or ideas to come up with your own specialties for controlling the movement of an arm.

Written by Christoph Schinko, 2006
office@christoph-schinko.com
www.christoph-schinko.com

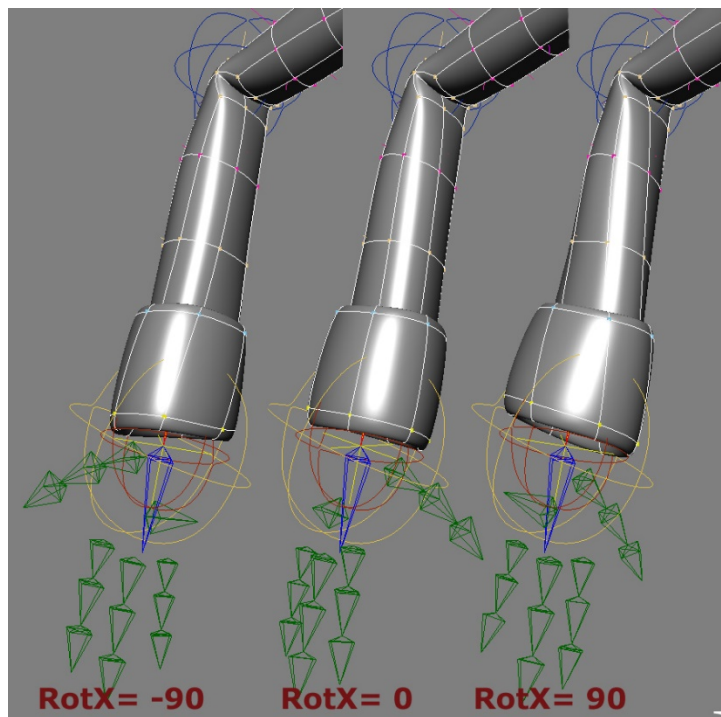


Figure 8: Arm roll divisions

THE UPPER ARM

Similarly, helper nulls can be set up for the upper arm roll.

62. Create a null and 2-point constrain it to r_arm and c_elbow.

63. Disable both tangent and upVector in the constraints' ppg.

64. Set a direction constraint towards c_elbow

When you picked c_elbow first in step 54. be sure to set the direction constraint towards r_arm. If you're

you like, or keep it for visual feedback. However, you will now see why it wasn't necessary to fix its rotation problems with the upVector earlier in step 20 (actually, you could do the same with the "b_loArmOffset" and its helpers).

ELBOW HELPER

Now you might set another null to the elbow and give it half the rotation of the lower arm, just to make things a bit easier when weighting and to keep the elbow geometry where it should be. Of course, it can also be linked to the lower arms rotation and relative values be set on the nulls translation,