PRORIG

MAGAZINE

Welcome

Welcome to PRORIG! 3D rigging is amazing yet it can be a bit challenging sometimes. Since this is the first issue of many (we hope so), we are going to start right from the basics. By doing so, at the end of this practice, you'll be able to master (almost) any type of character rig. The secret is using funny and unusual models, not the basic human one. Indeed, you will be practising on three of the aliens from the animated mock-commercial "Space Capades". Well, no more waiting; enjoy the issue! (and rigging, as long as things go smoothly).

To whom made it possible...

First, I would like to sincerely thank my supervisor, Prof. Roberto Pompili, for giving me the opportunity to work on this topic and for his dedication and patience.

Sincere thanks to Pietro Ciccotti for his help and for being understanding.
I'd like to express my gratitude to my family for all their support over the years.
Special thanks to my late night partner and co-producer Francesca, without you this year would not be the same.
Thanks to Bianca, for always being supportive.
Thanks Cristina for helping me with my doubt when writing.
Finally, I want to thank the whole Space Capades Team: Vito, Marco and Matteo.
Moreover, thanks to all those who accompanied me in these two years; classmates, teachers and all other amazing people who inspired me.
I'd like to thank Emanuele Ragnisco for always encouraging me and gave me the opportunity to work with Paesi Edizioni. In particular, thanks to Lucio L. Tirinnanzi and Catia Buselli for being so great. I'll be always grateful.
Last but not least, thanks to the Boar Hat of course for always being a safe place.

Contents

Core course

Rigging fundamentals

Step by step

Rigging the aliens from “Space Capades”

Step by step

Rigging simple props for animation

Deep dive

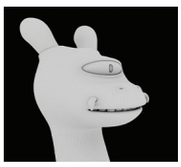
MEL and Python scripting

Core course

Rigging fundamentals

Maya 2022

“The plan is to go through each element separately for a clearer understanding of the whole process. At the end of this guide you should be good to dive into the actual rigging process”



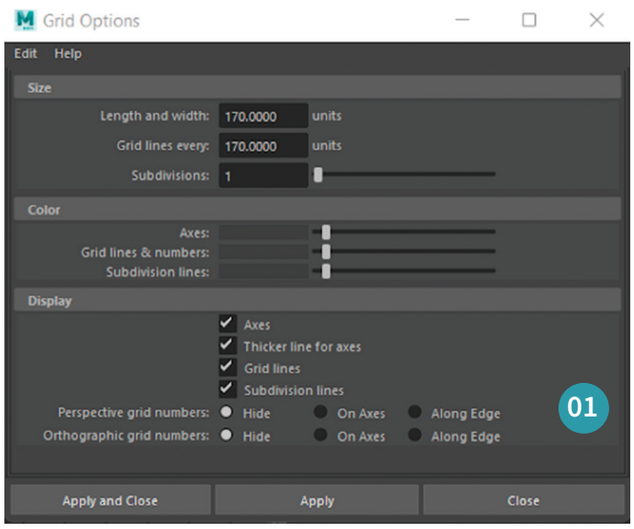
Model evaluation

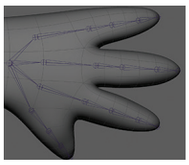
Make sure the model is ready for rigging

01 First and foremost, do examine the model thoroughly. Check the actual scale of the model by clicking on Windows > Settings/Preferences > Preferences . Under the settings tab you can see the working units (default: centimetres) then you have to adjust the grid so you can see how tall the character is: go to Display > Grid and open the options. By setting “Length and width” and “Grid lines every” to the desired height of the model you can quickly check if it’s big enough; reduce the Subdivisions to 1 to see the main lines only. And make sure that’s inside the square. If not, adjust the scale. We usually need a line each metre so set Length and width to 100.0000 units (since 1 m = 100 cm) after the model’s check.

Now, ensure that the model’s feet are actually on the x-axis. From the front view, the grid line is supposed to be in the middle of the model since it makes using the symmetry tool easier. A good topology is important because it allows good deformation and preserves textures. Turn on Wireframe on shaded (which can be found under Shading from the Panel Toolbar) and ensure there is enough topology in the deforming areas, consider adding edge loops if needed.

Finally, delete anything unnecessary, freeze transformations on the model and centre pivot. Also, if a mesh doesn’t need History, cleaning it up is always good: skinned objects with non-zero transforms can cause skin weighting to break. Remove all unused shaders from the hypershade too. Centre pivot on each mesh then group all of them under a geometry group and you should be all set.

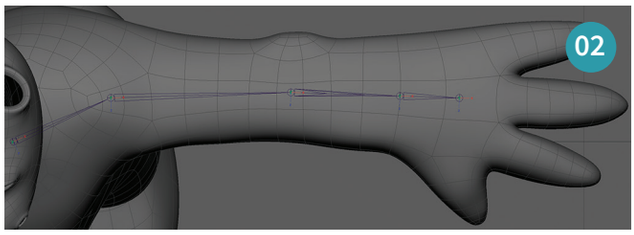


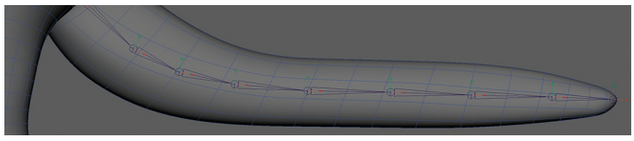


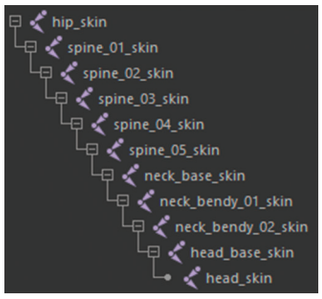
Create the skeleton

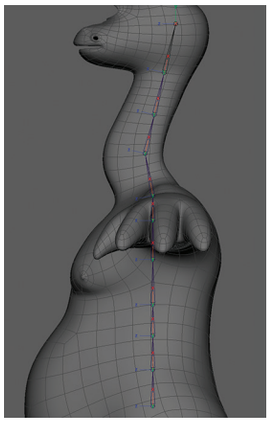
Joints: the building blocks of a rig

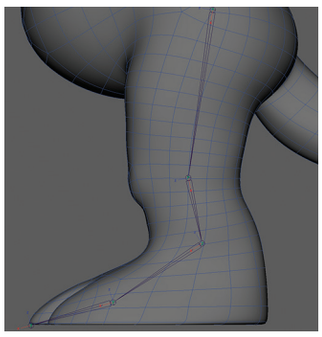
02 **Joint creation** First we need to change our menu set to Rigging (F3); the main menu updates so we can easily pick any rigging tool. Now, by selecting Skeleton from the main menu we can click on Create Joints and we'll notice that the cursor is changed to a crosshair icon (joint creation mode). By holding the X key (keyboard shortcut to turn the snap to grids tool on) we can easily place each joint on the nearest point on the grid. If we create another joint it will be automatically parented to the previous one. Of course we don't need to keep holding X to create them, it just allows us to avoid undesired rotation values. Also placing ourselves on different views will prevent these kinds of rotation. It is time to start building our rig. From top view, build both arm and tail structure as follow: snap the clavicle joint to the grid then holding shift key place humerus, radius and wrist joint on the same line. Moreover, we placed an additional joint at the end as a hand placeholder, aligned to the elbow, to ensure that the wrist's orientation matches that of the radius. In order to place the arm in the correct direction, rotate the clavicle joint backwards; rotate the humerus joint in the opposite direction.
Furthermore, we have to slightly bend the elbow joint on the y-axis because it will enable the IK system to work properly. Alternatively, we could make the radius assume its preferred angle by rotating it on the y-axis until we reach the desired angle. Then RMB on the selected joint and move your cursor onto the Set Preferred Angle button. When done, check it in the radius Attribute editor Joint tab (under "Preferred Angle" there will be non-zero rotations). On the contrary, the tail is much easier. If we look at the rig from the front view, we see that both arm and tail are snapped to the grid. We only need to move them up according to the mesh position. Move to the side view and build the spine and neck chain.
Staying in side view, place the femur joint then holding shift key place knee, ankle, foot_ball joint and finally the foot_end joint.

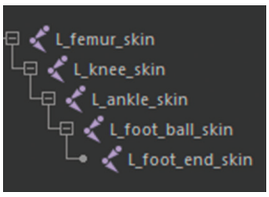












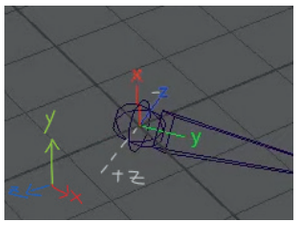
03 **Joint Orientation** As you can see, in the pictures all the joints are well oriented yet this is not by default. Indeed we have to correctly orient each joint. Joint orientation is important since, if the rotational axis doesn't point to its child joint, we will end up in wrong rotations.
Select all the joints in the hierarchy using this script *(page 46, 02)*:

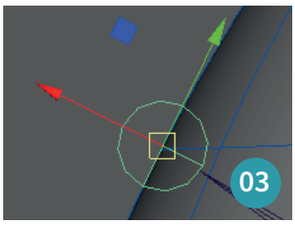
```
cmds.select(cmds.listRelatives
type="joint", ad=1))
```

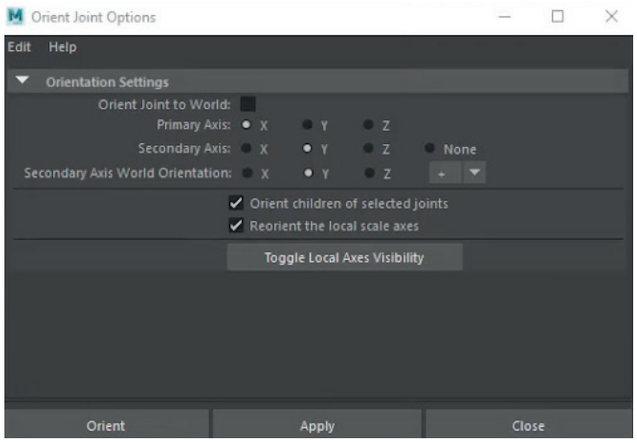
Then go to Display > Transform Display > Local Rotation Axes. The selected joint's local rotation axes appear. Now, we can fix joints' orientation and visually realise the change. Go to Skeleton > Orient Joint and open the options. We need the x-axis down the bone so it will be our primary axis. Instead we select y as both the secondary axis and secondary axis world orientation. The reason being these settings both work together; we have to tell Maya which world's axis the secondary axis should follow. Since we want the y axis to be pointing up we look for the world's axis who's pointing in that direction, displayed in the lower left corner of the viewport. Regarding the z-axis, we always want to point forward. To sum up, we are basically saying that we want the primary axis down the bone and the secondary axis to point in the same direction as one of the world's axes. We can also fix orientation only on the selected joint by simply turning off the Orient the child joints of selected joints option. We notice that Orient Joint Tool is not fixing the end joints. We want them to match their parent's orientation but since end joints don't have another one to point toward, they don't change. To fix them we have to enable Orient Joint to World and apply. The reason this works is because child joints' coordinates are relative to their parent space. In other words, you can read "Orient Joint to World" as "Match joint orientation to its parent joint". We can go a step further and adjust each end joint orientation in such a way that the x-axis is perpendicular to the geometry. So, duplicate the end joint, move the copy out so it's aligned to the geometry then parent it back to the end joint. Update the end joint's orientation using the orient joint tool (here, we don't need to enable Orient Joint to World); it will now move in a much more natural way. Delete the copied joint when done.

TIP

You may have noticed that there is a dropdown menu next to the Secondary Axis World Orientation. It allows you to choose between positive and negative world's axes. For instance, if you want z to point backward you can set the dropdown menu to minus and z will be pointing the opposite way of the world's z axis.







04 **Rotation order and Gimbal lock** If we rotate a joint using the default manipulator each axis moves with it and we can also click between any of the axes and rotate the joint. The thing is this isn't what's happening under the hood; if we double-click on the rotate tool (or select joint > shift key + ctrl key + RMB > Gimbal) and change Axis Orientation to Gimbal we see the true representation of what the rotational axes are doing:

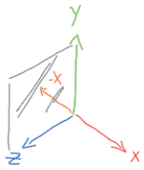
- By rotating Z, everything follows;
- By rotating X, Y and Z are left behind;
- By rotating Y, X moves and clashes with Z

The third case is the so-called Gimbal lock. Indeed, we are actually reducing the degrees of freedom of the joint since we can only rotate on Y axis and Z axis (X axis can only rotate in the same direction of Z). The reason behind this is the rotate order. In this example, the rotate order would be set to default order, xyz. When building the skeleton you have to set the best rotate order for each joint since it helps to reduce joint flipping. First, we need to keep in mind the primary axis of each joint and which axis is the most likely to rotate along. Please note that we can only minimise the chances that the gimbal lock will happen since we can never get to a situation where we can completely avoid it. To save some time we can select the root joint and easily update all rotate orders using the following Python script *(page 46, 03)*:

```
jointList = cmds.ls(type="joint")
for joint in jointList:
      cmds.select(joint+
      ".rotateOrder", 0)
```

Actually, some joints would benefit from different rotate orders. So we have to manually change them. Take the elbow, for instance. Its primary movement would be around the x-axis, so yzx is the best solution; everything follows now as the elbow bends. Same for the knee and ankle. Instead, since the orientation of the head joint matches the world orientation, we can set its rotate order to xyz. Once all our joints are created, oriented, and their rotate orders are set, add them to a skeleton group. These joints will be skinned, so for rig's consistency you should add the suffix _skin to any of them.

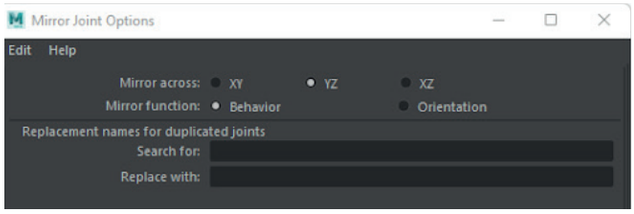
05 **Mirror joints** We may want to mirror a joint. Indeed we can save a lot of time by building the joints on one side only, then mirroring it to the other. Select a joint to act as the central point, it must be parent to the one you want to mirror. Open Mirror Joint Options which can be found under Skeleton > Mirror Joint. First we choose the right mirror across option. For a better understanding, let's try to mirror the left arm chain. Select the L_clavicle_skin joint > open the Mirror Joint Options and select Mirror across YZ axes. It might sound confusing which axes to choose, yet try to imagine the y and z axes are forming a plane, so the reflection will be on the opposite side of that plane.

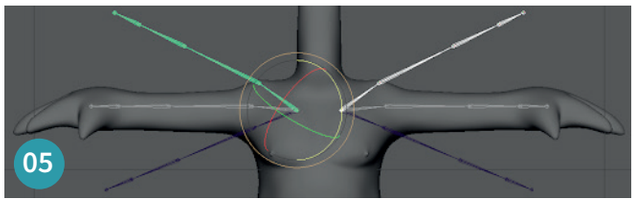


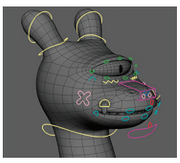
In order to get there, Maya has to mirror across the x-axis which is the primary axis of our joints. Since we are mirroring by Behavior, if we're raising the left clavicle it's going to raise the right clavicle too. As we can see, each joint orientation is flipped. Sometimes, we want to preserve the same orientation on both sides (i.e. the eyes). To achieve this, we simply change Mirror function (in Mirror Joint Options) to Orientation > click mirror and invert the translate x value, since we are mirroring across the x-axis.

TIP

As mentioned above, a child joint's coordinates are relative to their parent space, so the joint's translate values are based on its distance from its parent. In this case both right and left clavicles are parented to the world. In other words, they are equally distant from the world's origin; one on the right side (negative x) and one on the left side (positive x).







Building the UI

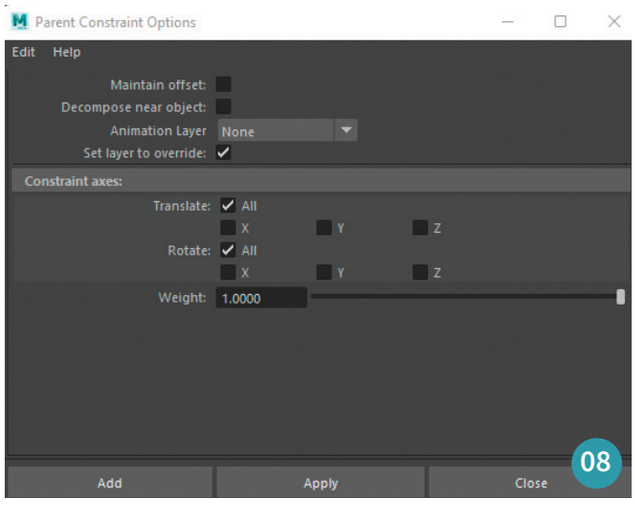
Controls and constraints

06 **Control creation** The natural step to take to continue building our basic skeleton would be to learn how to control those joints. Basically, we don't want the Animator to directly select the joints, so we need to connect them with controls.

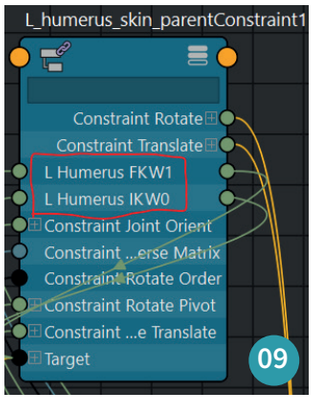
First we choose the shape of the control, which can be found under Create > NURBS Primitives. Alternatively, we could create custom controls with the EP Curve Tool or edit the control's shape through Component Mode.

07 **Offset groups** Create an offset group "jointName_offset" to store the actual values of the control (ctrl + G). Select both > centre pivot > delete history > freeze transforms. At this point, your control should be parented to its offset group and they should both sit in the grid centre with zero values. Snap the offset group to the joint the curve is going to control (or holding down the V key or select both group and joint > Modify > Match Transformations > Match All Transforms) then ensure you deleted history and froze transform on that joint.

08 **Constraint** At last, select the control > select the joint > Constraint > Parent (maintain offset disabled). This way, we are forcing the joint to follow every translation and/or rotation of the control. Disabling the Maintain offset option means forcing the joint to match the control's position. On the contrary, we can constrain the joint to the control without being in the same place by enabling Maintain offset.



09 **Unclean controls** It may happen that the joint moves away from its original position after connecting the control. It may be caused by freezing the transforms from the control before zeroing them out. You can fix this by undoing the constraint, move the control out of the hierarchy (and its offset group) and reset its transform values. Now, the control should move back to the grid's centre. If not, just snap it to the centre > delete history and freeze its transforms. Move its offset group out of the hierarchy and match its transformations to the control. Then move the control back to its offset group and the latter back to the hierarchy. Finally, ensure the control's rotate order matches the joint's rotate order. One last thing about constraints, if you load any type of constraint in the node editor you will see it comes with at least one weight attribute that corresponds to the target object. Constraint weights can be useful to specify which target object should control the object's influence. When there is only one target object the weight can only be 1 or 0. Otherwise the object's influence will be controlled equally by each weight.



TIP

You only need to zero out all rotate values and keep scale values to 1. Of course translation values will be preserved since they represent joint's coordinates in 3d space.

10 **Root control** It moves the whole character and it's the root of the rig, hence the name. For this reason, its pivot matches the world origin position and orientation. The Animator will use the root control to turn the visibility of any rig part on and off; to hide controls without diving into the hierarchy and much more. Therefore, you should give this control an intuitive shape like the character's silhouette or the shape of its head.

11 **Animator Friendly** Rigs can be scary, especially when complex. Therefore, think of the controls as the interface of your rig; Animators as end-users. As a UI, it needs to be: clear, you should easily understand each control's function; responsive, the rig should work fast and provide feedback (for instance, when you select something in Maya it turns light green; avoid using this colour for any part of your rig, otherwise it would be hard to understand if you selected successfully); consistent, naming convention and a default structure makes scripting easier; attractive, good looking control set pleases the Animators; efficient, your rig should be easy to animate; and forgiving, lock and hide everything the Animator shouldn't touch. This leads us to a final consideration: always

care about how Animators feel about your interface; ask them for tips and put yourself in their shoes.

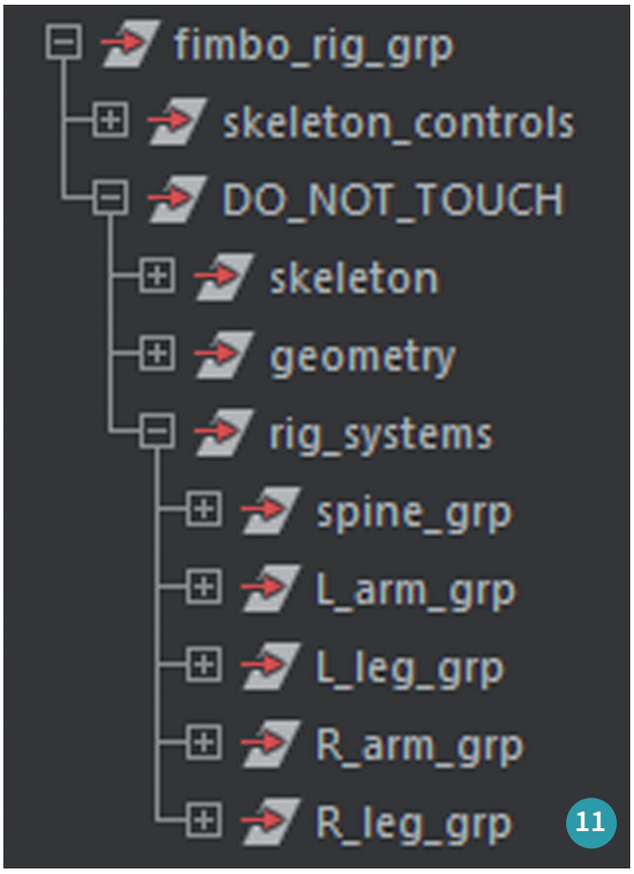
TIP

Sometimes you won't be able to actually select a control (for instance, you can't add keyframes on it). Of course you can select the control from the Outliner but it won't fix it and it can be pretty annoying to the Animators. The control could somehow be unselectable, so try removing it from any display layer. If the issue persists, add both offset group and control to a brand new display layer, check then uncheck the "R" on the layer and finally delete it. The bug should now be fixed.

Hierarchy set up

Consistency in rigging

12 **The importance of being consistent** In rigging, hierarchy is strictly important: it has to be clean and consistent. Bad hierarchy can cause inheritance problems and annoying issues. Besides, it also helps you debug and easily add new features. Plus, while in teamwork you should always decide on a naming convention and use it consistently.
So, create a main group called characterName_rig_grp then parent both skeleton_controls group and DO_NOT_TOUCH group to it The latter warns the Animator elements in there should not be touched. Store the entire mesh to the so-called geometry group and parent it to the DO_NOT_TOUCH group, along with the skeleton group. Last but not least, create the rig_systems group (§3.1) and parent spine_grp, both sides arm_grp and leg_grp. Currently these child groups are empty but we will populate them later.
Although these groups are mostly created to maintain order their pivots are fundamentals; they all match world orientation except for the rig_system's child groups, whose pivots match the first joint in each chain (i.e. L_arm_grp pivots matches L_humerus_skin's). Otherwise, some joints would move from their original position.
In other word, we are dividing the rig structure into two: control and deformation system. By separating them we are able to work on the rig without changing the main structure. Clearly, this is really useful when working in a pipeline. So, this far gone your rig structure should be as follows:



Step by step

Rigging the aliens from “Space Capades”

“In this tutorial we’ll walk through building a fully animable rig using three of the aliens from “Space Capades” as examples. In particular, we will explain how to deal with both conventional and unconventional setups, learning to solve all issues that may occur. By the end you’ll have all the tools you need to build your own complex rig, no matter which character it is.”

P1 Back to basics

How to create a simple rig

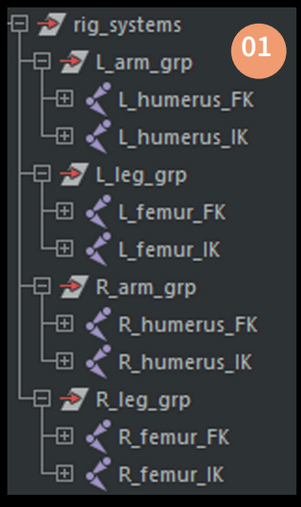
01 **Rig systems** After creating the basic skeleton, it’s time to build the rig systems. Although these joints won’t be skinned they’ll drive the whole skeleton’s movements.

Select humerus, femur and hip joints then duplicate these chains twice and delete all the duplicate constraints in there. We’ll use these chains to build forward and inverse kinematics. Select a chain for each duplicate joint and replace the _skin suffix with _FK. On the remaining ones, replace the _skin suffix with _IK. Then parent each IK and FK chain to the corresponding _grp in the rig_systems group

Match each group’s pivot with that of the first joint in the chains (i.e. left arm group’s pivot matches the left humerus pivot).

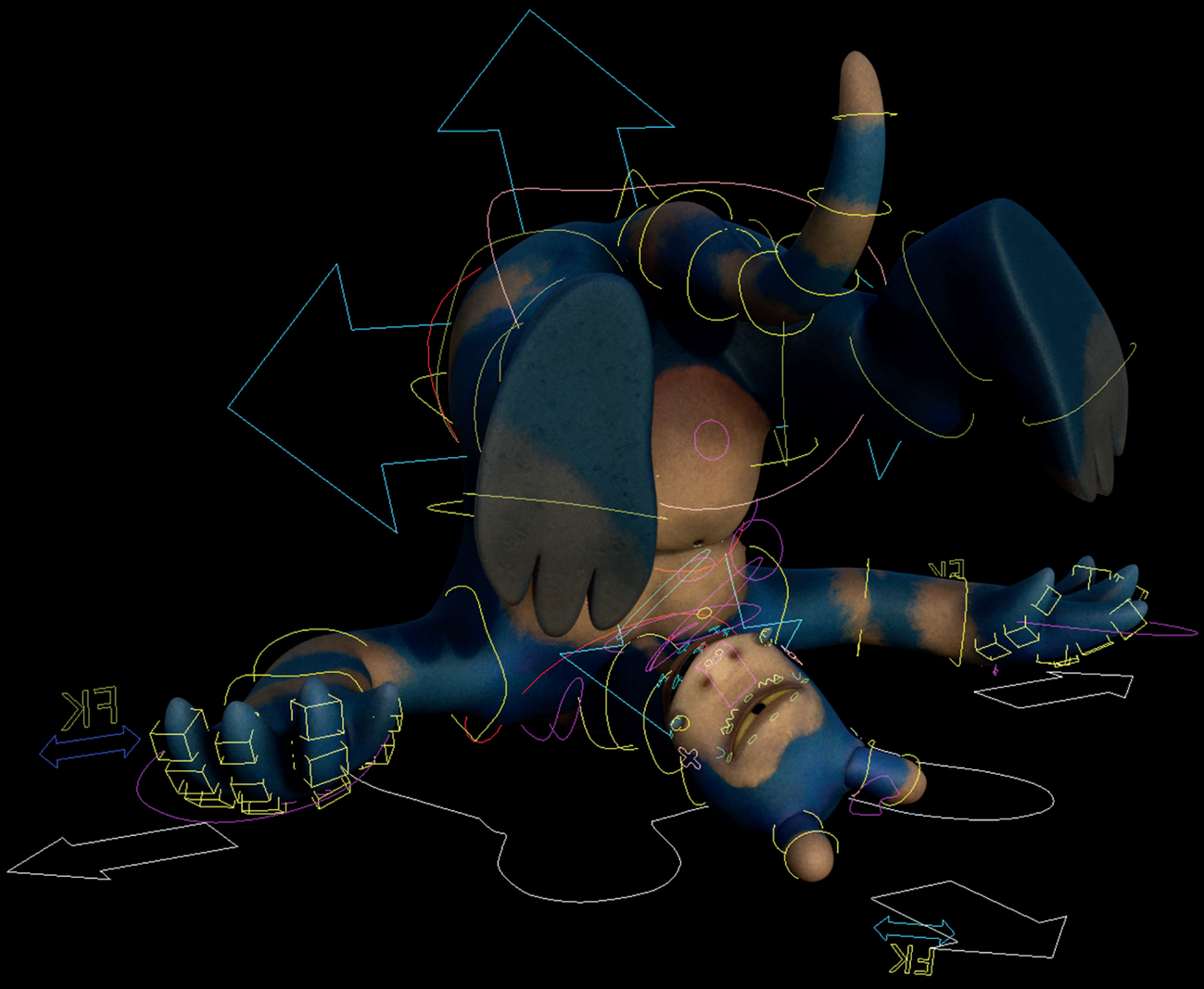
Now, for each IK and FK joint parent constraint (Constrain > Parent and disable Maintain Offset) them to the corresponding skin joint. In case the skin joints got non-zero rotation values after the constraint, consider rebuilding and duplicating them again since it could be caused by bad joint creation.

Sometimes, especially when it comes to visually overlapping joints, you can distinguish between different types of chains through specific colour assignments. Select the joints chain > Display > Wireframe Color and choose the right colour for your chain. You will see that rigging and debugging is a lot easier.



TIP

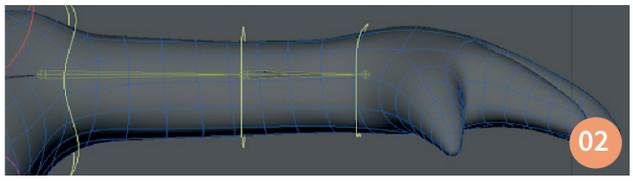
“Kinematics” refers to the mathematical description of motion without considering the underlying physical forces.


FK
FK

Adding FK/IK

Forward and Inverse kinematics system

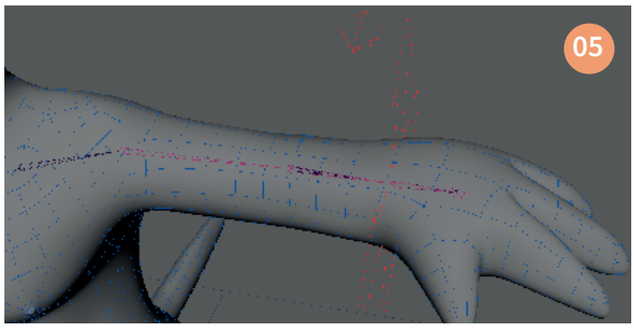
02 **Forward kinematics** Adding FK is pretty straight forward so we are going to do that for arm only. Use the NewControl script *(page 47, 04)* to create the one control for each arm's joint: humerus, radius and wrist. Adjust their shapes and hierarchy such that the joint's rotations evaluate from the humerus down to the wrist. Again, since we think of the controls as a user interface we can pick a specific colour to make it easier (use the slider under the drawing overrides tab in the control's Attribute Editor).



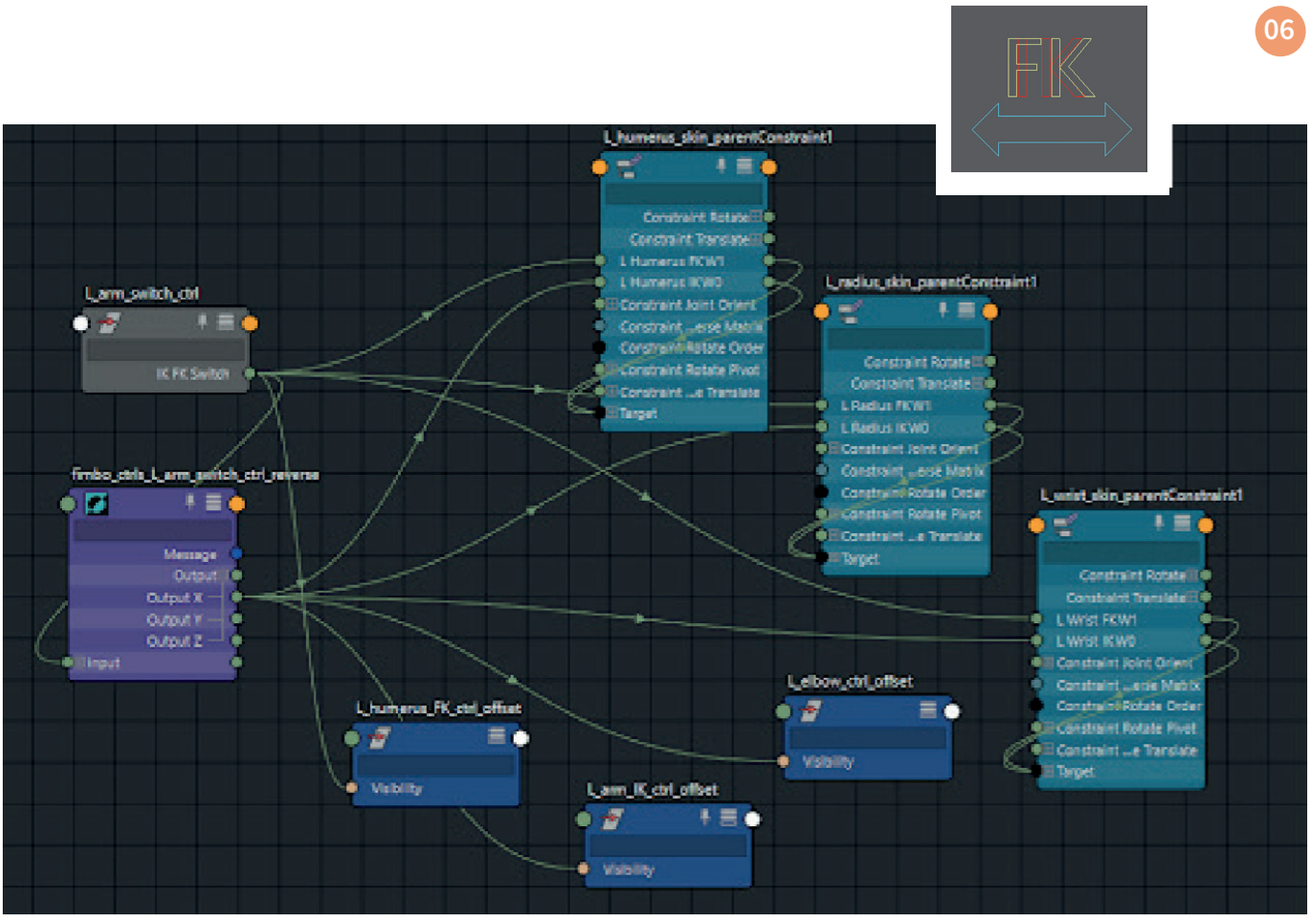
03 **Inverse Kinematics** Let's create the IK controls: one for the wrist and another for the elbow joint. We now need an IK handle so go to Skeleton > Create IK Handle and open up the options. Select the Rotate-Plane Solver, from the IK Solver dropdown menu, which creates a plane along the selected joint chain. Therefore, joints must have non-zero rotations on the axes outside the plane. In other words, in this example we have to avoid rotations along the z axis since the IK handle will create a XY plane. Otherwise, the arm might twist to maintain the alignment. Now that we know how it works, select the humerus IK joint then the wrist to create the handle. Parent the latter to the IK control and use it to drive the arm; the handle evaluates the joint's rotations from the wrist up to the humerus, hence the name Inverse Kinematics.

04 **Wrist rotation** Currently, the IK arm doesn't have control over the hand since the IK handle only works on translations. All we need to do is constraint orient (maintain offset disabled since they should both have the same orientations) the IK wrist joint to the IK control. In addition, the hand maintains its orientation no matter what the body does.

05 **Elbow control** The Rotate-Plane Solver we added before creates an IK chain that can be rotated and it's exactly what we need to control the elbow's position. Still we can't adjust the elbow directly. So, match the elbow control offset group's position to the radius joint then move the control backward (still using the offset group). By doing so, the elbow control points directly toward the elbow. Select the elbow control > select the IK handle > Constrain > Pole vector. This constraint uses the plane created along the arm and tries to keep the elbow aiming at its control. Since we moved back the elbow control we prevented the arm from flipping after adding the constraint.



06 **IK/FK switch** At this point, if we move IK or FK chain away the main skeleton will always lie between the two because of the constraint. Also we see both IK and FK controls at the same time; definitely not a user-friendly interface. To add the ability to blend between the two modes create the following custom control with an IK/FK switch attribute:
Use that attribute to control both the constraint weights and the visibility of each control.
Besides, you can use the same attribute to control IK and FK text visibility, keeping track on which mode is currently active. A little tip: if your arm moves when blending between IK and FK you probably ended up with some rotations on the bind chain. Check for non-zero rotations on the IK joints which could be caused by non-clean controls (this can happen on the FK chain too. Check §2.2 for solution) or bad joint creation. In the last instance, you should rebuild the bind chain (and recreate the kinematics' chain from it).



Hand rig

And how to Set Driven Dey

07 **Thumb fix** Once created the hand structure we have to fix thumb orientation before adding controls using the NewControl script *(page 47, 04)*. Select thumb joint > turn on the component type mode, then RMB the Select Miscellaneous components icon and select Local Rotation Axes. We can now edit each axis manually. Select all thumb's child joints (except for the end one since we have to reset it anyway) > rotate the joints so the z-axes point more towards the back.

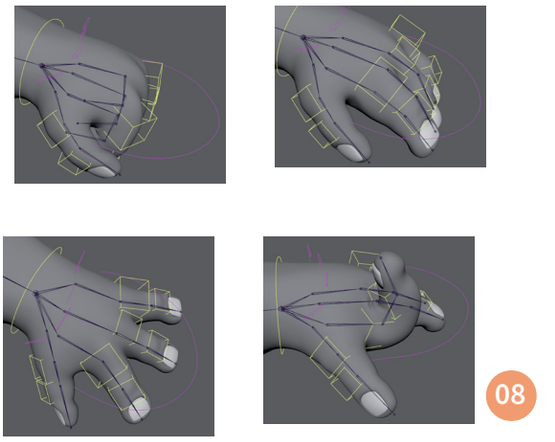
We might fix the rotation axes however there comes another problem: the translate axes aren't oriented the same way as rotate axes. We know there is an offset also because the Rotate Axis values from the Attribute Editor are not zeroed out. To fix this, switch back to object mode > open the Script Editor and run the following script for every thumb's child joints *(page 48, 01)*:

```
joint -e -zso
```

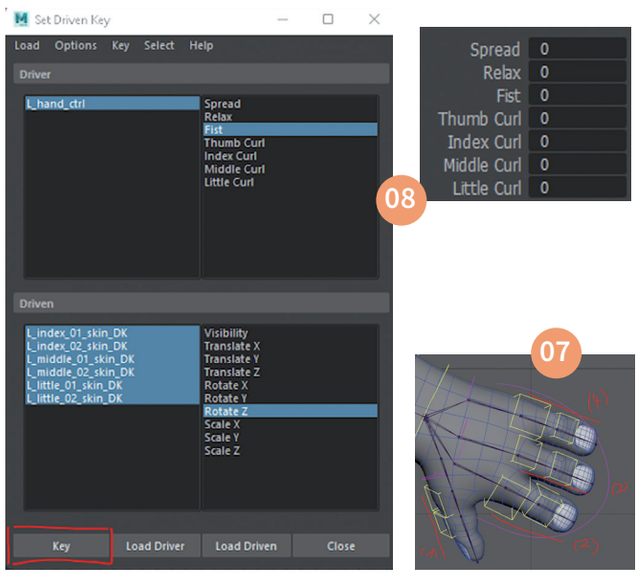
After deleting the history, freezing the transformations of each joint and adding the controls to them we are good to go and set driven keys. All finger controls are parented to a group, which is constrained to the wrist joint.

08 **Driven key** There are some poses the Animator might do often so it's a good practice to add simple attributes driven poses. By doing so the Animator can achieve those poses in the blink of an eye. First we need to change our menu set to Animation (F4) then go to Key > Set Driven Key > SDK window.

The problem is that SDK adds keyframes onto the nodes so we can't create them directly on the controls nor the offset groups. Therefore, for each control create a group beneath its offset group and add _DK suffix to its name. SDK window, load all DK groups from the controls as driven. Now, we need something to drive them so we create a general hand control, constrained to the wrist joint, and add poses attributes to it.
For instance, let's see how to set the fist attribute. Load the attribute as driver then select rotate Z from each DK group. Since we are posing the hand by rotating the DK, select all the controls then press the up arrow on the keyboard which will correctly select those groups. Set both the fist and the rotate Z attributes to 0 and store this pose by clicking Key. Change the fist attribute to 90; pose the fingers by rotating the DK -90 degrees. Hit Key to save the fist pose. Now, by sliding the pose attribute the hand animates automatically. Open the graph editor (Windows > Animation Editors > Graph Editor), select all curves then click on linear tangents (1), pre-infinity cycle with offset (2) and post-infinity cycle with offset (3). Which means there is no ease in/out and the hand fist endlessly both positive and negative. Repeat the process for all attribute driven poses. Some common driven key set ups are the following: fist, relax, spread, curl.

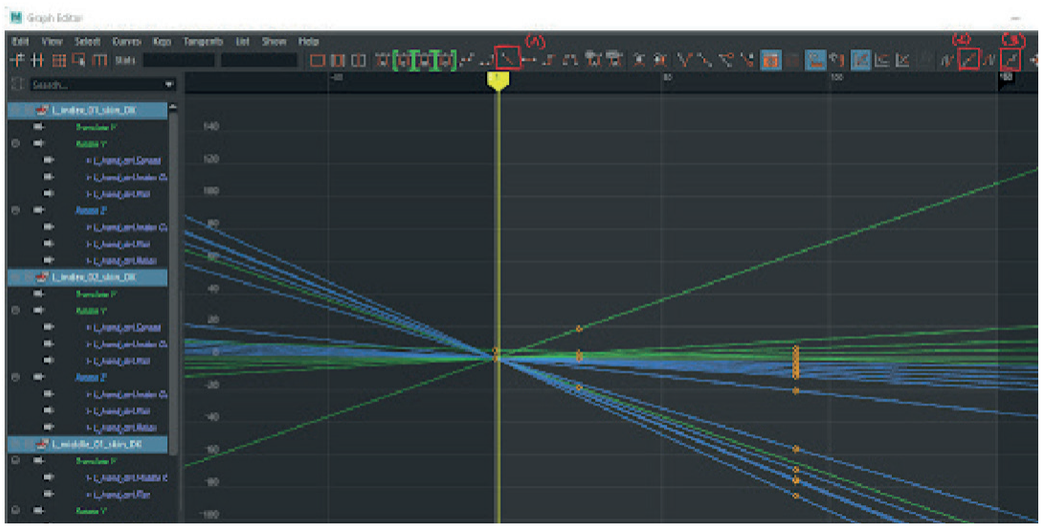


08



08

07

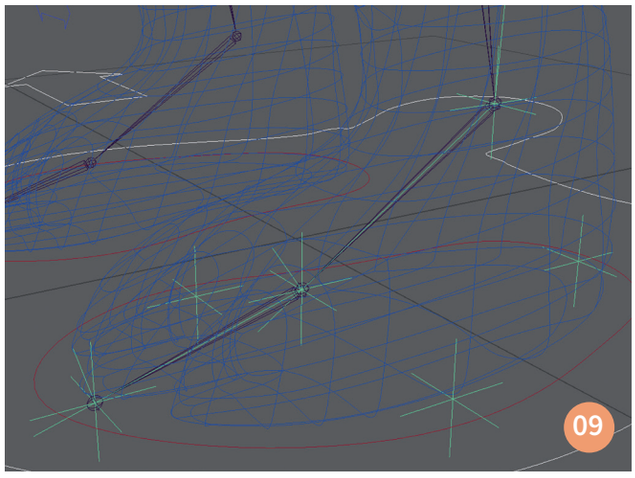


08

Foot rig

How to use the Expression Editor

09 **The locators** In this part of our tutorial, we'll build a foot setup to provide intuitive controls for the Animators. First, add all the main foot poses to the IK leg control. Just like in the hand rig, we can create a smarter and way efficient set up for the foot. This time, we are going to use expressions instead of SDK. Expressions are like a coding language and the advantage of using them is that they are very fast and more flexible than driven key. But, first we need to get information about foot joint's position and orientation. We can place a locator on the back of foot, on the toe, on the left and right side. Also snap a locator to the ball joint (in the middle of the foot) and another one to the ankle. Now, we add an IK handle (single chain solver) from the toe to the ball joint and from the middle to the ankle.



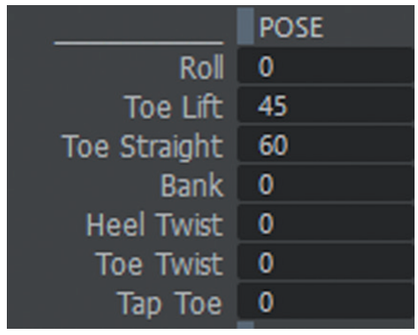
10 **Expressions** Finally, open the Expression editor (from Windows > Animation Editor) and write the first expression to enable foot roll:

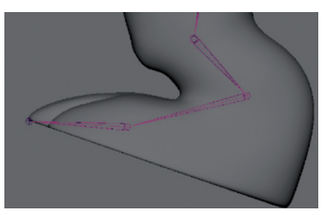
```
$roll=R_leg_IK_ctrl.roll;
$toe_lift=R_leg_IK_ctrl.toe_lift;
$toe_straight =
R_leg_IK_ctrl.toe_straight;

R_heel_locator.rotateX =
max(0,-$roll);
R_ball_locator.rotateX =
(linstep(0,$toe_lift,$roll))*
(1-linstep($toe_lift,$toe_
straight,$roll))*(-$roll);
R_toelift_locator.rota-
teX = linstep($toe_lift,$toe_
straight,$roll)*-$roll;
```

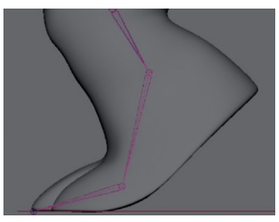
Finally, enable the tap toe attribute.

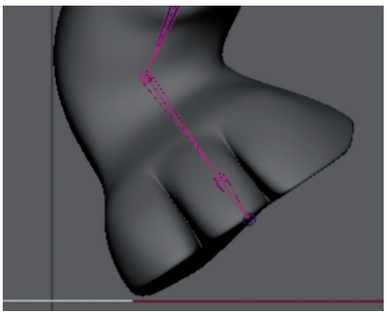
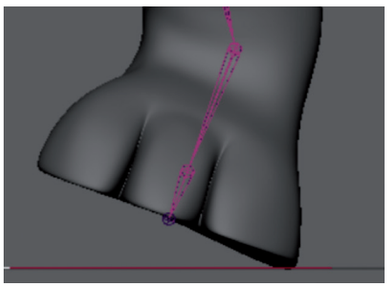
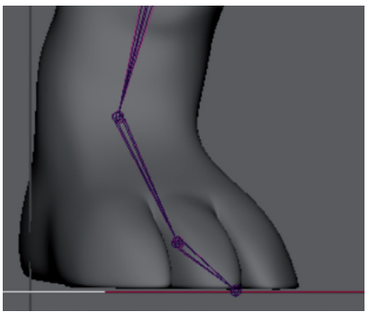
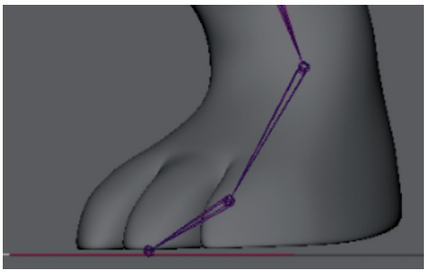
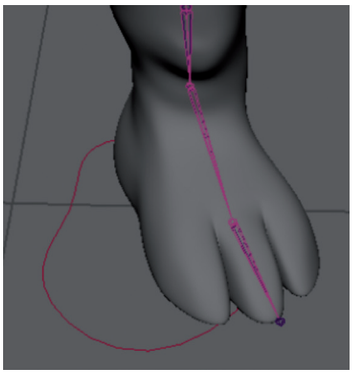
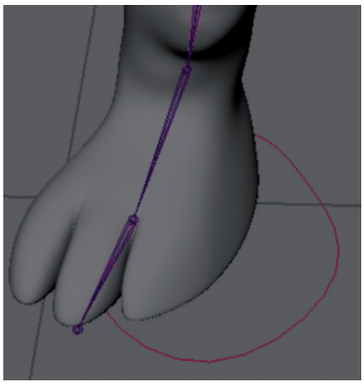
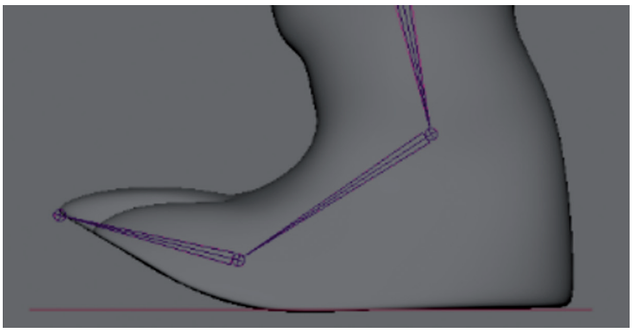
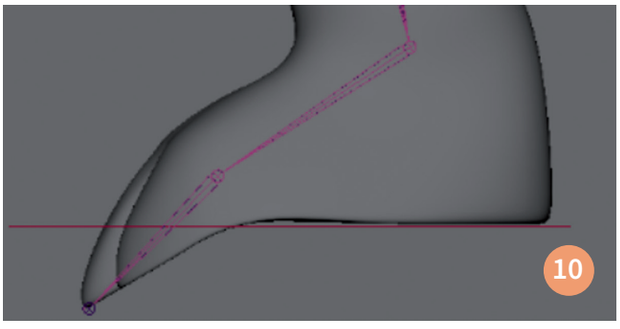
```
$tap_toe=R_leg_IK_ctrl.tap_toe;
R_ball_tap_toe_locator.rotateX =
-$tap_toe;
```

Now, the advanced foot rig is set up and the Animators can use the attributes on the IK leg control to tweak it . Indeed, we set Toe Lift and Toe Straight to default values: 45 and 60, respectively. It means that, when rolling, the foot gets off the ground as soon as the Roll value exceeds the toe lift. Therefore, by changing these values the Animator can change the foot behaviour.
In conclusion, some common foot expressions set up are: roll, inner and outer bank, both side heel and toe twist and least but not last, tap toe.







Torso rig

IK/FK spine and waist control

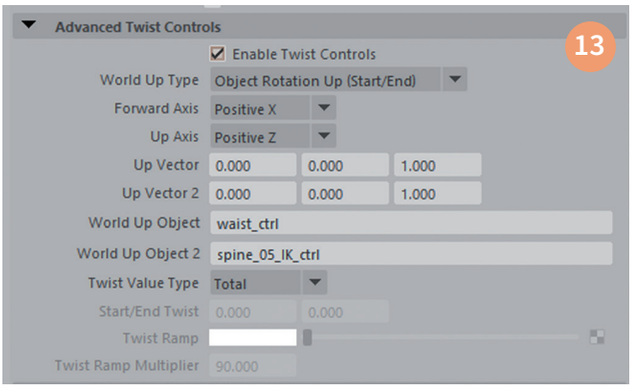
11 **Build the spline IK based spine** In this tutorial we are going to build a complete spine with both IK and FK controls. A common way to handle this task would be to create three joint chains: an IK, an FK and a bind chain then blend between IK and FK to drive the bind chain. Here, we'll try a different approach keeping both kinematic chains active. Duplicate the bind chain twice then rename those (the duplicate chains) IK and FK.

Create two controls; one for the last spine IK joint and one for the hip joint. We don't want them to influence the spine joints directly so let's create two control joints each. Since the IK controls should match the world's orientation we want these joints to match it too in order to prevent offsets when constraining. Once fixed, delete history, freeze transformations on these joints then constrain parent (maintain offset off) the upper control joint to the corresponding IK control only. At

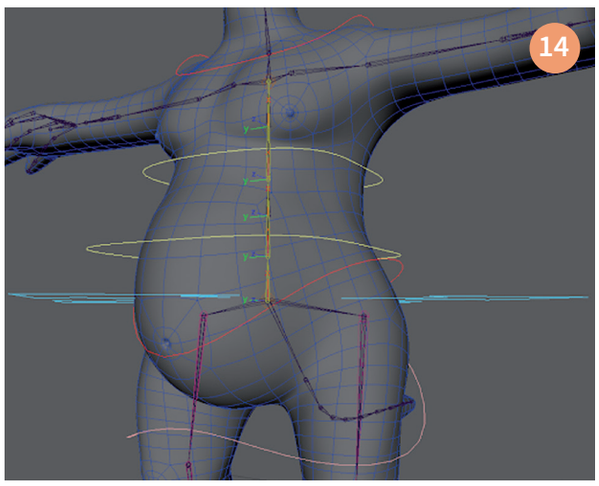
this stage you might be confused but trust the process. The heart of this spine setup is a curve. Go to Create > Curve Tools > EP Curve Tool and set the curve degree to 3 Cubic; it gets a softer curve. Holding V, click next to each joint in the IK spine to curve points. Press enter when finished. Now, go Skeleton > Create IK Spline Handle and open the options. We already have a curve so uncheck Auto create curve; we don't even want it to be parented to the joints so uncheck Auto parent curve too. Finally, select in this order: hip IK joint, upper IK joint, the curve. To animate the IK handle we move its points obtaining a more natural movement. So, bind the control joints to the curve (reset bind skin options to default and choose selected joints from the first drop-down menu). Basically, the IK controls drive the control joints which deform the curve and the IK spine with it. Some areas might not deform as expected but we can easily tweak them by editing the weight values on the curve's skincluster.

12 **Waist control** The time has come to give parent to the orphan hip control joint. Using the waist control to drive this joint will give us the ability to wiggle the hips. Snap it to the joint that's directly above the hip and bring the shape down to the character's waist. Constrain parent the control joint to this control. To see the full effect of this we need to parent constraint both the legs IK joints and humerus FK controls.

13 **Advanced twist** Before we move on to building the FK spine, let's enable the IK spine twist. When we rotate both the waist and shoulder IK controls only the arms and legs twist. Select the IK handle, open the Attribute Editor and open the Advanced Twist Controls tab, under the IK Solver Attributes tab. Check the Enable Twist Controls option and try rotating the IK controls again. We're now getting some movements yet it's not enough. Select Object Rotation Up (Start/End), from the World Up Type drop-down menu, because we are using two controls (one to twist each end of the joint chain). Now, we need to tell the handle which controls to use to twist the joints: add the waist control as World Up Object and the top IK control as World Up Object 2. Next, the forward and up axis refers to the spine joints. Therefore, the first one corresponds to the primary axis (positive x) whereas the up axis is the one pointing to the front view (positive z). As you might have noticed, positive and negative are decided according to the world space (for instance, if the z-axis points in the opposite direction of the world z-axis it will be identified as "negative z"). Add 1 (positive) or -1 (negative) in the up axis corresponding channel. Finally, thanks to this setup we are able to twist the IK spine. In addition, the waist and the upper IK controls are independent of each other so we have more control over the spine pose.



14 **FK spine** Usually, this chain has less joints than the IK one. But it's up to you; there is no rule. Add FK controls only to joints whose corresponding IK joint is not connected to any of the IK controls. Update the hierarchy so this spine works as an FK chain would. Now, we want the FK spine to drive the IK spine therefore parent the upper IK control to the last FK control. The hip IK control should move the entire spine without driving any joint therefore parent the lower FK control to it.
Finally, snap the cog control to the hip joint position then move the waist and the hip IK control into it. Essentially, the cog will drive the entire spine.

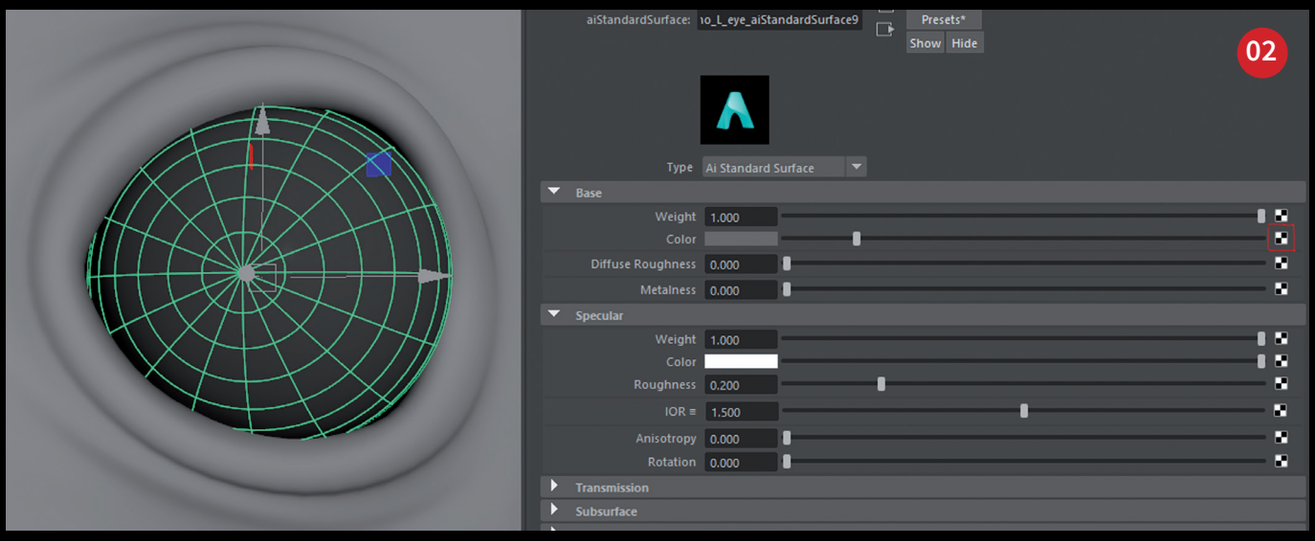


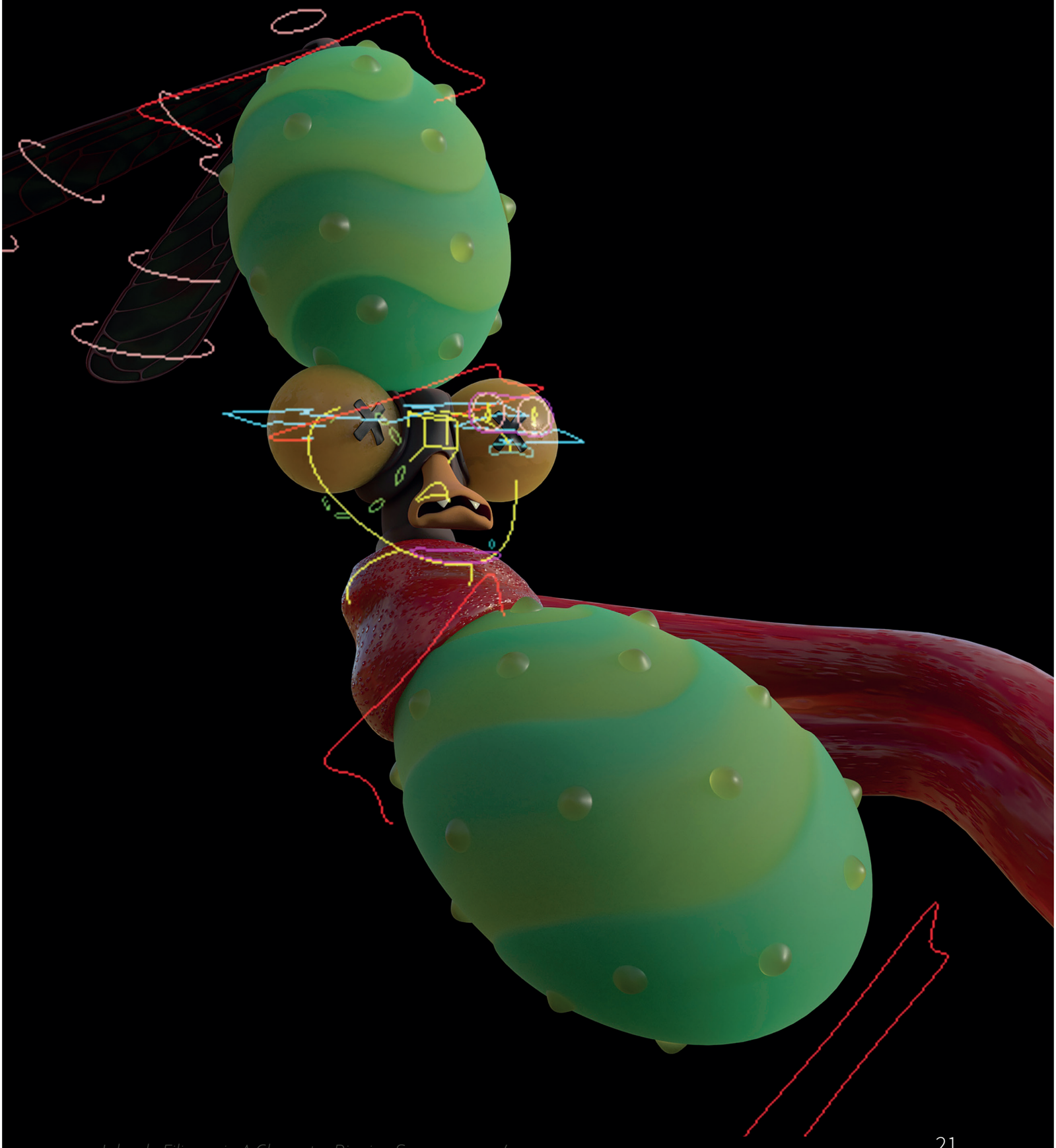
P2 Final touch
Facial rigging

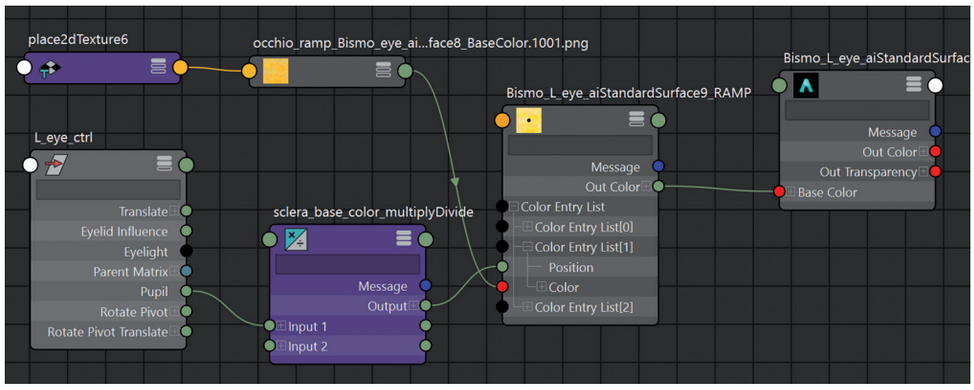
01 **Face rig** It's time to make the rig fully capable of facial animation as well as provide a wide variety of facial expressions. To drive the facial poses you would either use blend shapes or you would use joints. Here, we'll try to combine the two approaches: blendshape will give us the nice forms we need, and joints to add another layer of animation to move the main areas of the face around. In order to not increase the file size, we are going to use utility nodes instead of set driven key.

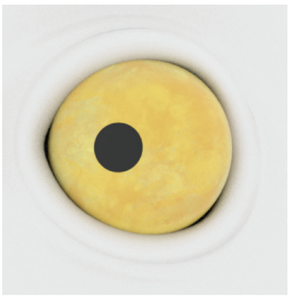
Cartoon eye rig
on spherical and oval-ish eye

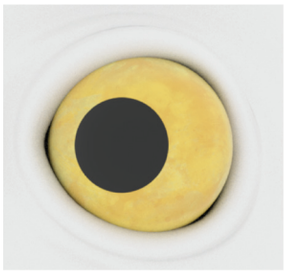
02 **Spherical eye rig** A fast and easy way to create a cartoon eye on spherical shape is to select its material then click on the chessboard icon on Color attribute and choose Ramp. Open the ramp tab in the Attribute editor and set the type to Circular Ramp to create the pupil.
In the bar there are two handles: one for the black colour (on the left) and one for the white colour (on the right). First, tweak them in order to get a sharp-edged circle. Then, click on the top circle from the white colour handle and click the chessboard icon from the Selected Color attribute. Now, choose File and select the sclera base colour from your computer. Basically, we are able to adjust the pupil's size by moving the ramp bar handles. The problem is that we don't want the Animator to access the ramp to animate the pupil. If we open the Ramp Attributes tab from the ramp we can see the Selected Position attribute changing as soon as the handle moves along the shader bar. This means that the Animator needs to indirectly control the Selected Position attribute. Therefore, create a multiply-Divide, set it to Divide, set Input 2X to 10 in order to gradually move along the ramp position range (0, 1). Add a Pupil attribute to the eye control and connect it to Input 1X. The Animator will use this attribute to control the pupil's width. Finally, pipe Output X into the ramp Position attribute and we are all set.

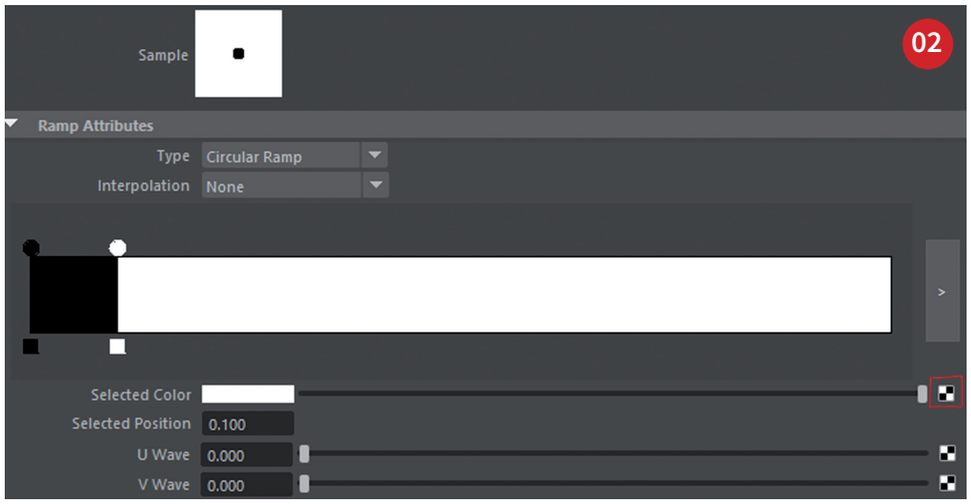






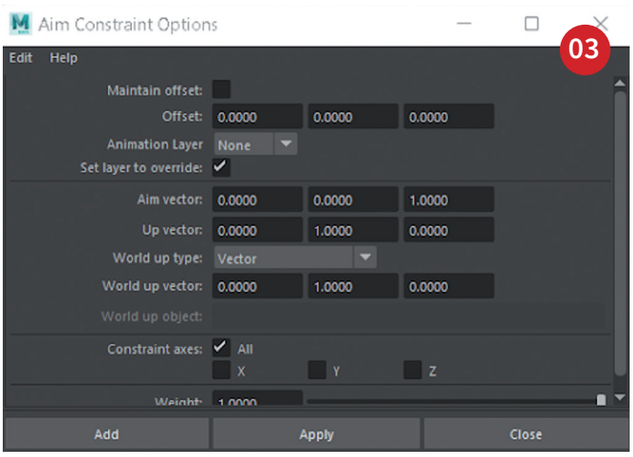


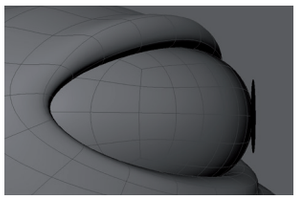


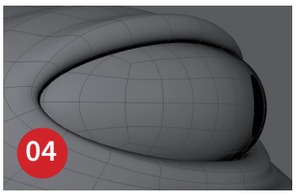


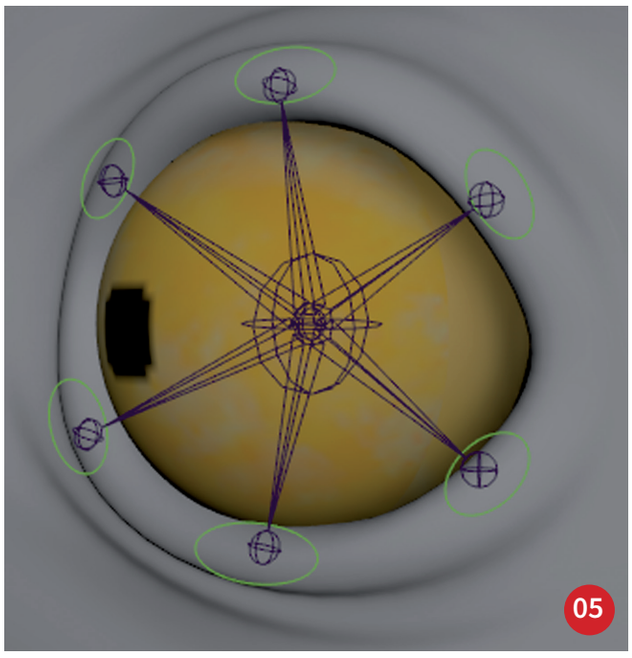
03 **Eye control** Create a joint and snap it to the centre of the mesh. Add a lookAt control parented to the root and composed of as many controls as the eyes. We want the eyeball to always look at the control's position, hence its name. The most popular way to do so is using an aim constraint (Constrain > Aim and open up the options). Select the joint and check which axis is the one pointing toward the control; set it as the aim vector. By default World Up Vector is set to Vector so we have to specify which axis is the Up Vector; the one pointing up. Then we set the corresponding up axis in world space (World Up Vector).It might seem confusing but the good thing about the aim constraint is that we can edit after applying it (in the constraint's Attribute Editor). So with one less worry aim constraint the eye joint to the corresponding control.

04 **Oval-ish eye rig** Rigging cartoon eyes can be tricky when it comes to unconventional shapes. Now, it's the turn of rigging some sorta of oval eye. Import the pupil mesh and line it up with the front part of the eye. Now, select the pupil, then the eyeball and go to Deform > ShrinkWrap (projection: closest). Technically, the pupil is now shrink wrapping onto the surface yet it probably looks wrong. So, open the eyeball attribute editor and open the shrinkWrap tab. Adjust the Target Smooth Level to make it, you guessed it, smoother and tweak the Offset attribute to avoid the pupil from sinking into the eyeball mesh. Finally, constraint parent the pupil to its control and connect scale through direct connections.

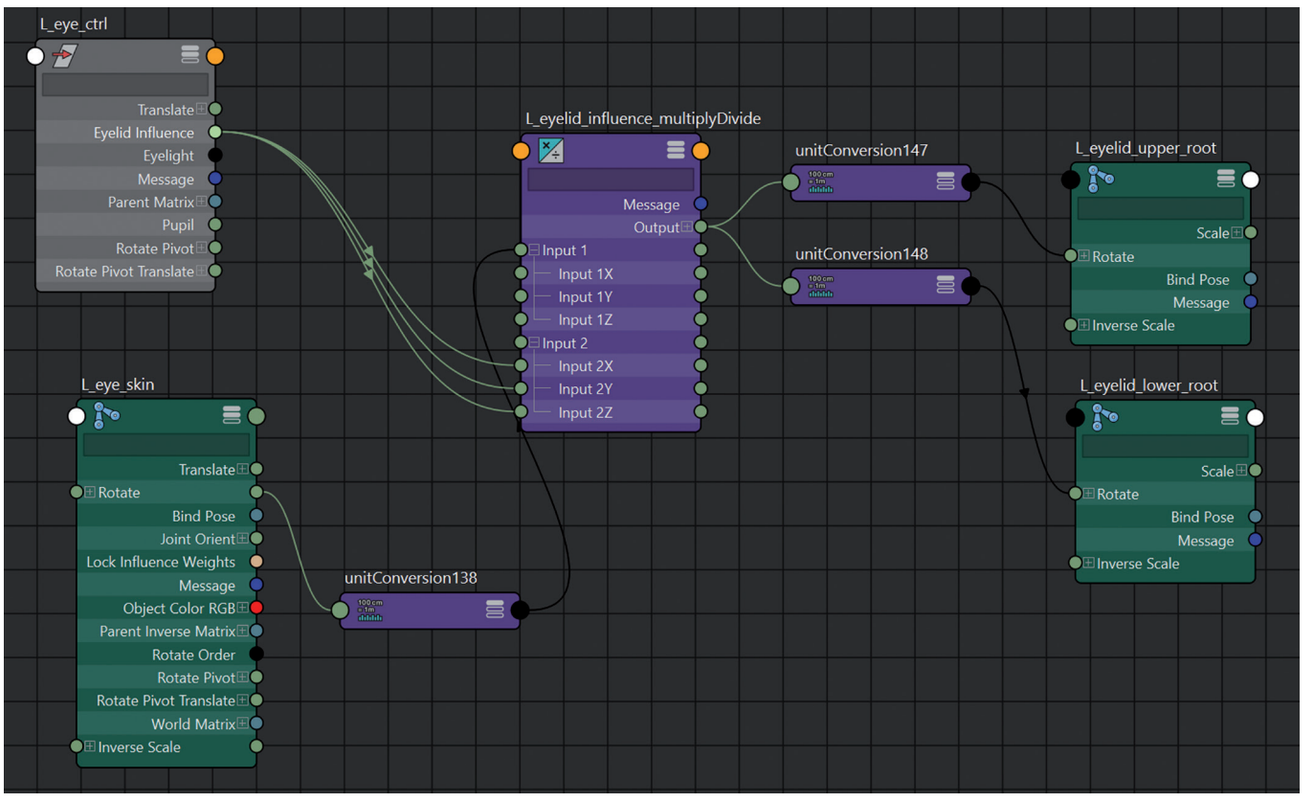








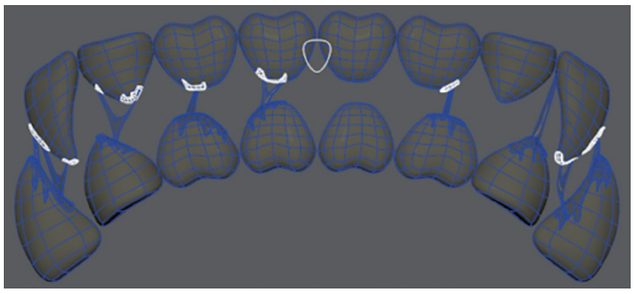
05 **Making eyelids follow the eye** We can add eyelid joints as an extra layer of control since the blinking is created using blendshapes. In the same position of the eye joint create both lower and upper eyelid root joints, then add eyelid joints.
We are one step away from building a satisfying eye rig. The only thing left is to add some skin sliding effect to the eye area. Indeed when a real eyeball rotates the eyelids don't stay still. To achieve this slight movement we add a multiplyDivide node which divides the eye joint's rotations by a certain value. This value can be adjusted by the animator using the Eyelid Influence attribute on the corresponding eye control.

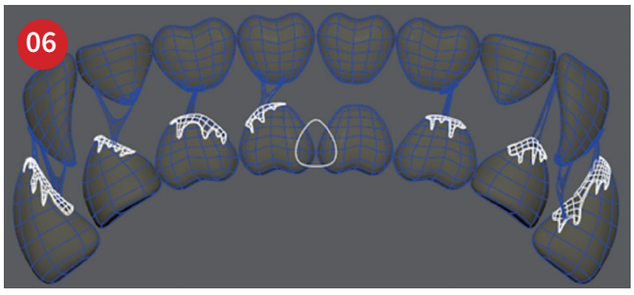


Saliva and teeth rigging

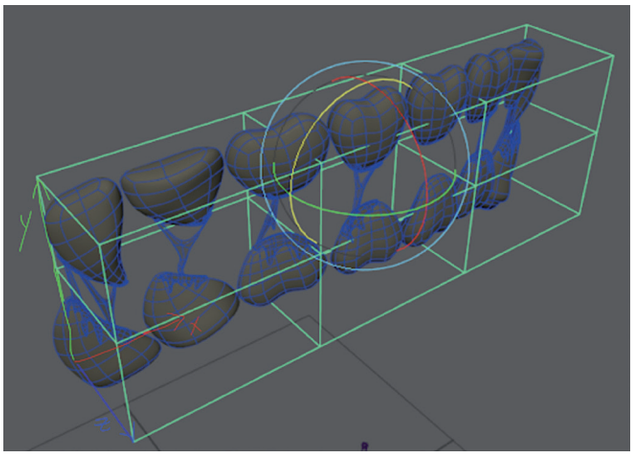
Rigging using Deformers

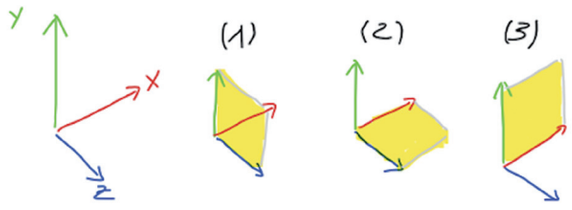
06 **Saliva setup** We've almost done, only things left are teeth and an extra feature: saliva.
By adding the latter directly to the rig it will automatically stretch as soon as the Animator opens the mouth. Teeth are divided into upper and lower arches, whereas saliva consists of three parts: top, mid and bot. The top part is stuck to the upper teeth therefore we both bind them to the upper teeth joint.
We skin the bot part and lower teeth to the lower teeth joint instead. Then we parent the upper teeth joint to the head and the lower teeth joint to the jaw.
Finally, we create two teeth controls to drive upper and lower teeth separately.



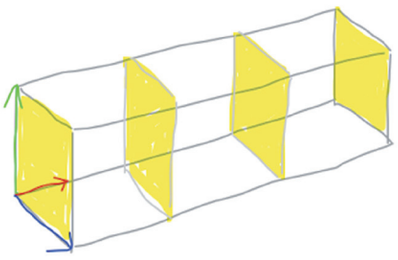


07 **Adding Lattice** Regarding the mid saliva, we'll use a different approach.
We create a lattice deformer, that can be found under Deform > Lattice.
It includes two lattices: an influence lattice, ffd#Lattice, and a base lattice, ffd#Base; both grouped under ffd#LatticeGroup. The lattice's deformation is based on any difference between ffd#Base's lattice points and ffd#Lattice's lattice points.
In other words, we focus on the latter. Sometimes, you need to adjust the lattice's structure by changing lattice's divisions (which can be found under ffd#LatticeShape tab in ffd#Lattice Channel Box). In order to understand the amount of S, T and U divisions you have to understand how STU space works. The coordinate origin is in the lower left corner of the lattice and as shown below.
We can create three different planes: YZ (1), XZ (2) and XY (3). These planes are the coordinates of the lattice's STU space since YZ refers to S, the XZ plane is the T coordinate, XY corresponds to the U coordinate. Basically, the number of each division is nothing more than the number of the corresponding plane cutting the lattice.



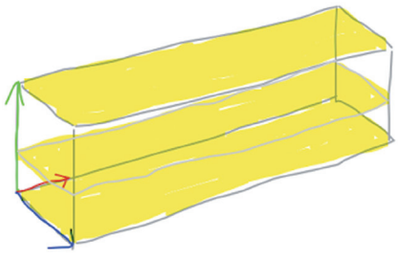


4 S divisions



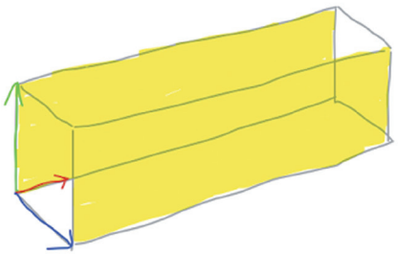
07

3 T divisions

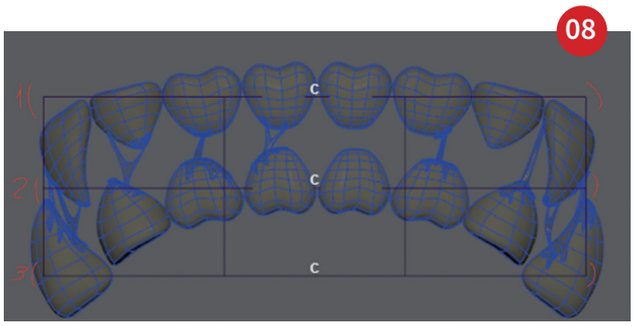


07

2 U divisions

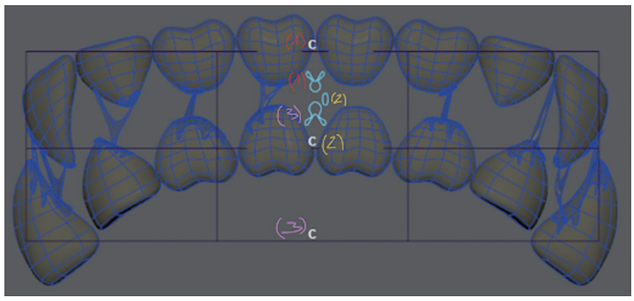


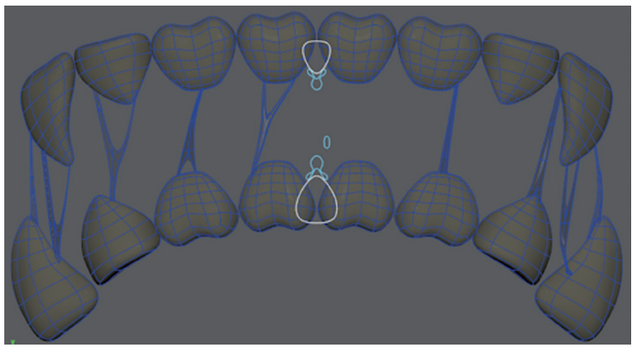
08 **Adding clusters** At this point, we need controls to drive the lattice's points so we need middle men: clusters. They are able to drive selected vertices therefore we create three clusters to control top vertices (1), middle vertices (2) and bottom vertices (3). Select ffd#Lattice > RMB > Lattice Point > select the vertices you want to drive > Deform > Cluster and open the option > enable Relative mode (prevents additive deformations) > Apply. Now, group all clusters then parent both clusters and the lattice group to the rig_systems group.

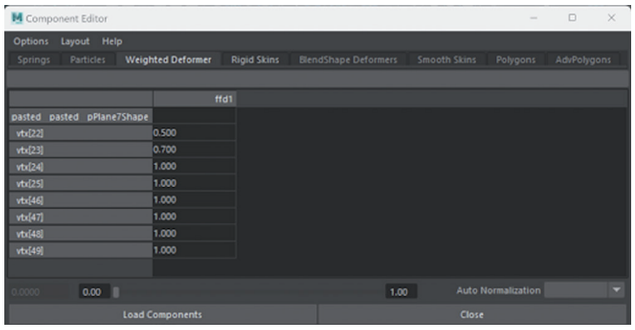


08

09 **Clusters control** Finally, we can create controls to drive the clusters: one for the top cluster (1), one for the middle cluster (2) and one for the bottom cluster (3). Top control (1) will be parented to the upper teeth's control; the others (2, 3) will be parented to the lower teeth's control.
These controls will allow Animators to tweak saliva's animation. Below is an example of how constrained saliva works. Eventually, we can polish the cluster's influence up for each mesh vertex by going to Windows > General editors > Component Editor > Weighted deformer.



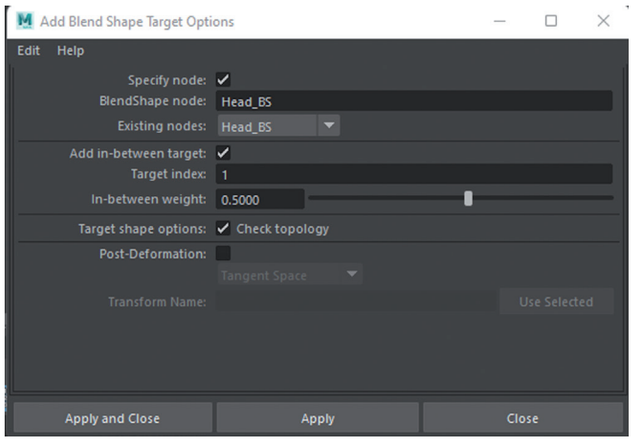


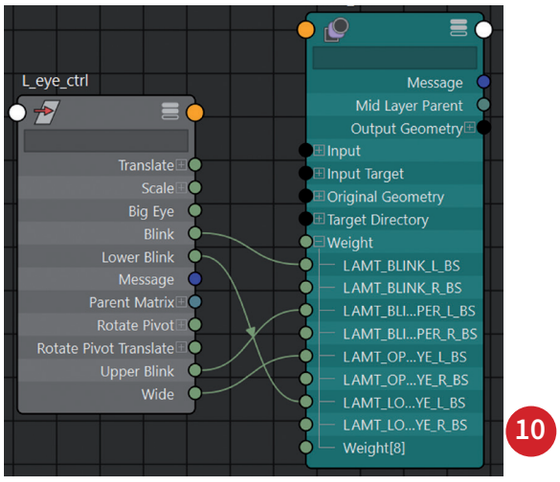


09

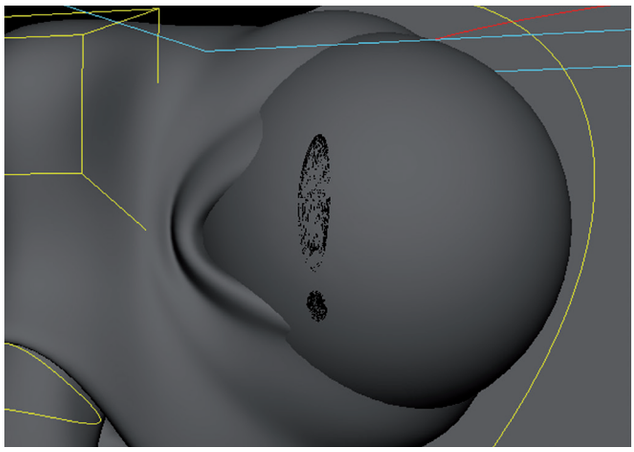
Blendshape system
using the Node Editor

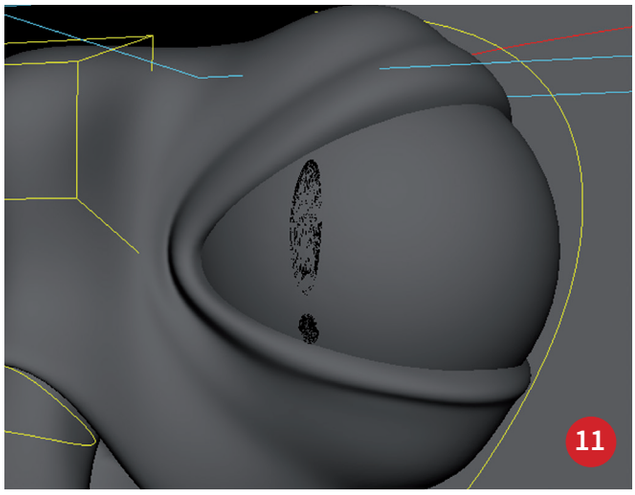
10 **Adding blendshape** We won't walk through modelling blendshapes yet we'll show you how to add them to a facial rig. Start by importing all your blendshapes and grouping them into the blendshapes group (under the DO_NOT_TOUCH group). Always remember that you can centre pivot and delete history, but do not freeze transformations since you need to know the blendshape offset from the main model. In order to add a blendshape, follow these steps: select the blendshape > Deform > Blend Shape and open the option. Now, we add the blendShape node name then go to Advanced and set Deformation order as Pre-deformation. By doing so, we ensure that blendshape get applied after the skincluster. Now, add attributes to the corresponding controls, and connect them to the corresponding blendshape weight from the blendshape node.



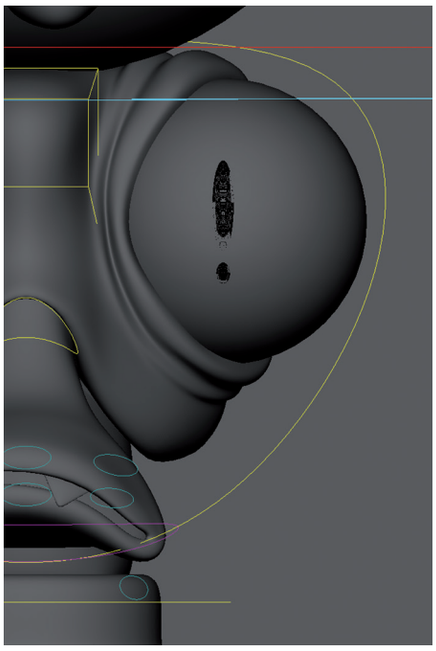


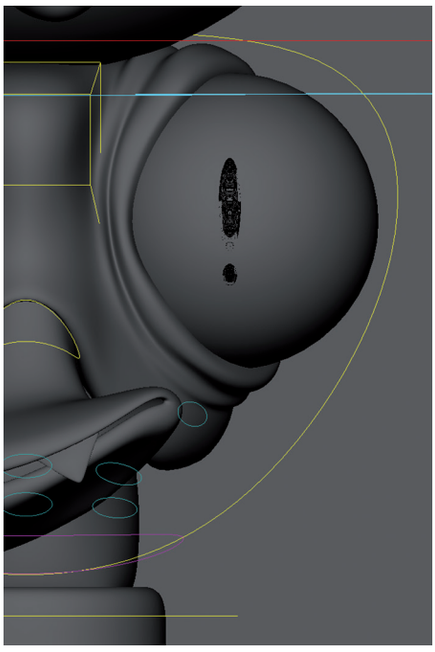
11 **Fixing the blink** When adding the blink blendshape, it might happen that halfway the eyelid intersects with the eyeball mesh. To solve this issue, we can add a halfway blendshape. Since it is a corrective deformer, we don't want the Animator to directly control it so we need to trigger it off the blink attribute. So, go to Deform > Edit Blend Shape > Add and open the option. Enable Specify node and select the blendshape node which includes the blink, check Add in-between target and set In-between weights to 0.5. To identify the correct Target index, we open the blendshape node and we write down the index of the blink (indexed from 1 to length of the weights list). Finally, select the middle blink, then select the base mesh and click apply. As we can see, the blink now works correctly.



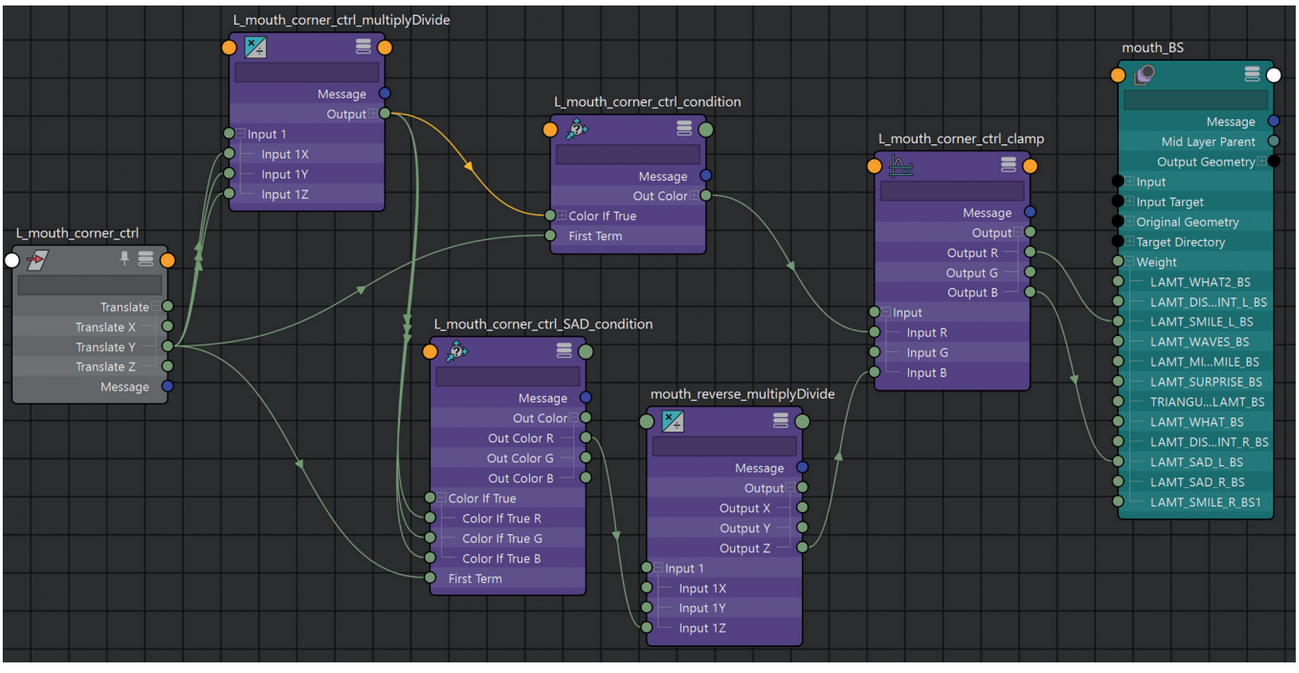


12 Side smile setup Let's set up the mouth's smile blendshape. Indeed, we want both corner mouth controls to pull the corner of the corresponding corner of the mouth around, so when the control goes up it triggers the smile. First, open the node editor and bring the control and blendshape node in. Create a condition node to check if the control is moving up. Indeed, we can check it by comparing the control's y translations to the second term, 0. Basically, we set the operation to Greater Than, Colour if false to 0, 1 instead. In other words, when the control is moving up the blendshape weight will be set to 1. The control is now triggering the smile. Besides, we can improve our set up by adding a multiply divide if the corner control moves far too quickly. Also, the control has no limit when moving on its axis so the Animator could easily exceed the maximum blendshape weight value, one, destroying the mesh. So, we can make the Animator's life easier by adding a clamp node in order to force the result into the weight value's range (0, 1). Actually, we could go through setting up the frown (y downward) and other common blendshape but it would be quite the same process.



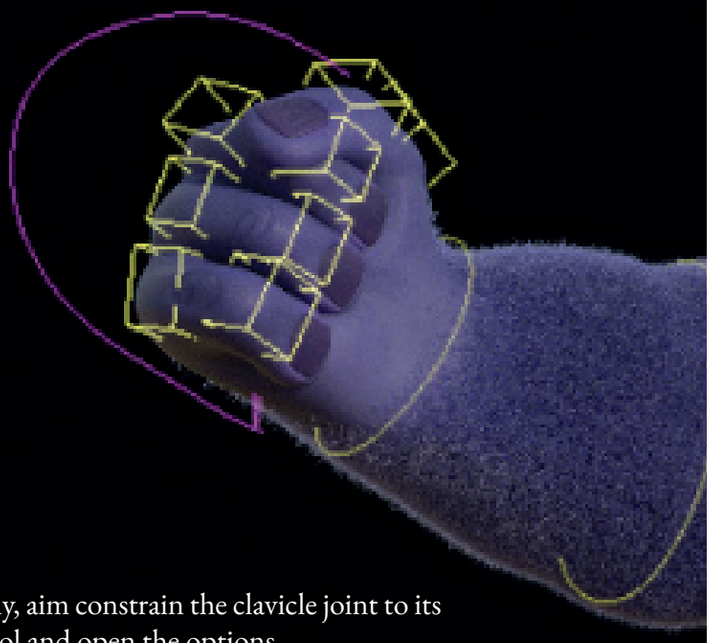


12



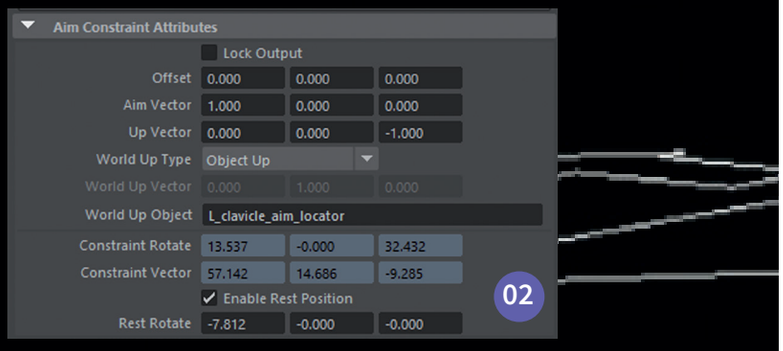
P3 One step further
Complex rig

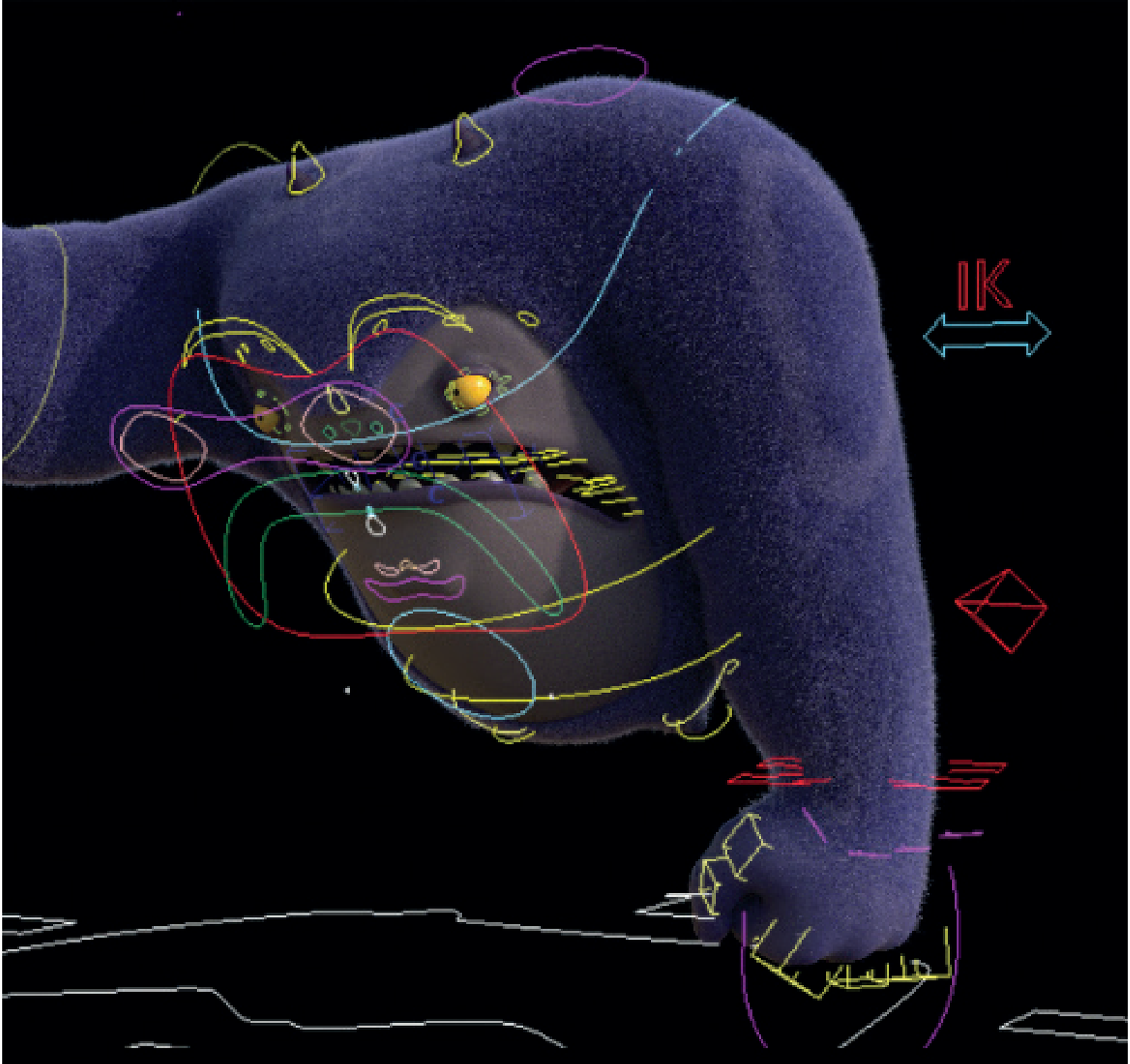
01 **Advanced rig** We have finally built a working basic skeleton with a complete facial rig. At this stage we might want to It's time to add some extra features and improvements to this setup in order to make it ready for animation.

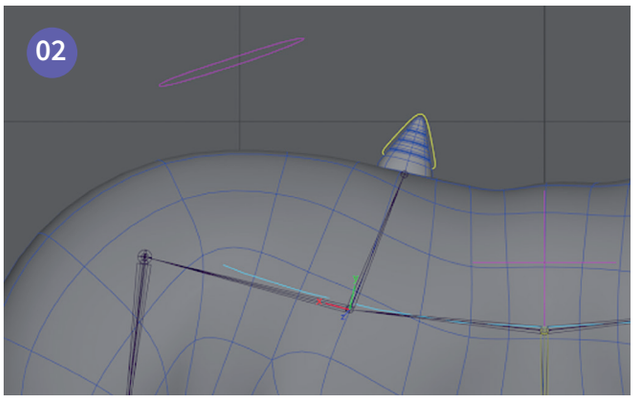


Clavicle and twist Joints
Increase Mobility

02 **Advanced Clavicle Setup** The common clavicle setup is usually all we need to correctly drive our rig. Sometimes however it's not enough.
Especially when it comes to extreme shoulder poses such as ball throwing or some sort of alien gorillas handstanding. It might also happen that if you raise the last spine control the humerus follow but the clavicles don't, ending up with an extended neck.
To improve this setup we need to focus on the clavicle orientation. For instance, if we move the clavicle control up it might seem that the clavicle joint is following yet its orientation doesn't update.
First, constrain parent the clavicle to the upper IK spine control (Maintain offset enable and disable Rotate). Therefore, the clavicle moves with the upper IK spine control. Let's work on the rotations.
To avoid any rotational issues or joint flipping, create a locator and match it to the upper IK spine position. Then, align it to the clavicle's z-axis and rename it to clavicle_aim_locator. It should move with the torso so parent it to the upper bind spine joint.
Finally, aim constrain the clavicle joint to its control and open the options.
Reset settings then set the aim vector to the primary axis, change World Up Type to Object Up so we can use the locator's position and add its name World Up Object. The Up Vector matches the axis which will be pointing at the locator then click apply.
It might happen that rotations would be applied to the clavicle joint; adjust the locator's position to remove them. Now, when we move the clavicle controls the joint actually rotates adding nicer deformations to the mesh.

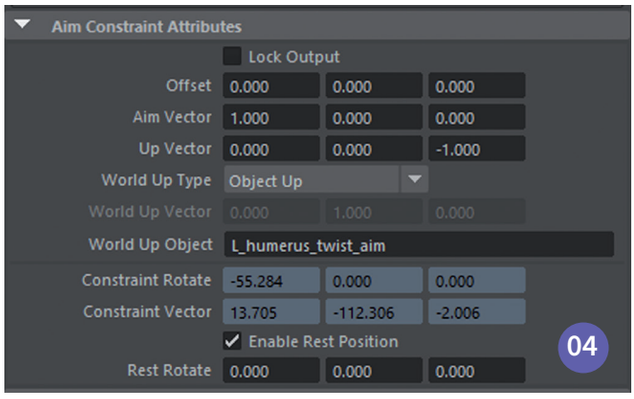



IK



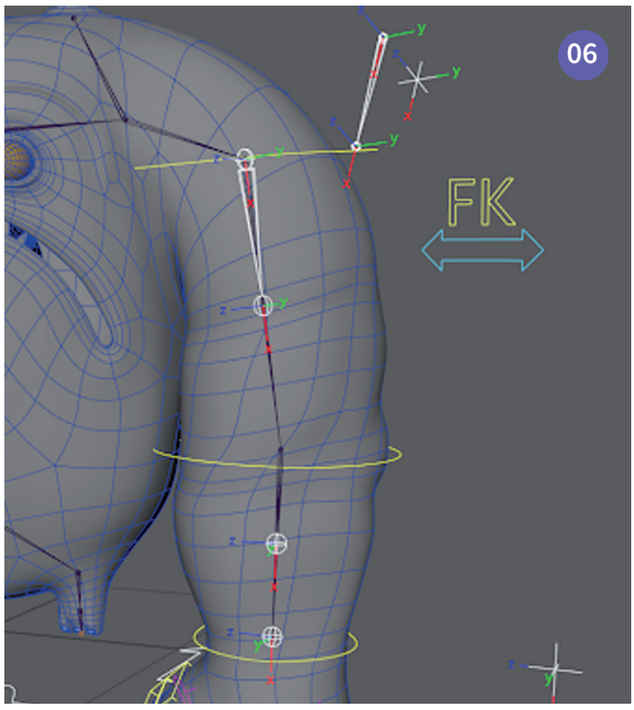
03 **Arm twist joints** At this stage, limbs rotate badly when rotating on the primary axis. Plus, this is even more evident on characters with a lot of volume on their arms or legs.
Moreover, some walking styles like the knuckle-walking emphasise this artefact. We will focus on solving this issue on the arm yet you can repeat the same process for the leg. Start by switching to the FK mode then duplicate the humerus joint twice.
Delete everything beneath them, rename them to humerus_twist_01_skin and bicep_twist_01_skin, parent the second to the firs joint then parent the humerus_twist_01_skin to the humerus joint.
To place the bicep joint precisely in the middle of the upper arm do as follows: select the humerus, then the radius joints, the bicep joint and finally Constraint > Parent (Maintain offset and Rotate off).
Notice that the bicep joint moved in the middle since it tries to stay between the humerus and radius joint. Delete the parent constraint.

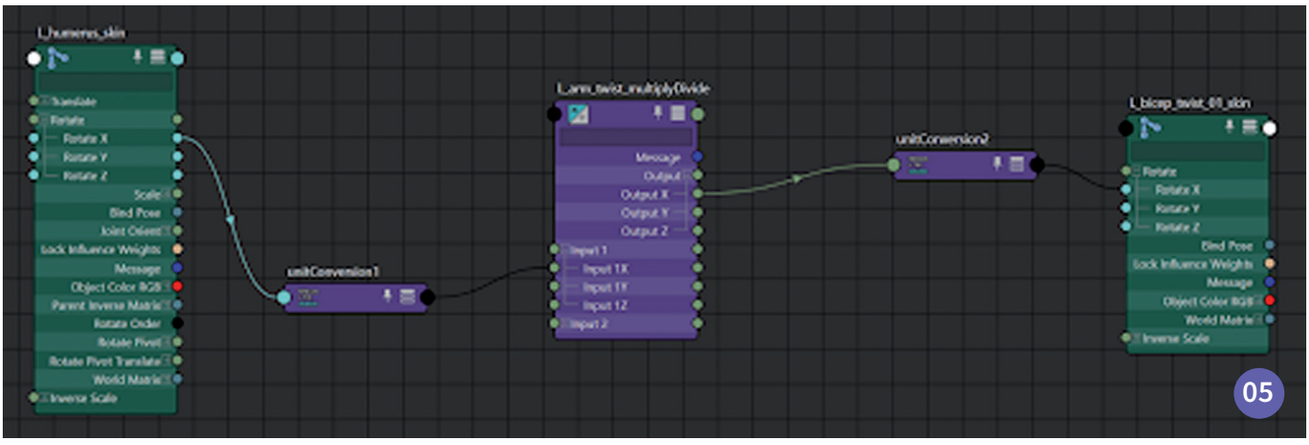
04 **Solid shoulder twist setup** Now, turn on the rotational axes and create a locator called humerus_twist_aim.
Use it to lock the x axis on the twist joints since we want it to stay still as the humerus control is moved. Match its position and orientation to the humerus joint then move it behind the arm, along the z-axis. Therefore, the humerus joint and the locator should be always aligned on the z-axis.
We just need to create an aim constraint: reset the options, the aim vector will be the humerus twist joint's axis pointing down to the elbow (since we still need it to rotate with the rest of the arm), change the World Up Type to Object Up and put the aim locator into the corresponding box.
Notice that the joint's z-axis is pointing forward therefore negative z is the Up Vector. Finally, aim constraint the humerus_twist_01_skin to the radius joint.
Currently, even though we can rotate the x-axis only by moving the locator, there are poses where the twist joint flips. To get around this we need to make the locator move with the arm so it can automatically adjust the twist joint x rotations.
Duplicate the humerus_twist_01_skin, move it behind the arm and out of the hierarchy then rename the joint chain to end with "_follow".
Now, parent the aim locator to the humerus follow joint then add an IK handle (Rotate-Plane Solver) between the two joints, finally parent and snap it to the radius joint.
However another problem arises: the x-axis is no longer locked because the IK handle twists with the arm. Fix it by setting the Pole Vector attributes on the IK handle to 0.
As you can see, the arm starts to twist at extreme angles but it doesn't flip anymore. You could play with the position of the locator and test if you can reduce the twisting even more.



05 **Bicep twist** In order to get a more realistic arm twist, there should be a gradual blend between the shoulder and the bicep rotation.
Basically, we need to reduce the amount of rotation (multiplyDivide set on multiply, input 2 = 0.5).
We're not done yet. Indeed as we twist the arm the bicep seems to get slower the more we twist.
We need to set the correct Rotate order to both humerus twist and bicep twist (otherwise the joints may flip); here it would be zyx.

06 **Wrist Twist** Setting up the twist joints on the lower arm is a bit simpler.
Duplicate the radius joint twice, delete everything beneath the duplicate joints, rename them radius_twist_01_skin and wrist_twist_01_skin and parent them to the radius.
Match the wrist twist position to the wrist joint and move the radius twist in the middle of the arm (just like we did for placing the bicep).
Again, we need to lock the twist joint's x rotations. Create a locator called wrist_twist_aim, match it to the wrist twist then move it behind the arm, along the z-axis. Parent the locator to the bind wrist joint then aim constraint the radius twist to the wrist joint.
The final step is adding the radius twist; we only need to repeat what we did to set the bicep twist.





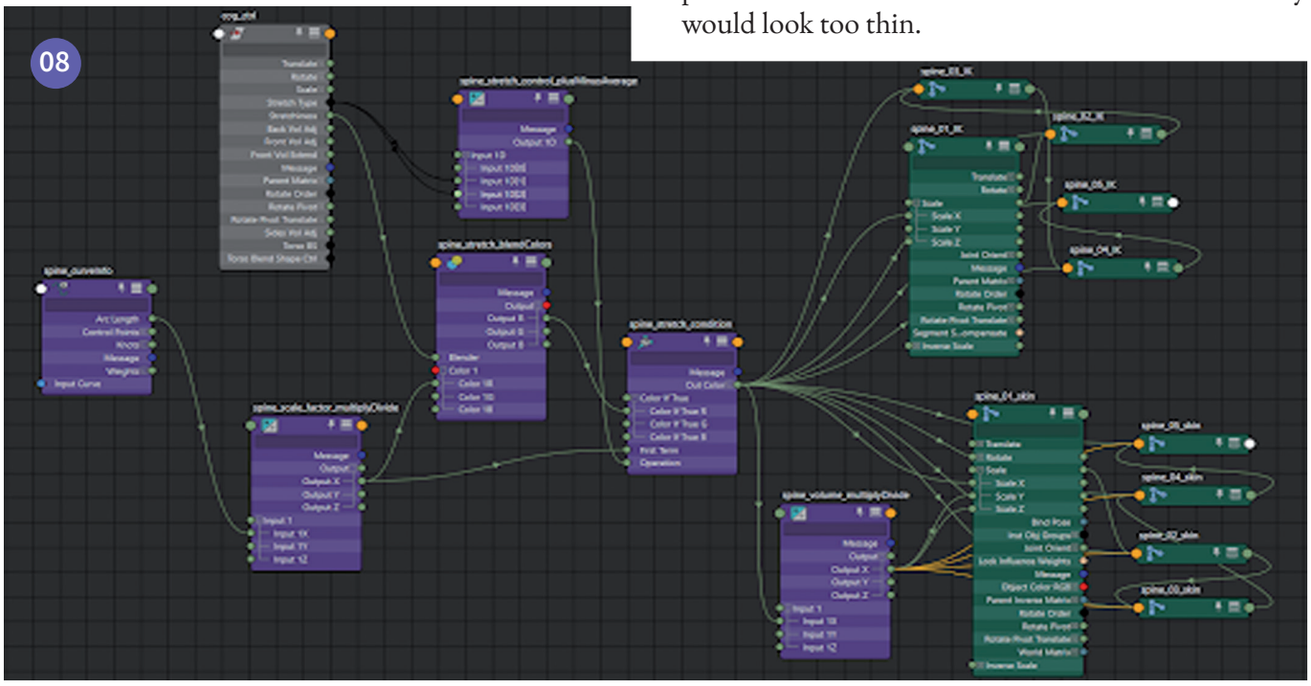
Squash and stretch

Make the character flexible

07 **Using utility nodes** When we think about cartoon characters, amazing shape deformations immediately come to mind. And we can bet it happens also to Animators. With our basic rig all set up it's time to learn how to implement squash and stretch. If not set correctly, it might cause scaling issues and other annoying headaches, use with care.
For this tutorial we are going to use the utility nodes rather than expressions since they are faster than the latter. These nodes can be accessed by opening the Node Editor (under Windows), which can be described as an extension of the Connection Editor.
Indeed utility nodes allow adding operations or conditions on the attributes.
Basically, it's like visual programming where nodes are treated as entities, connected by arrows which represent relations.

08 **Spine squash & stretch** We can use the spine curve to get the current spine length. Open the Node Editor, select the curve and run the CurveLen script *(page 48, 02)*. A curveInfo node will suddenly pop up in the Node Editor. Use the Arc Length attribute to check whether the spine is stretching. So, we add a multiplyDivide (operation: Divide) to compare the current length against its base length. Basically, the result will be 1 when there is no stretch since input 1 is equal to input 2 (constant, it's the base length). Therefore, use this output to adjust the scale of the joints. But we also want to allow the Animators to turn the stretchiness on and off as needed. So, let's bring a blendColors node in and pipe the Stretchiness attribute (float 0,1) from the cog control into its Blender attribute. Now, set all blendColors attributes to 1 because that's the default scale value for each non stretchable joint. Then, create a condition node to check if the spine is stretching, squashing or both: the Animator can decide the spine stretch type, therefore the condition node operation changes depending on this choice. Set the second term from the condition node to 1. So, when stretch type = Both, operation = Not equal; if stretch type = Stretch, operation = Greater or Equal; stretch type = Squash, operation = Less or Equal. Then, the operation attribute should correspond to the operation's index in the drop-down menu. Add a plusMinusAverage node and pipe the stretch type attribute into input 1D[1], input 1D[2] and set the input 1D[0] to 1. Basically, we are using the stretch type indices to get the correct operation index. Finally, connect Out Color R from to scale x from the IK and bind joints since x is the axis pointing down each joint. Do not pipe the hip and the neck joint because this would affect both the pelvis and neck. At this stage, the spine behaves correctly yet it doesn't maintain its volume. Add a multiplyDivide node and set the operation to Power. So, this node outputs the value in Input 1 multiplied by itself the number of times of Input 2. Set the Input2X to -0.5 because the scale x value should get smaller the more the torso stretches. Since the skin joints are the only influencing the model, connect outputX to scale y and scale z from

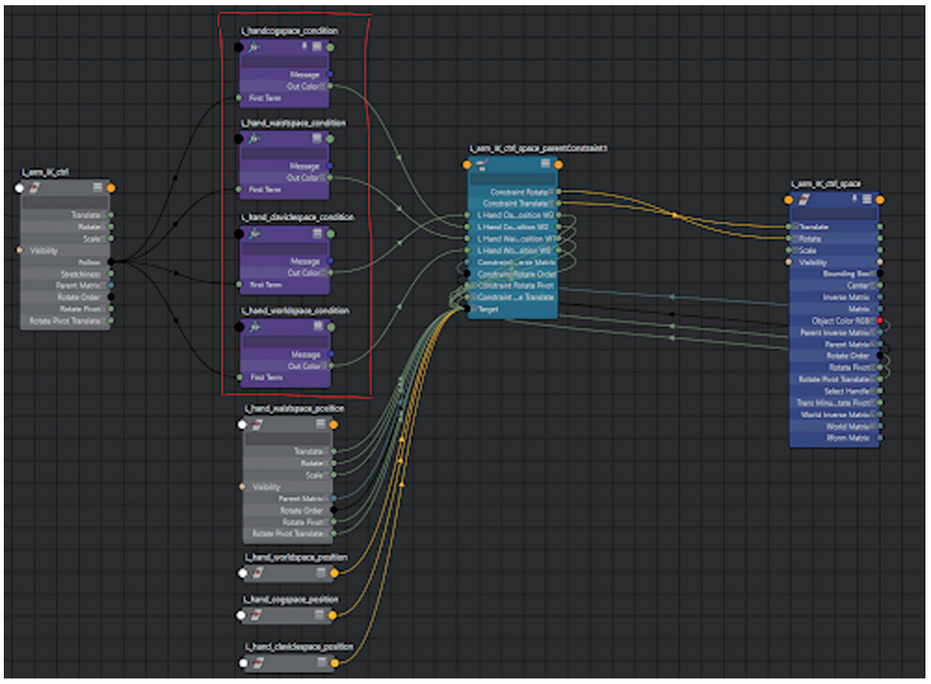
09 **Tips on arm squash & stretch** Adding this feature to the arm is very similar to the spine. To make the arm stretchy we need to keep the wrist always locked to the IK control. Currently, when the IK control moves away the arm doesn't follow. So, we need to know how long the arm currently is and its base length but remember: there is no curve to get this value from; the arm is slightly bent. Therefore, you can use the distance tool to measure the upper and lower arm lengths. By summing these values you can get the actual arm length. The same reasoning applies to the leg. One more thing before we move on: add squash and stretch also to twist joints yet do not add volume preservation to the humerus or femur one or they would look too thin.



Space swapping
Adding dynamic parenting

10 **About** Space swapping, or switching, is dynamic parenting. In other words, allows the animator to swap an object from a parent to another. When the object is in one of its parent's spaces, it follows the parent's movements and is no longer influenced by its other parents. In this section we will set space switching only for the left hand since each object follows pretty much the same implementation. First, add the Follow enum attribute to the arm IK control so we can swap between World, Cog, Waist and Clavicle spaces. To avoid limiting the hand animation by adding constraint we need to create a group holding the space swapping information. Add a _space group beneath the control's offset group. Now, we can get into the actual implementation.

11 **Space Locators** Set Follow attribute to World then create a locator for each of the controls the hand will follow; cog, waist and clavicle control. Rename each locator to L_arm_cogspace_position and so on. Moreover, create a locator and match all its transforms to the arm IK control: it will store the hand's current position. Parent constraint the hand to all these locators (maintain offset disabled). As we can see, the hand moved trying to sit between all of the locators. Set cog, waist and clavicle weights to 0 in such a way that the hand will move back to the only non-zero weight; handspace_position. Snap each one of the other constrained locators to the position of its corresponding control. Furthermore, parent each locator to the relative control then freeze its translations.
Open the node editor and use condition nodes to set constraint weights correctly.
Basically, each condition node decides if the current space is the one it's relative to. Every option in the Follow's dropdown menu is indexed according to its position; 0 World, 1 Cog, 2 Waist, 3 Clavicle. For instance, the cogspace condition is true (R = 1) if and only if the follow's index is equal to the second term (which corresponds to the cog's index).
One more thing, the hand currently snaps to the centre of the influencing control so adjust the corresponding locator's position and orientation to a better one (along the thigh instead of letting the hand snap in the centre of waist control when in its space).

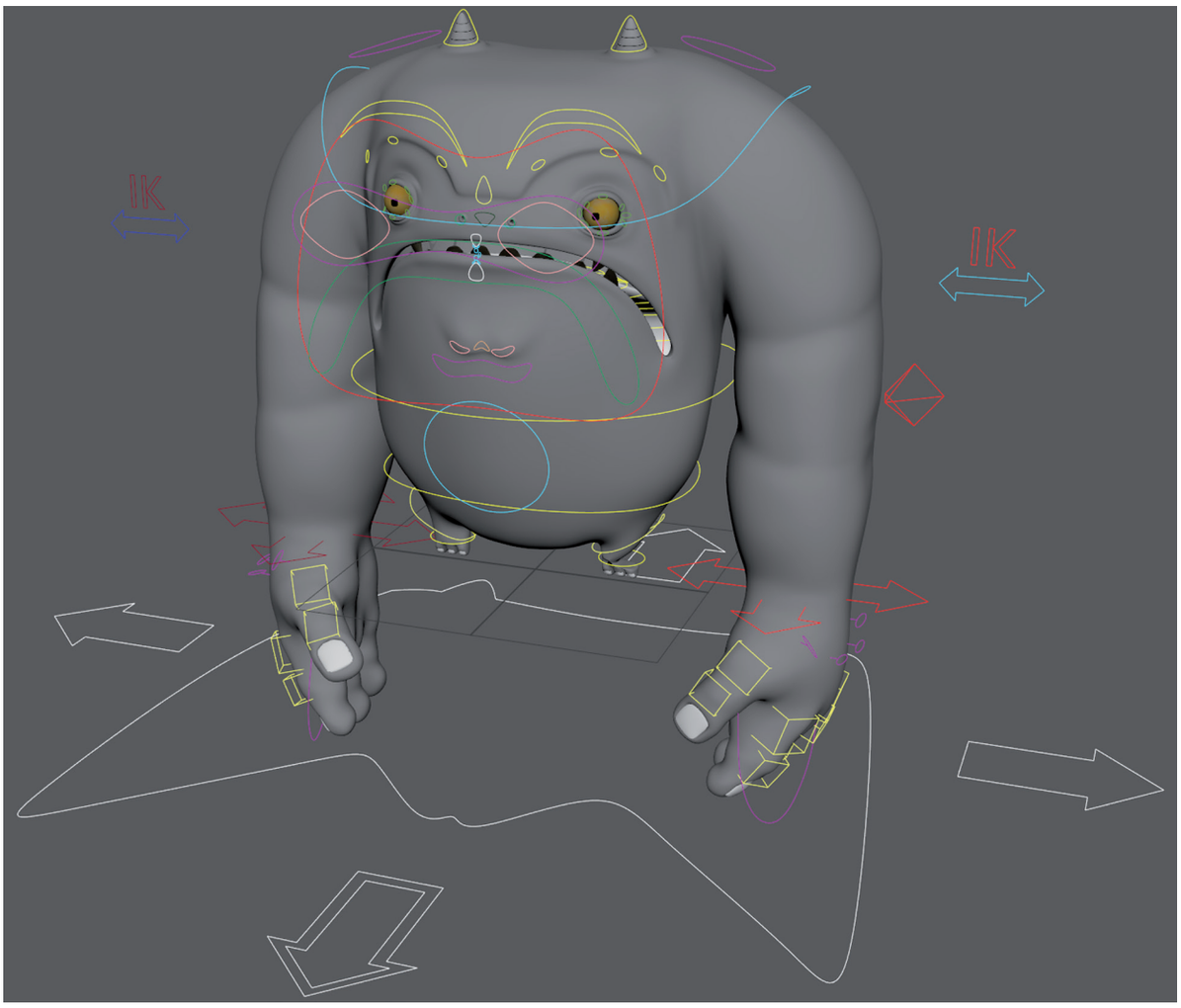


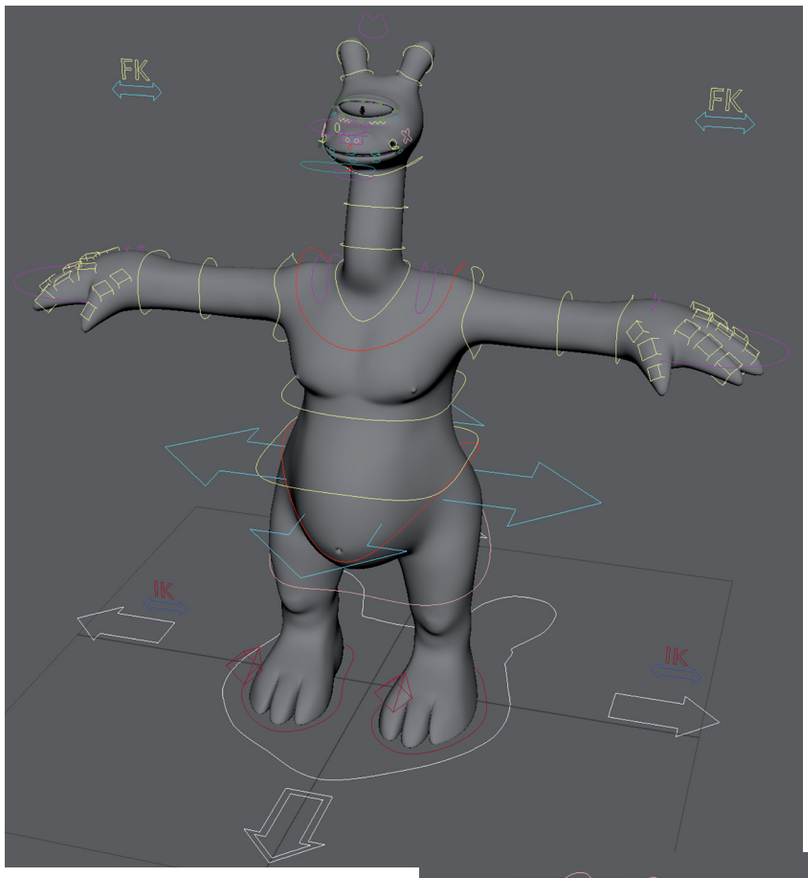
11

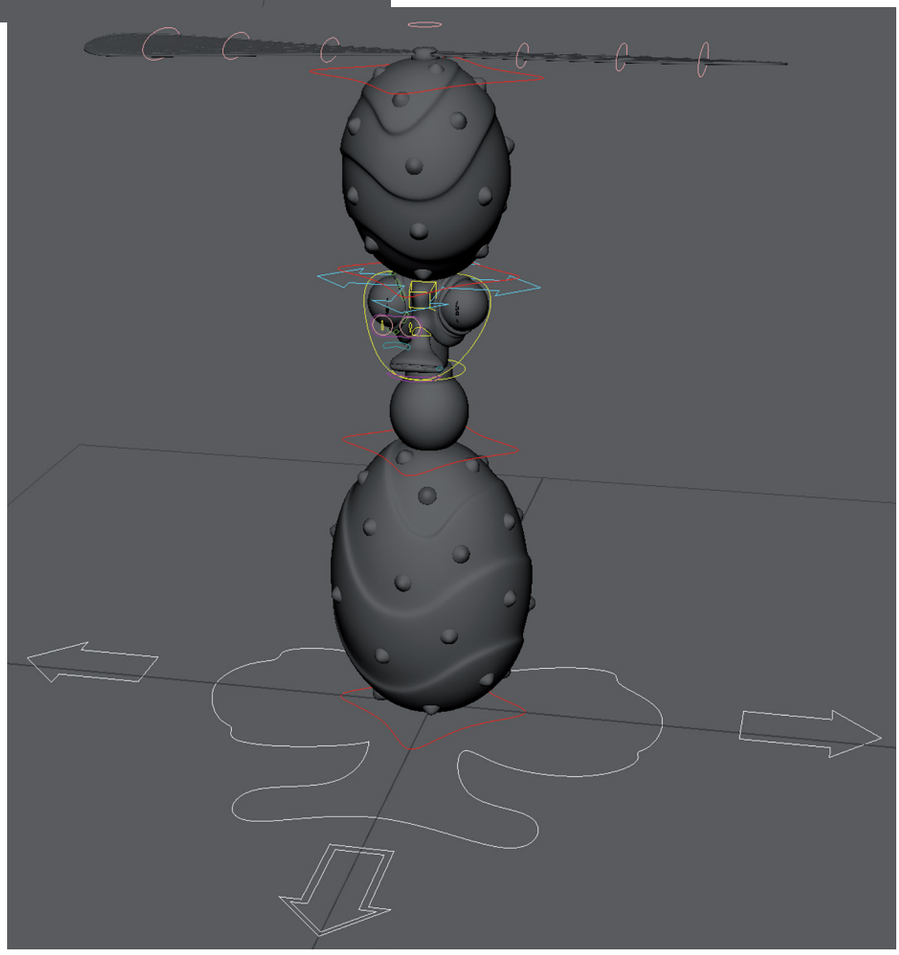
Global control

Getting ready for animation

Before referencing It may look like we have completed our rig yet there is still something to do before starting to reference it. Here are some general guidelines to make a rig file as solid as possible.
Save your rig as an ASCII file since they are not compressed, they allow the users to edit the data inside.
Make sure the outliner is as clean as possible in the rig file, since the Animator cannot change the outliner once the file has been referenced in.
Now, try moving around the character using the root control. If some body elements rotate in the wrong direction, it is probably because of the worldspace_position locator we used when adding the space swapping feature. Ensure it is parented to the root control and repeat the test. Everything should move with the root. Another important thing to fix is visibility; visible elements can also be accidentally selected by the Animators, who could break the rig while animating. Therefore, add attributes to the root control in order to control the visibility of each part of the rig. At the very end we lock off all the offset, space and grp groups so the animators can't touch them. Here, is where being consistent comes in handy. Select the first transform node of the rig and run the LockOffset script *(page 48, 04)*.




FK
FK
IK
IK



P4 Volume control
Adding extra features

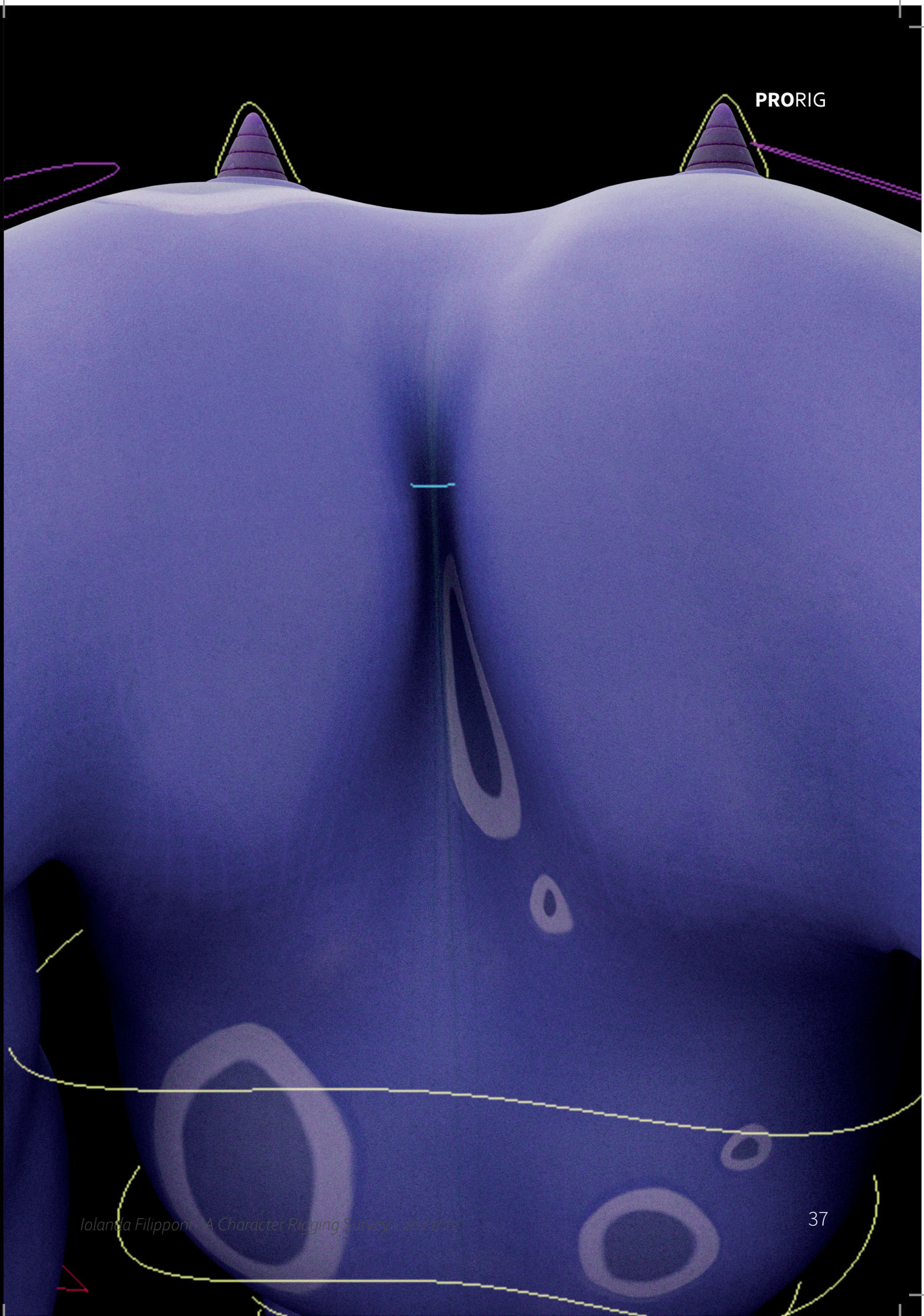
01 **Joint based system** So far we built the basic structure of our rigs yet we can still add some extra features before passing them to an animator. There might be characters slightly tricky to rig because of their volume and structure. But that's okay, we are going to focus on adding muscle and fat deformations to our rig, giving the animator a more anatomically correct setup. Moreover, we'll walk through two different breathing systems and learn how to choose the most suitable for each rig. Please notice that we used a joint-based approach which is perfect for gaming rigs and less heavy than dynamics.

Fat system
Rigging large characters

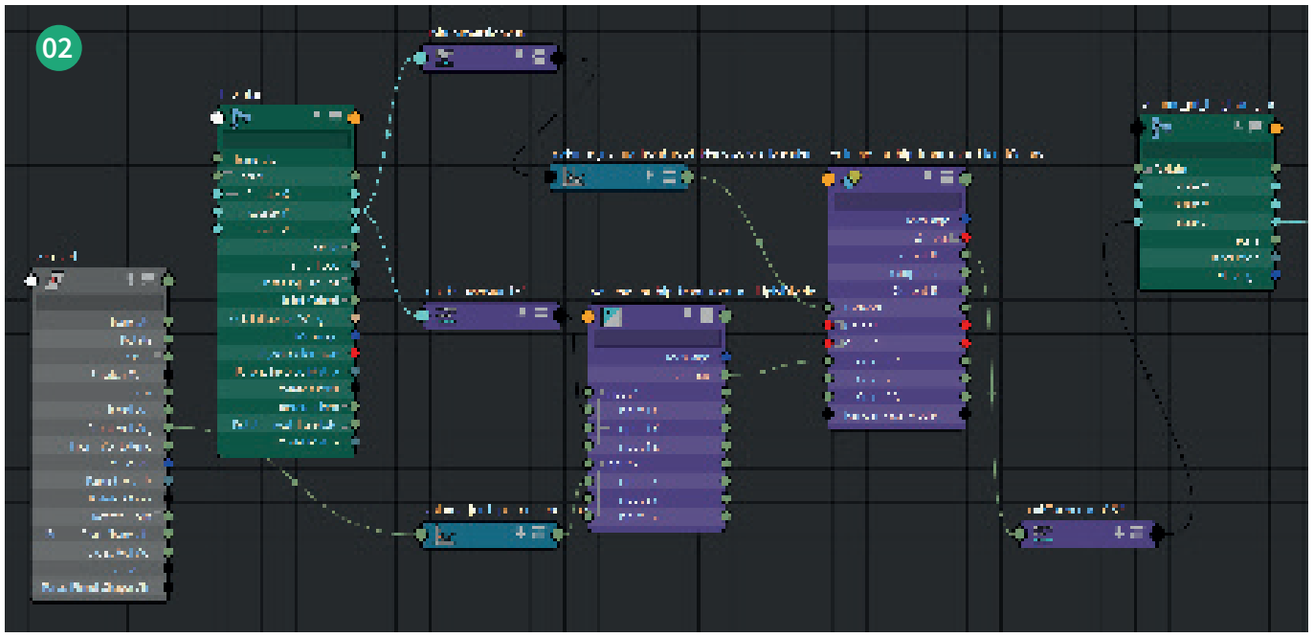
02 **Volume root joint** set up When posing fat characters in extreme poses you can easily get collapsing geometry. To avoid it we are going to use a set of volume control joints hooked up to the basic spine rig. We are going over the actual set up of one of the volume joint chains, since it's the same for all corrective joints around the body. Duplicate the base spine joint twice and create the chain.
Make sure the root joint matches base spine joint orientation while the end joint's x-axis should be perpendicular to the geometry. Bind the end joint only. Add zero-to-ten attributes to the cog control so the animators can autonomously tweak the rotation offset then connect them to a multiplyDivide node. Pipe the output to Color 2R from the blendColors node, so it can adjust the percentage of the rotation offset based on the direction the torso is rotating.
By using the blendColors node we get smooth transitions while switching rotation, the condition node would make the joint popping instead. First, connect the rotation of the base spine to Input1X on the multiplyDivide. Then, pipe the corresponding volume adjustment value, from the cog control, into the Input2x of the multiplyDivide node using SDK. Load the adjustment value as the Driver, and the Input 2X as the Driven. Set two keys based on how large your character is, just test it. Now, connect the output to the blendColors node then use another SDK to control the blender attribute. Finally, connect the output from the blendColors to the volume joint root. At this point, as the character bends forward, the joint starts to slow up.

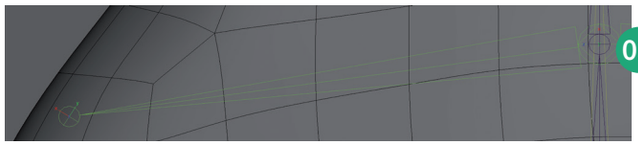
03 **End joint set up** Again, we use a multiplyDivide node to pipe in the corresponding extend values from the cog control, using SDK. Connect the rotate value from the root joint to Input 1X of the multiplyDivide. So, we got the initial extension calculation which we have to plug into the Input 1D[0] from the plusMinusAverage node. Set Input 1D[1] to 1 then connect the output to the condition node. This last node allows us to check if the root joint is rotating and in that case extends the end joint.

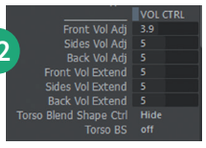
04 **Bouncing belly set up** Last things are some additional tricks to control large character torso using rigs on blendshapes, avoiding affecting the main rig. Place a base joint, one joint for each pectoral and a more complicated joint structure to control the belly. Basically, we create a world oriented root joint and five child joints: belly button, lower belly, upper belly, one for the right side and one for the left.
Now, create a null group, match it to the belly root joint then parent this last to the group. As we need the belly to bounce we create a locator, match all its transforms to those of the belly root joint and move it in front of the belly. Constrain Aim the belly root joint

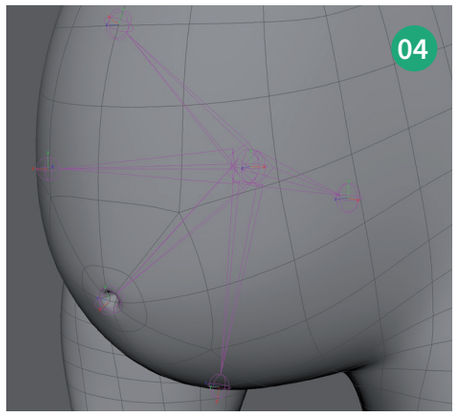


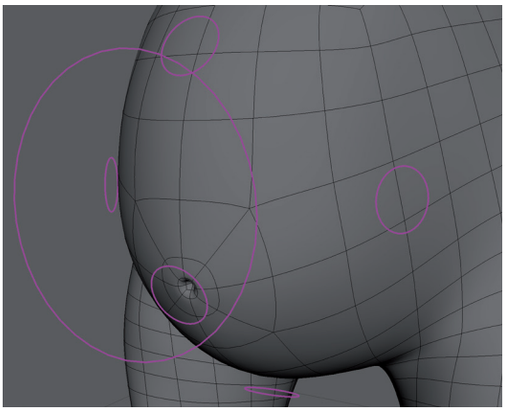
to the locator in such a way that drives the joint's rotations. To add the ability to jiggle, create pectoral and belly controls on the blendshape.
Besides, the Animator should control these joints directly from the main shape. So, directly connect controls to joints and the locator (using the connection editor). Since we think of the controls as the rig's interface we need to keep it simple: add blendshape visibility and switch attribute to the cog control. In the end, parent each blendshape joint group to a main group, whose pivot matches the world orientation.

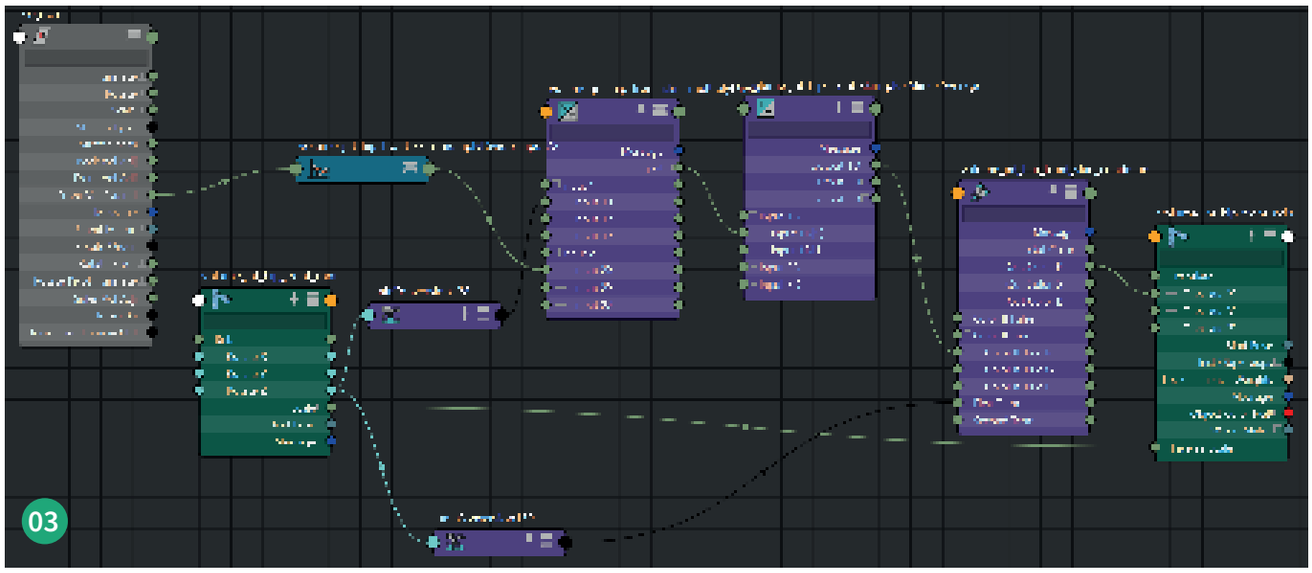


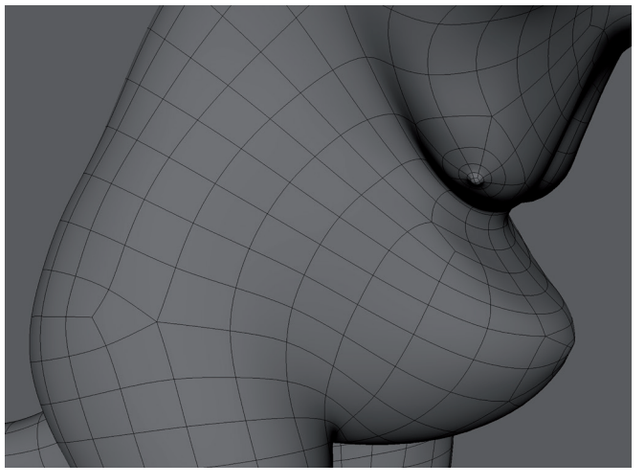


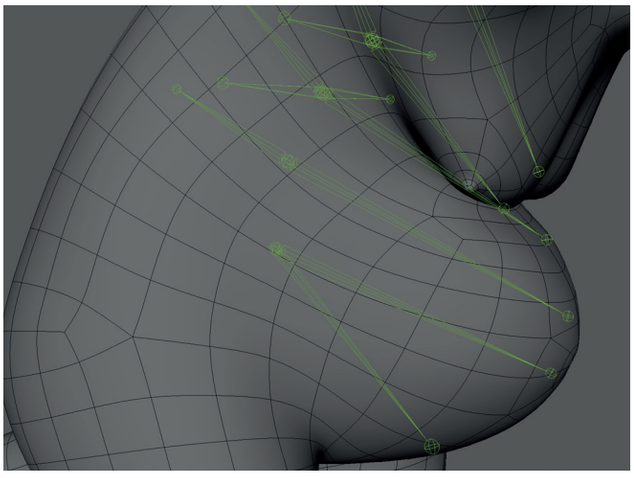












Muscle system

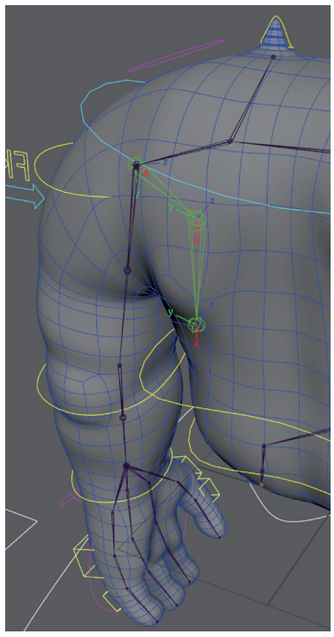
Add strenght to your rig

05 **Create the scapula** In this section we are going to set up a joint based muscle system for supporting the whole shoulder and back movement. First build a two joint system to enable scapula's rotation around the ribcage. Snap the root joint to the humerus then rotate it to place the second joint where the outermost point of the scapula will be. Rename these joints to scapula_roll_root and scapula_temp (this is a placeholder; we'll delete it later) . Now, create another two joint chain and snap the root to scapula_roll_root. Place the end joint right under the root, then move it to the lowest point of the scapula and push it a bit inward. Rename these joints to scapula_01 and scapula_02 then delete the scapula_temp and parent scapula_01 to the scapula_roll_root. Now, group these joints and rename it to scapula_grp then parent it to the muscle_system group in the rig_systems transform. Finally, constraint parent scapula_grp to the humerus joint to make the scapula follow the arm.

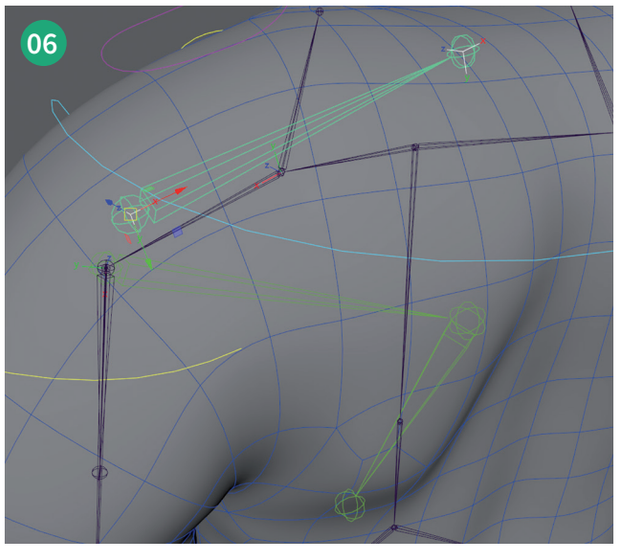
06 **Add the trapezius** Let's set up another fundamental joint chain: the trapezius system. Create a two joint chain and place between the hu-

merus and upper spine joint. Group the chain under the trapezius_grp, which is in the same position of the root joint, then constrain parent it to the clavicle joint. Now, we want the trapezius joint to always aim towards the neck (or the upper spine joint if the character is neckless) so create a locator to use as the World Up Object of the aim constraint. Repeat it also on scapula_01 and scapula_end in order to make it actually rotate around the back.

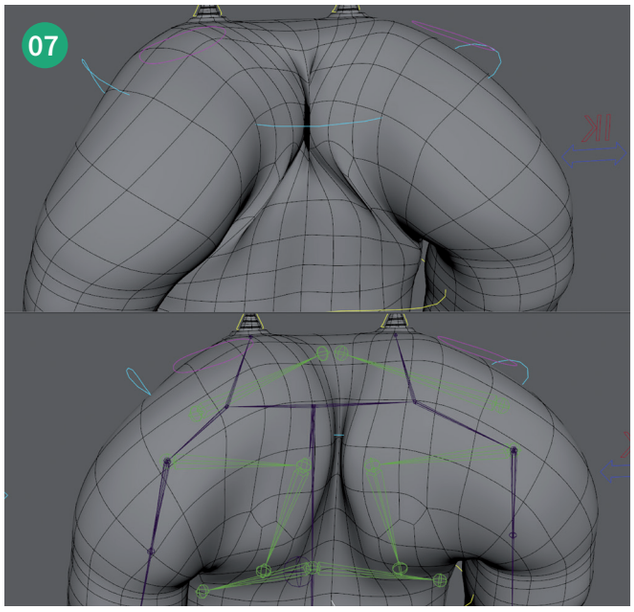
07 **Movement adjustments** After setting up the scapula on both sides we face an issue: when the character moves its shoulders back, scapulae intersect instead of colliding. Moreover, when the clavicle rotates forward the scapula should come out a little bit. On the other hand, the scapula should be pushed in when the shoulder moves backward. Basically, we can use the rotate y of the clavicle joint to drive translate x of the scapula_01 joint but we need to specify a range of motion for translate x. Therefore, open the node editor and create a setRange node, connect rotate y of the clavicle to Value X and Value Z from the setRange node. We will use X for the backward motion; Z for the forward one. Set Old Max X to the maximum backward rotation of the clavicle. Old Max Z is clearly equal to the maximum forward rotation of it. Min X should be set to the default value of the translate x whereas Max X will be smaller. Indeed, to push the scapula in we need to reduce its value. When the shoulder moves forward, it's quite the opposite: Max Z is equal to the default value of translate x, while Min Z is the maximum outward translation of the scapula. Finally, add a condition node that checks if the clavicle is rotating backward or forward. According to the result translate x will be replaced with Out Value X or Value Z from setRange.



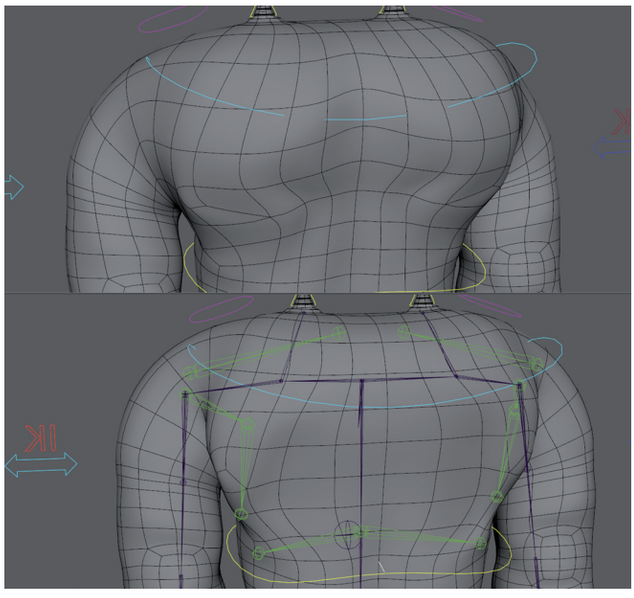
05

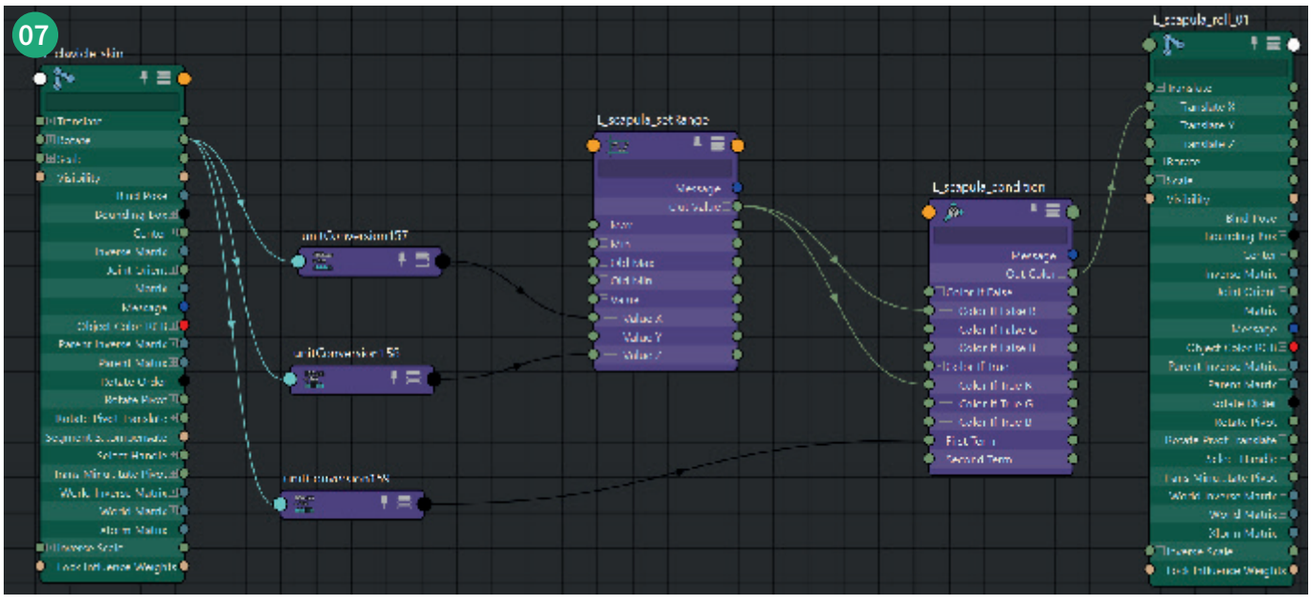


06



07

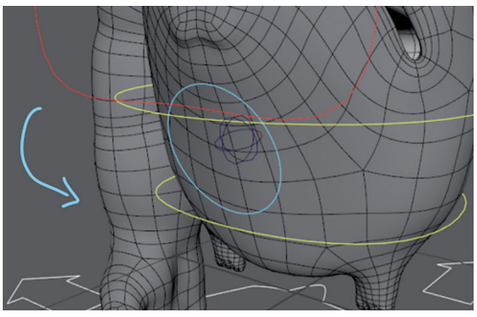


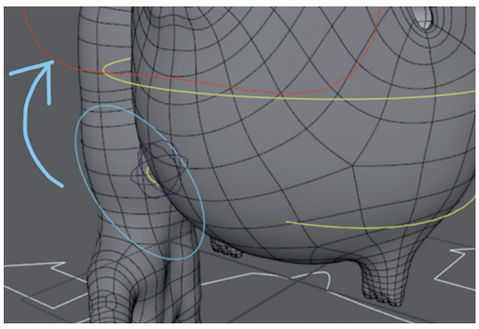


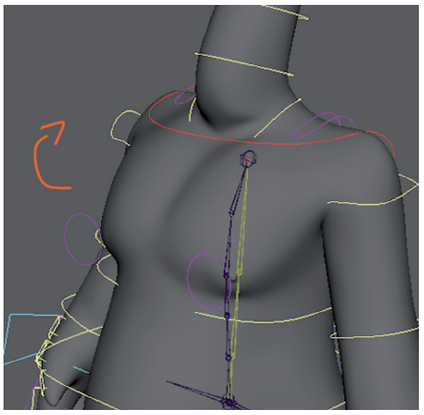
Breathing system

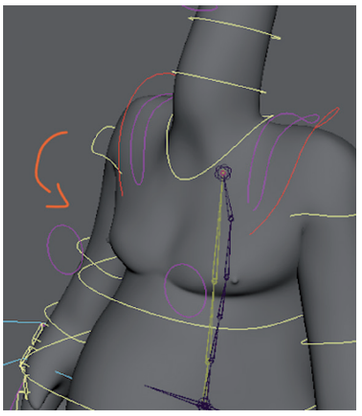
Add breathing support

08 **Breath joint** It might sound basic, yet being sure our rig allows proper breathing is one of the most important things to do. A set up relying on a spline IK based spine already supports proper breathing, unlike an FK spine. Therefore, a simple yet efficient solution is adding an extra joint. Create a control and constraint the joint to it. Since we only need to expand the belly to simulate abdominal breathing, we can leave translations unlocked only. Finally, parent the breath control to the spine FK control while grouping the breath joint to the skeleton group.









Step by step

Rigging simple props for animation

“This tutorial will focus on building basic yet efficient rigs for props focusing on dynamic parenting and secondary animations. As well as the character’s rig, these setups are all scalable.”

Egg and pillow rig

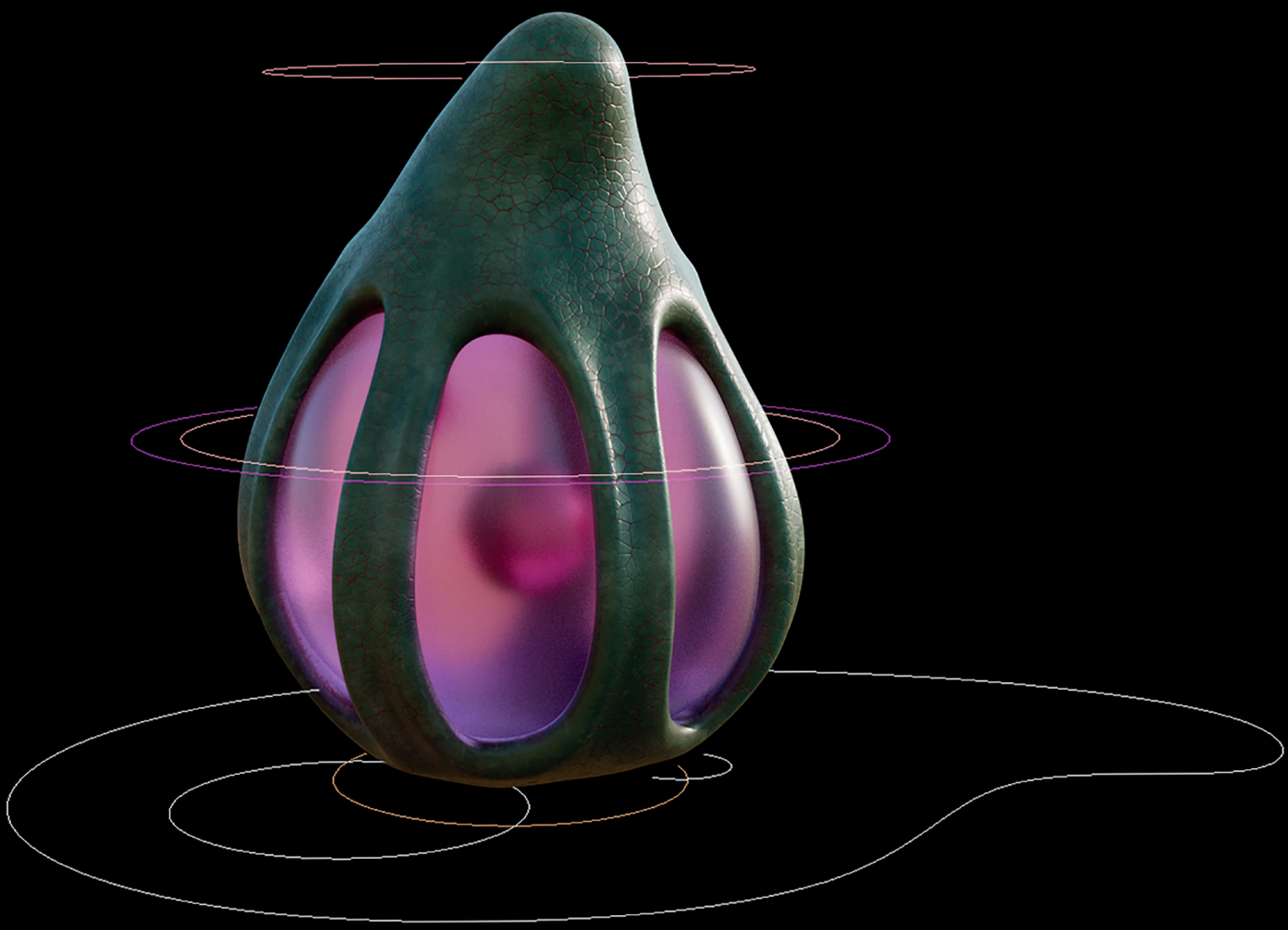
Squash and stretch using deformers

01 **Egg rig** Let’s start from an egg that will play (literally) the role of a soccer ball. We create a lattice (divisions: 2, 3, 2) then RMB > Lattice point and create three separated clusters (Deform > Cluster with Relative mode turned off) for each lattice T division: top, middle and bottom deformers. To create a satisfying squash and stretch effect we need to measure the distance between the top and bottom cluster. So, create > measure tool > distance tool and snap one locator to the top and the other to the bottom. Parent the cluster to the corresponding locator, add controls to top, bottom locators and middle cluster (since it has no locator attached). To keep the middle control, you guessed it, in the middle we constraint parent (maintain offset enabled) it to both top and bottom controls. Finally, open the expression editor (which can be found under Windows > Animation Editor > Expression Editor) and type the following code:

```
mid_cluster.scaleX = 44.083/up_bot_
distanceDimensionShape.distance;

mid_cluster.scaleZ = 44.083/up_bot_
distanceDimensionShape.distance;
```

Where cluster_distance is a number, which is the distance between top and bottom cluster before any transformation was applied; distanceDimensionShape is the shape node of the distance tool. Here, the y-axis is the up vector so we scale along the x and z axes to modify the egg’s width. Add the root and main control and eventually the squashable egg is done.

02 **Pillow rig** Try to think about the flour sack setup and make it simpler. Indeed we can place a basic skeleton: a chain for each angle (3), a joint to control the middle area (1) and volume joints (2). Then duplicate the pillow and add squash and stretch to it, just like in the egg rig. Use the connection editor to make top and bot controls drive the locators’ translations. Finally, add the copied pillow as a blendshape and create controls for each joint with the NewControl script (5.1.3).



Suitcase, Sunglasses and Cocktail rig

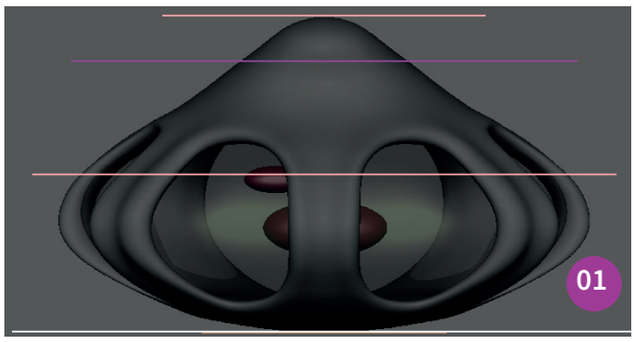
How to set dynamic parenting

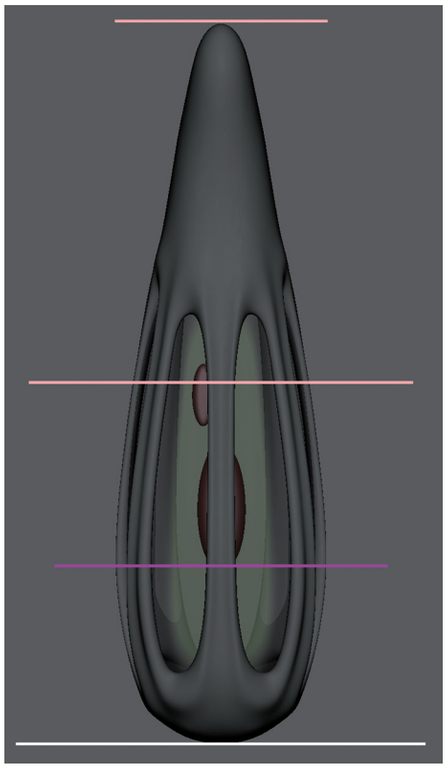
03 **Dynamic parenting** For each of these rigs, the Animators need to easily constrain them to some character's body part without compromising extra animations on the prop. Therefore we create a locator and constraint it to the root control. Then we add the BlendParent attribute (select all pale-blue eyelighted attributes in the Channel Box > shift + W > shift + E) to the latter and rename it Driven. Now, the Animators can happily blend between following or not the locator. For instance, if they want to make the sunglasses follow an alien's head they have to: snap and constraint parent the driver locator to the head control, then set Driven to 1 and place the sunglasses in the correct position. But, as long as the driver constraint is active the Animator won't be able to animate the root, because it will be following the driver locator. For this reason, we create other controls to allow extra prop's animations. Since we don't want the Animators to directly edit the rig we group the driven locator outside the controls and DO_NOT_TOUCH group. Thanks to this setup the prop inherits the character's animation without losing its own.

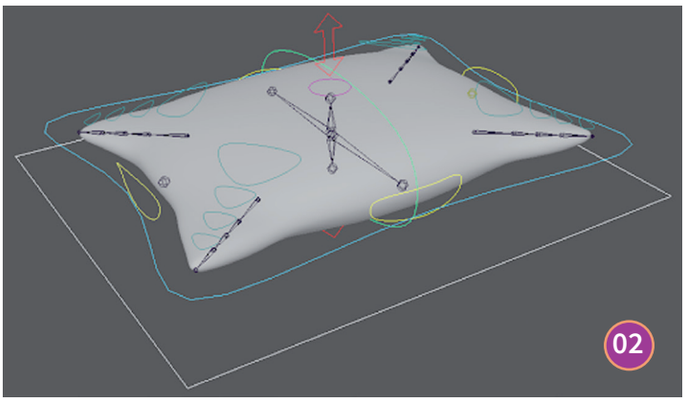
04 **The swinging suitcase** The main control (2) on this rig allows swinging the suitcase even though it's inheriting the hand's movement, for example.

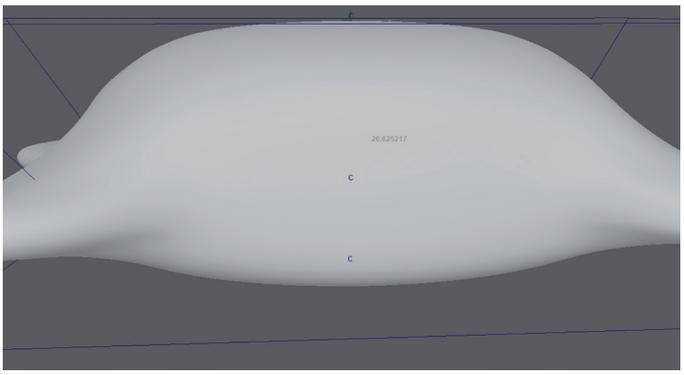
05 **Alien sunglasses** Another benefit of this set-up is that it still allows secondary animations: we can move the whole glasses by keying the main control (5), whose pivot is placed at the end of the temples to facilitate the sunglasses' bounce.
Last but not least, we can tweak the glasses by adjusting the temple (2, 3) or moving the lenses (4).

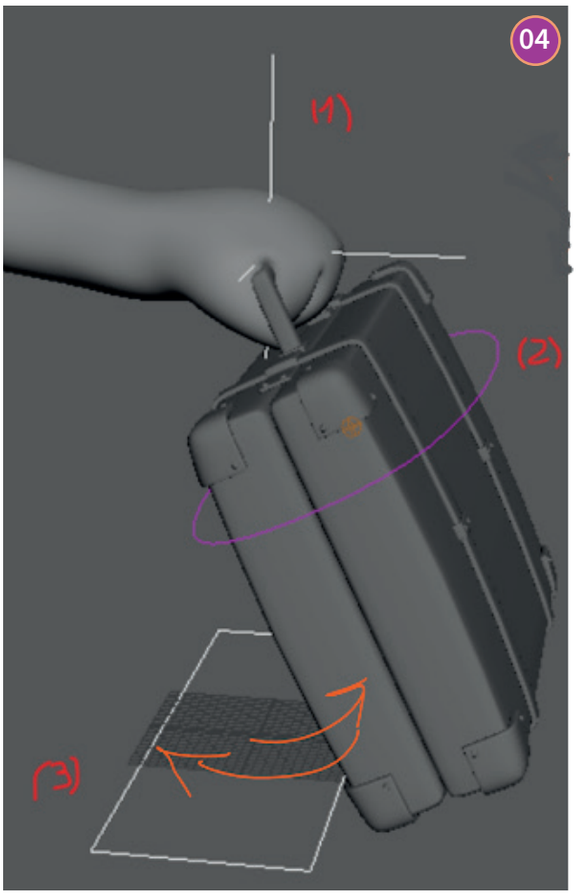
06 **The Intergalactic cocktail** Here, we create a main control (5) to move the entire cocktail; a liquid control (4) to simulate its oscillation; a control to place and bend the straw (3), two controls to move each fruit around (1, 2).
And that's pretty much it. Of course you could create more efficient and interesting rigs yet sometimes less is more (whether the Animator agrees).

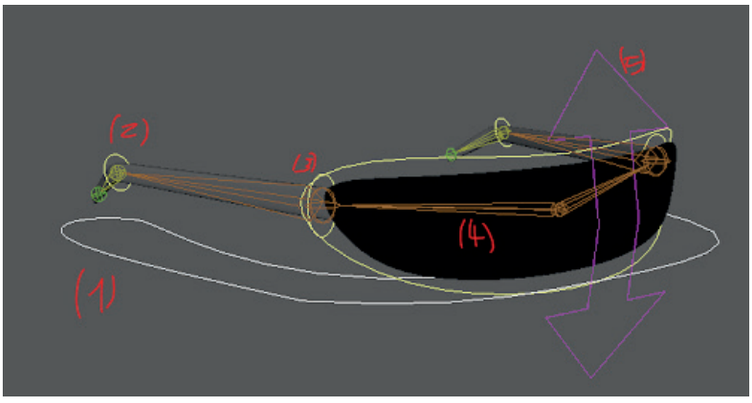


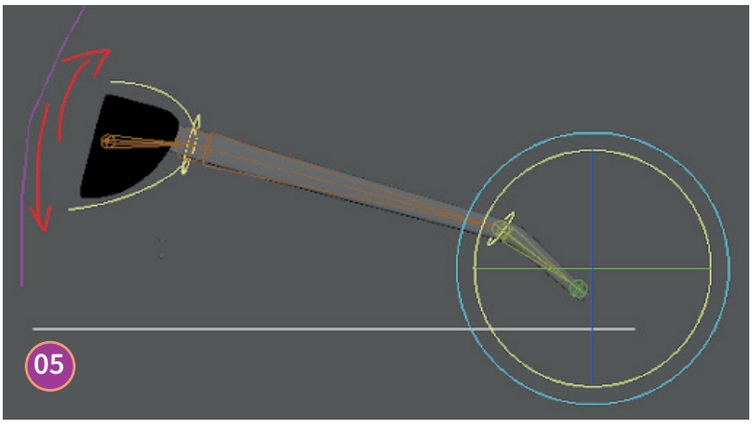


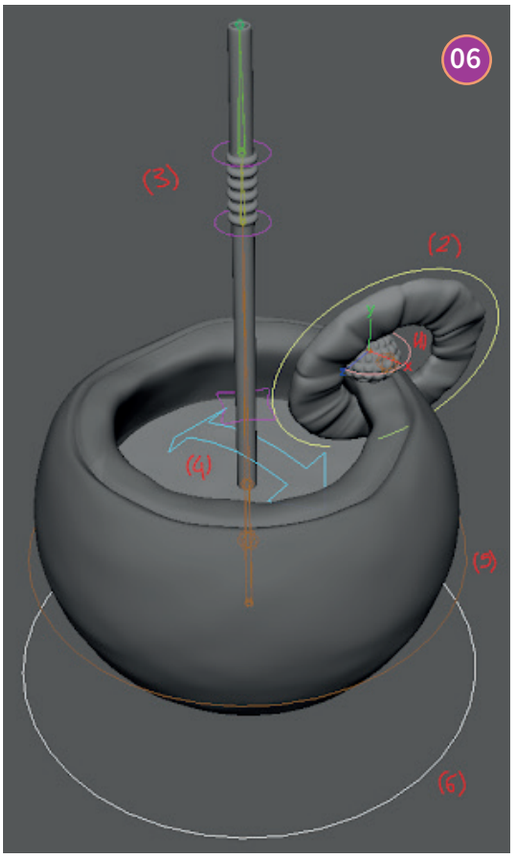

02




04
(1)
(2)
(3)


(2)
(3)
(4)
(5)
(1)


05


06
(3)
(2)
(4)
(5)
(6)

Deep dive

MEL and Python scripting

Maya 2022

"Here you can find a description of each script we used during the rigging process. You can run these scripts in a Python or MEL tab from the (Windows > General Editors >) Script Editor."

Python scripts

Version covered: Python 3

01 **Cmds command** Every Python script starts with the following line of code:

```
import maya.cmds as cmds
```

Which imports the main Maya scripting library for Python, maya.cmds.

02 **SelJnt** This script selects only joint children of the selected element. It will come in handy when skinning the whole character but remember to run it on the selected group, such as the skeleton group.

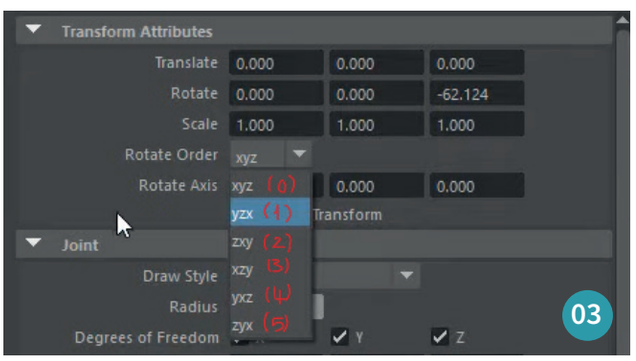
```
cmds.select(cmds.listRelatives
(type="joint", ad=1))
```

By delving into this small code, it will search for all descendants (ad=1) joints (type="joint") and store them in a list.

03 **AutoRO** We can use this script to easily update all rotate orders of the selected joints:

```
jointList = cmds.ls(type="joint")
for joint in :
    cmds.setAttr(joint +
".rotateOrder", 0)
```

Select joint > Attribute Editor > Rotate order. If we open the dropdown menu we see that there are six options and each of them is identified by an index (from 0 to 5). In depth, type="joint" tells Maya we are only looking for joints therefore jointList creates a list of all the joints in the scene. Then it applies the command cmds.setAttr(joint + ".rotateOrder", 0) to every joint in the list. At last, it sets the rotate order of the joint to the one whose index is 0, xyz.



04 NewControl

```
import maya.cmds as cmds

# select the joint you want to create the controls for
jointList = cmds.ls(sl=1, type="joint")
jointControlColorIndex = 19
size = 50

def createControl(joint):
    jointControl = cmds.circle(n=joint + "_ctrl", radius = size)[0]
    cmds.setAttr(jointControl + '.overrideEnabled', 1)
    cmds.setAttr(jointControl + '.overrideColor', jointControlColorIndex)
    return jointControl

def getRotateOrder(j):
    index = cmds.getAttr(j + '.rotateOrder')
    return index

def matchAllTransforms(w, matchTo):
    cmds.matchTransform(w, matchTo)
    return

def matchRotateOrder(w, index):
    cmds.setAttr(w + ".rotateOrder", index)
    return

def createOffsetGroup(control, joint):
    offsetGroup = cmds.group(control, n = control + "_offset")
    rotateOrderIndex = getRotateOrder(joint)
    matchRotateOrder(offsetGroup, rotateOrderIndex)
    matchAllTransforms(offsetGroup, joint)
    return offsetGroup, rotateOrderIndex

def cleanControlValues(control):
    cmds.makeIdentity(apply=True, t=1, r=1, s=1, n=0)
    cmds.delete(control, constructionHistory = True)
    return

for joint in jointList:
    jointControl = createControl(joint)
    cleanControlValues(jointControl)
    rotateOrderIndex = createOffsetGroup(jointControl, joint)[1]
    matchRotateOrder(jointControl, rotateOrderIndex)

cmds.warning("Note 1/2: Controls created using this script are GREEN. Select control > Attribute Editor > Display > Drawing Overrides > Color to change its color.")
cmds.warning("Note 2/2: Controls created using this script are HUGE. Select control > hold RMB > Control Vertex to change its size.")
```

MEL scripts

Maya Embedded Language

01 **FixThumbT** The following script is used to quickly match translate axes to orient axes of a joint.

```
joint -e -zso
```

-e stands for edit whereas -zso stands for zeroScale orient.

02 **CurveLen** Create a curve info node by running the following script on a selected curve.

```
arclen -ch 1;
```

03 **CombineCrv** You can use this script to combine curves and create a more complex control. Indeed we can't simply parent them.

```
parent -s -r;
```

It parents relatively the shape to another one.

04 **LockOffset** This script searches for _offset and locks the offset, space and grp groups off, without us having to do them all one at a time.

```
select -r "_offset";
select -r "_space";
select -r "_grp";
```

Sources

Rigging Large or Fat Characters - Volume Control Joints (Part 1), https://www.youtube.com/watch?v=dn9GPIZsMl0&list=PLzmRb2HRVMkM4eHme5pcqQqquHm6m9uN4&index=2&ab_channel=PointonPalley

#RiggingInMaya | Part 1 | Fundamentals | Joints, https://www.youtube.com/watch?v=urC_TBQQA7o&list=PLgala72Uap1qKmxnWBEneWOUifKrh8mk_&ab_channel=antCGi

#RiggingInMaya | Part 3 | Fundamentals | Control Creation & Setup, https://www.youtube.com/watch?v=gtm345xR2Ao&list=PLgala72Uap1qKmxnWBEneWOUifKrh8mk_&index=6&ab_channel=antCGi

Component Modes, https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-Basics/files/GUID-E3DBC099-2E93-47F3-B7FA-A219DB72B4A7-htm.html

8 Characteristics Of Successful User Interfaces, https://www.linkedin.com/pulse/8-characteristics-successful-user-interfaces-al-bashrawi-csm-cmmi

Qualities of Good User Interface Design, https://www.geeksforgeeks.org/qualities-of-good-user-interface-design/

Key Characteristics of Good UI Design – According to 8 Experts, https://www.uxpin.com/studio/blog/good-ui-design-characteristics/

Pole Vector constraints, https://help.autodesk.com/view/MAYAUL/2022/ENU/?guid=GUID-73C8C5B2-B0F8-4B96-9BB3-8AD257747E3D

Create IK handles, https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-A68E47F5-8F28-48C1-9B0F-370AA57ADDA8-htm.html

Lattices, https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-A2185BCC-FE7F-444D-9F47-8FB759CEF9ED-htm.html

Lattice options, https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-6EE9D875-5FEE-4631-81DE-772C2562B5A8-htm.html

Maya Facial Rigging 03: Skin Deformers, https://www.youtube.com/watch?v=-D_Q3J_UeYg&t=5s

#RiggingInMaya | Part 01 | Model Evaluation, https://www.youtube.com/watch?v=e74KphYwMww&list=PLgala72Uap1rtI7sy75fDHqV7VKQBMknt&index=2

Face Rigging in Maya - Part 1, https://www.youtube.com/watch?v=K9qT9kmRCAg

Interface overview, https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-Basics/files/GUID-F4FCE554-1FA5-447A-8835-63EB43D2690B-htm.html

Orient a joint's local axes manually, https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-FE43EB66-30EB-4148-9324-34390A4A0D3E-htm.html

Modelling with Animation in Mind, https://topologyguides.com/

Rigging Guideline for the Artist: What's Important for a Good Rig?, https://www.pluralsight.com/blog/film-games/rigging-guideline-artist-whats-important-good-rig

#RiggingInMaya | Part 01 | Model Evaluation, https://www.youtube.com/watch?v=e74KphYwMww&list=PLgala72Uap1rtI7sy75fDHqV7VKQBMknt&index=4w

Face Rigging in Maya - Part 1,
https://www.youtube.com/watch?v=K9qT9kmRCAg

Interface overview,
https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-Basics/files/GUID-F4FCE554-1FA5-447A-8835-63EB43D2690B-htm.html

Orient a joint's local axes manually,
https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-FE43EB66-30EB-4148-9324-34390A4A0D3E-htm.html

Modelling with Animation in Mind,
https://topologyguides.com/

Rigging Guideline for the Artist: What's Important for a Good Rig?,
https://www.pluralsight.com/blog/film-games/rigging-guideline-artist-whats-important-good-rig

Face Rigging in Maya - Part 1,
https://www.youtube.com/watch?v=K9qT9kmRCAg&t=502s

#RiggingInMaya | Part 12 | Intermediate | Face Rig - Joints & Controls,
https://www.youtube.com/watch?v=UpHKfUPyyBI&list=PLgala72Uap1rYD3m7Z6BpzNpdIQzgVtsM&index=8&ab_channel=antCGi

Rigging Large or Fat Characters - Blendshape Hacks (Part 3),
https://www.youtube.com/watch?v=KRnJRmVjrd8&list=PLzmRb2HRVMkM4eHme5pcqQqquHm6m9uN4&index=3&ab_channel=-PointonPalley

#RiggingInMaya | Part 5 | Basics | IK & FK | Limb Rigging,
https://youtu.be/uzHn_4ByyjY?t=970

Constraints,
https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-CharacterAnimation/files/GUID-7665A291-FAA7-44C0-BDEB-A6C83482116C-htm.html

#RiggingInMaya | Part 6 | Basics | Hand Rigging,
https://www.youtube.com/watch?v=3vJxXLw16Ak

#RiggingInMaya | Part 16 | Intermediate | Space Swapping/Switching,
https://www.youtube.com/watch?v=1JCUhEcxyRo&t=2139s

MAYA TUTORIAL- HOW TO MAKE RIG STRETCH & SQUASH AND EASY ANIMATION IN MAYA,
https://www.youtube.com/watch?v=_YKBfsBIXJo&ab_channel=MAYATUTORIALS

Maya Facial Rigging 05: Eye Controls,
https://www.youtube.com/watch?v=c6XPGvq1Vck&t=812s

My Full Workflow for Animating Character Props + Constraints,
https://www.youtube.com/watch?v=Fc0VbslxwOE

#CGTip | Advanced Twist Controls,
https://www.youtube.com/watch?v=DADqzaPigCE

#RiggingInMaya | Part 17 | Clavicles & Fingers,
https://www.youtube.com/watch?v=qYpSmKxOsOU