Character Rigging

endorphin lets you create your own character rigs using a powerful editing environment. A range of productivity tools such as mirroring and alignment make it easy to build character rigs that you can use with both kinematics and dynamics. A range of editors let you easily visualise and manage complex scenes composed of many characters. With *endorphin* Python and Lua scripting many recurring editing tasks can be automated. Characters can be animated or simulated directly in endorphin (using Nvidia's PhysX engine), or can be exported for use in *morpheme* or *euphoria*.

You can use *endorphin* to work with existing characters that you have created in other systems. Your custom *endorphin* build will include a plugin for the 3D authoring tool of your choice (such as Maya or 3ds Max), making it easy to exchange data with that systems using the open XMD file format. In addition, *endorphin* also natively supports other industry-standard animation and modelling formats such as FBX.

In this tutorial, you will learn how to create an *endorphin* physics rig from a character stored in an external Maya file and how to export it to the *euphoria* file format.

Contents

Before you begin

If you're starting with a Maya animation file, you will have to convert it to a format first that *endorphin* can import. The format used for this purpose is XMD and a corresponding Maya plugin should be provided with *endorphin*.

To install the plugin and export your character:

1. **Copy** the XMDMayaExportPlugin[*MayaVersion*].mll file provided with *endorphin* into your Maya \plug-ins folder and the MayaFileExportScript.mel into Maya's \scripts folder.

2. **Start Maya** and make sure the plugin is loaded (Window > Settings/Preferences > Plug-In Manager).

3. **Open** your animation file, choose **Export All** in the file menu and select XMD as the file format. **Export**. For the purpose of this tutorial we will work with the deer rig, so open and export the deer01_set.mb file.

4. Alternatively you can choose to modify the XMD export options and remove scene elements you don't want to export. For the purpose of this tutorial we only need the joint hierarchy, the mesh, and possibly the animation data. As the plug-in will export the whole scene, there will still be data in the output file that is not required for the purpose of rig-creation. We will be able to remove this later in *endorphin*.

If you're not starting from a Maya animation file, you can use the deer rig XMD file provided with this tutorial.

Overview of the *endorphin* interface

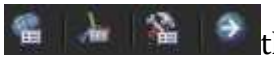
When you launch *endorphin* you will see a workspace containing at least four windows: the viewport, the scene explorer, the attribute editor and the timeline.

- **Viewports** provide a set of tools for selecting and manipulating scene objects, as well as for displaying scene data in visual form. Options for selectively displaying only certain types of objects in the scene as well as the type of shading can be found in the **Display** menu. The default viewport uses the Perspective camera. Other orthographic viewports can be added by selecting them from the **Window > Viewing** submenu. You can also switch the perspective of the current viewport using the 1-4 number keys.

- The mouse is used to control the camera view in a way similar to Maya:
  - **Alt + left mouse** (or middle mouse-button) to rotate the camera.
  - **Alt + middle mouse** (or middle + right mouse-button) to pan the camera
  - **Alt + right mouse** (or scroll-wheel) to zoom the camera.

- The **Scene Explorer** is a tree-view control useful for navigating scenes in a hierarchical manner. It supports context-menu operations such as rename,

remove, and expanding of child branches. Selecting an item in the explorer will simultaneously select and highlight it in the viewport and vice versa. The **Select All** command will select all objects of the same node type.

- The **Attribute Editor** is the main editor for working with object attributes in the scene hierarchy. Attributes are logically grouped and support highlight colours that indicate whether an attribute contains keyframed data (blue) or whether an attribute is connected to another attribute, receiving input data from the attribute (green). The context menu allows for control of keyframes (add, clear) and for copying values from one node to one or several others. The latter command, **Unify Values**, will set the selected attribute of all selected scene nodes to the value of the last selected node. At the top of the attribute editor you'll find a toolbar  that let's you switch between attributes of scene elements (**Active Selection**, capsule icon), properties of the currently selected tool (**Active Tool**, transform icon), and attributes of the currently selected command (**Active Command**, hammer and wrench icon). It also has a **Run** button that will execute the currently selected command.

- The **Timeline** is the main control for navigating over the scene time. The timeline contains the usual controls for starting and stopping playback as well as skipping forward and backward. It doesn't currently have controls for starting and stopping physics simulation. For those functions you will need to use the keyboard shortcuts, **Ctrl+Spacebar** and **Spacebar** respectively.

Several other types of windows, not initially visible in the workspace, will be used throughout this tutorial. **Group editors** (Window > Grouping) are the various editors used for creating and editing scene object groups.

- The **Display Layers** editor is an organisational tool to help manage complex scenes. Each entity in a scene may be associated with a display layer. Display layers have various associated properties, such as a colour and label, as well as settings such as visibility, selectability and editability.

- The **Selection Sets** editor is useful when you want to save and reuse specific object sets for selection.

- The **Collision Sets** editor is important for specifying non-colliding physics bodies in character definitions.

Creating and editing of groups works uniformly across the different types. The right-click context menu provides functions for the creation of a new group, adding of currently selected scene nodes, managing folder of groups, renaming and others.

There are other windows and tools in *endorphin* that are not being used in this tutorial. Since *endorphin* 3.0 is still in development, there is no guarantee that the use of any feature not documented here won't break your scene, and so shouldn't be considered safe. Generally it's recommended to save your file frequently and to keep copies of different stages in the authoring process. That way you can revert back to any previous step if a faulty action irreversibly breaks your data.

A handy list of important shortcuts for the functions used in this tutorial can be found at the end of this file.

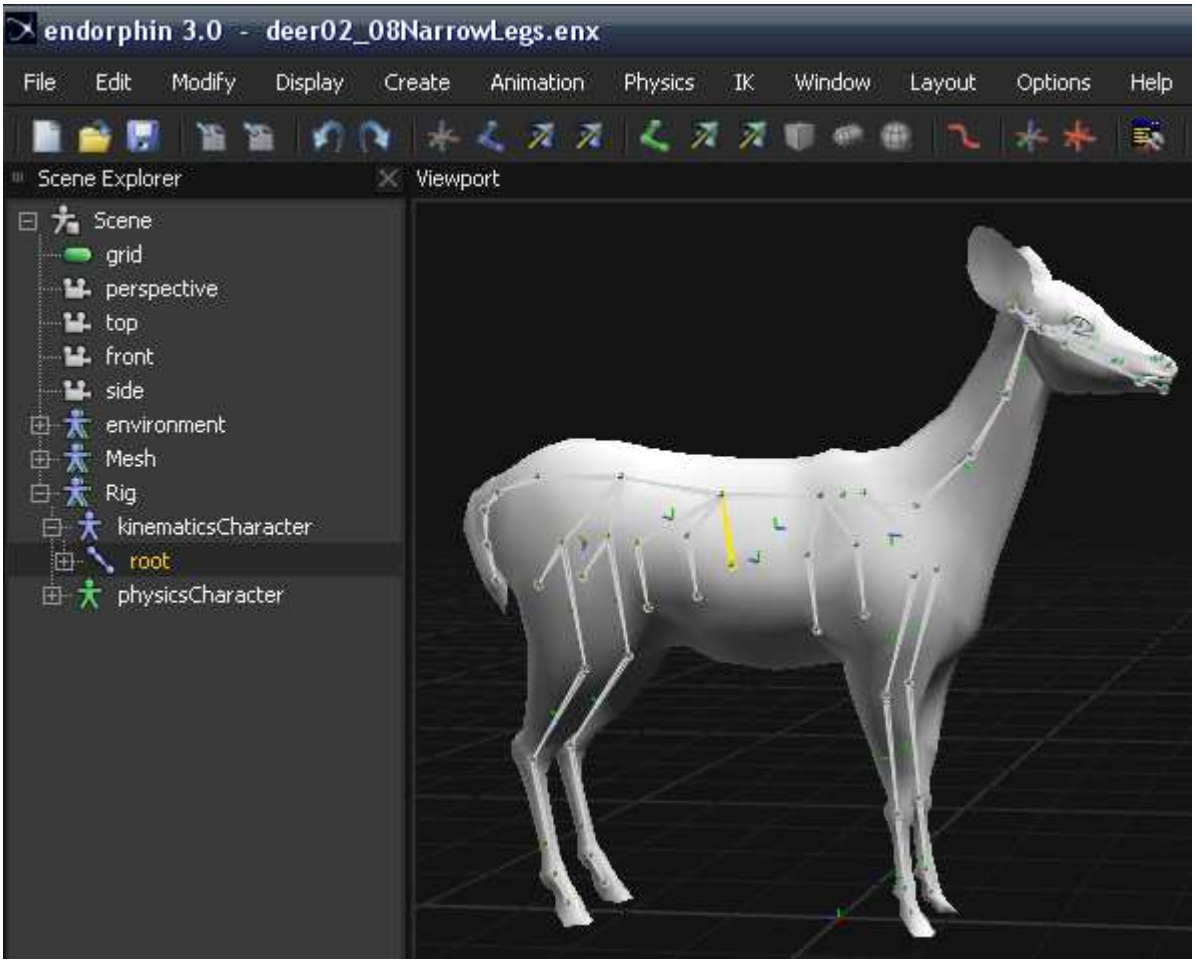
Step 1: Import the kinematics character

1. Launch *endorphin*. You'll see that the default scene is not empty, but already contains various objects, such as cameras (for use with various viewports), a grid and an environment character that contains a physical box object for the ground.

2. Select **File > Import...** or click the **Import** button on the main toolbar and then use the browser to open the file that contains your kinematics character. In this tutorial, we will assume you're using **deer01.xmd**. *endorphin* will now import all the nodes in the file that it can interpret. You'll notice in the scene explorer that endorphin automatically created **character** objects for each hierarchical structure found in the animation file. Each character node in turn holds both a **kinematics character** and a **physics character** object. This arrangement is due to the fact that a mapping needs to be maintained in endorphin between animation joints and physics joints. The viewport will also display the imported character, with kinematics skeletons rendered in purple and meshes in white.

3. At this stage you might want to delete objects that aren't necessary for creating the physical rig. These might include camera views that existed in Maya or any other spurious transforms that don't belong to the character you're interested in. The deer.xmd file contains e.g. a single "transform" object that can safely be deleted, as well as a character node that contains a single transform named **persp**. All we really need in this tutorial is the joint hierarchy and the mesh. Everything else, other than endorphin's default nodes, you can safely remove.

4. You might also want to rename some objects now. For example, **character1** holds mesh data, so we rename it **Mesh** by using the scene explorer context menu. Equally, **character2** holds the actual animation rig, so we rename it **Rig**. In larger scenes this will make it easier to navigate the explorer tree.



The imported animation rig and mesh.

Step 2: Organize your scene

Our rig will contain both kinematic and physical joints, joint limits and volumes. To allow for easier selection and editing of the different types it helps to organize them in different layers.

1. Bring up a layer window by selecting **Window > Grouping > Display Layers**, and dock it into a position of your choice.

2. Create a new display layer, by right-clicking on the first line of the window ("Layers") and selecting **New Display Layer** in the context menu. Rename it **Mesh** using the context menu.

3. To add the mesh to the corresponding display layer, first select the **meshShape** node, then right-click the display layer and select **Add Selected Items**. In general this will add all currently selected items. Selection itself can be done in both the viewport and the scene explorer. You can select multiple elements by using the Ctrl and Shift keys.

4. Create another layer for the kinematics rig and add the kinematics joints to it. Instead of manually multi-selecting all individual joints however, you might want to use either

the **Children > Select All** or **Select All** (by type) commands in the context menu of the kinematics character or its root joint.

Display layer window

5. You should be able now to toggle visibility, selectability and editability of the elements contained in a layer by ticking the **V**,**S** and **E** columns respectively. By double-clicking on an individual layer you can verify that it contains the desired elements: all contained nodes will be selected in the scene explorer as well as the viewport. You can further author the content of layers (removing items, clearing) by using the options in its context menu.

6. At this point you might want to use the display layer to disable visibility of the mesh. We will work with the joint hierarchy in the following steps, which will be easier if the mesh isn't blocking our view.

Step 3: Create the physics character

Although it would be possible to create a character from scratch by manually building up the joint hierarchy one-by-one and then matching joint positions to the kinematics rig, *endorphin* simplifies this step by providing a **Physicalize** command.

1. Select the kinematic root joint either in the scene explorer or in the viewport.

2. Go to **Modify > Physicalize** and click the box next to the command in the menu. The attribute editor will now switch to the options for the Physicalize command. Notice that this is the general mechanism for bringing up additional options for a given command. In contrast, clicking it directly will run the command with its default options. You'll see that the physicalize command has options for creating physics joint, limits and bodies. Although we could create all of these in a single step, if the joint hierarchy needs to be re-authored (by deletion and reparenting of joints), it will be easier to first create only the joints.

3. Ensure that only the **Physics Joints** option is turned on, leave the rest at defaults, and click the right-arrow button in the toolbar of the attribute editor to run the Physicalize command. *endorphin* now populates the physics character with joints that match the kinematics character joint hierarchy in name, position and orientation. This will be reflected both in the scene explorer as well as in the viewport by a green skeleton.

4. Display layers now come in handy as the viewport contains two skeletons occupying the exact same positions (kinematics purple, physics green). **Create a layer** for the physics joints so you can switch between the two more easily. You might also want to make the animation rig non-editable, so as to avoid accidentally modifying it.

5. You might also want to **change the appearance,** especially the size, of the joints as rendered in the viewport. The corresponding attributes can be found in the attribute editor's various display categories (see e.g Joint Display).

Step 4: Modify the joint hierarchy

Now is a good time to author the joint hierarchy. As we want to ensure matching kinematics- and physics rigs, we'll only consider here the deletion of joints we think are unnecessary.

Note: In its current version, *endorphin* doesn't automatically delete simulation data when a joint hierarchy is modified. This can lead to undesired behaviour, because most data is stored local to a node's parent. For example, if a joint in the middle of a chain is removed, local positions as output by the simulation are now local to a different parent node. This will make joints pop into new positions that are different from the kinematics character's positions. To avoid this, **it is important to always clear all simulation frames before modifying the joint hierarchy (Physics|Clear Simulation)**.

1. Removing an end-joint or a whole chain of joints: If no re-parenting is necessary, simply select a joint and hit the Delete key or select **Edit > Delete**. For example, for the deer you might want to remove the tail, the belly and the facial joints like this.

2. Deleting a joint in the middle of a chain by reparenting:

> 2.0 **Clear** all simulation frames (select **Physics > Clear Simulation**).
>
> 2.1 **Unparent** the child of the joint to be deleted by selecting it and choosing **Edit > Unparent**, or simply press **U**. So for removing neck01 in the deer, select neck02 and press **U**.
>
> 2.2 **Unparent** the joint to be deleted. In the deer example, select neck01 and press **U**.
>
> 2.3 **Remove** the joint to be deleted: select neck01 and hit delete.
>
> 2.4 **Re-parent** the child of the deleted physics joint to its new parent: first select the child (neck02) then shift+select (in the viewport) or ctrl+select (in the explorer) the parent (neck) and press **P** (or select **Edit > Parent**).

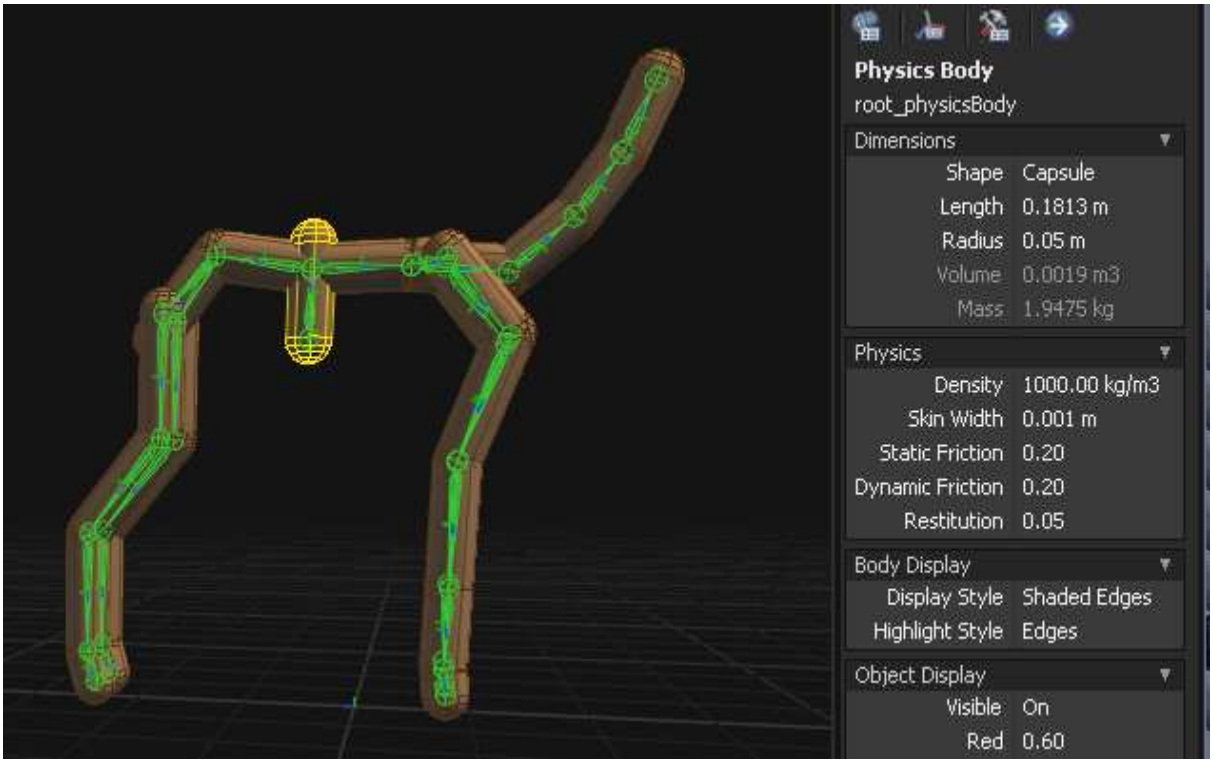
If the simulation frames were cleared correctly, the physics rig should now skip the joint we just deleted when compared to the animation rig. Repeat the steps above for all joints you deem unnecessary in your physical rig.

Step 5: Create physics bodies

endorphin won't be able to simulate your character unless the rig has physical bodies attached to its bones, so let's create some.

1. Select the kinematics root joint.

2. Bring up the options for the **Physicalize** command, turn Physics Joints and Physics Joint Limits off, and Physics Bodies on. Run the command. You should see that endorphin has created a new capsule object for each joint in the rig. By default the capsules will be oriented with their long axes along their respective bone. If a joint has several children, as is the case for the pelvis and hip joints e.g., the capsule will be aligned with the children's midpoint. If you select a body, you'll further notice in the attribute editor that each body is in fact associated with both the geometry that determines how the part collides with the environment, as well as the physical mass that determines the part's dynamics. The grayed out Volume and Mass attributes indicate that these values are calculated from other attributes. In the case of the capsule e.g. its volume is determined from the shape's length and radius, while mass is always the product of volume and density. The type of geometry (capsule, box, cylinder, sphere) can be changed through the Shape drop-down menu in the attribute editor.



Physics rig with auto-created shapes.

3. Although the volumes at this point won't reflect the desired end result, we're set to simulate for the first time now. Press **ctrl+space** to start simulating and **space** to stop. You'll immediately notice at least two things wrong with the simulated rig. Firstly, we haven't created joint limits yet, so joint rotations are completely unconstrained. Secondly, many parts are popping and jittering. The latter is due to the fact that by default all parts can collide with each other unless they're directly connected by a joint. Many parts will be auto-created however such that their volumes intersect others even if they're not directly connected. The simulation trying to resolve these initial interpenetrations is causing the observed jitter. We will remedy this in the next two steps.

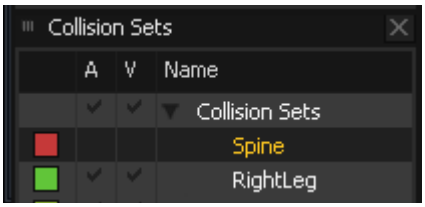
4. Before moving on, you might want to **create a new display layer** for the body parts you just created.

Step 6: Define Collision Sets

Before we author the properties of the rig's body parts, it's useful to create a few collision sets. This will a) allow us to ignore collisions that are due to interpenetrations in the initial pose, and b) let us optimize the performance of the simulation by ignoring collision that we deem irrelevant given the targeted fidelity.

1. Add a collision set window to your workspace: **Window > Grouping > Collision Sets**.

2. **Create a collision set** named "Spine" in the same way you previously created display layers.



Collision set window

3. **Select all joints** along the spine of the character whose associated shapes you don't want to collide with each other. In the deer e.g. choose the head and neck joints, the spine, pelvis and root joints as well as the hips and clavicles. In the viewport you might find it easier to select the bone pointing out from the joint towards its child, rather than the joint itself. With all desired joints multi-selected, add them to the Spine collision set.

4. **Simulate** again. You should see that the jitter and pop is now gone. Depending on your needs, you could go ahead now and create additional collision sets (e.g. for the limbs). This will reduce the maximum number of collision pairs produced during simulation.

In the remaining steps we will match the body parts to the character mesh and create joint limits to produce more believable motion.

Step 7: Adjust bodies to fit character mesh

In this step we will adjust the shape, position and orientation of the rig's physics bodies to more closely match a mesh or desired collision surface.

1. Make sure the mesh display layer is visible and change the opacity of the mesh in the attribute editor so you can see the physics bodies inside.

2. Select a shape you want to modify. You'll see that the attribute editor allows you to change all the necessary properties by entering numerical values directly. Keep in mind that positions and orientations are local to the shape's parent, in this case the joint it is

associated with. Instead of the attribute editor you can also change a shape's spatial attributes in the viewport using the transform tools:

- **Move Tool**: Press **W**. A manipulator will be displayed in the viewport that let's you move the selected item along any of the three spatial axes. Small transparent rectangles also allow you to freely translate the item in any of three planes. The planes are colour-coded (red, green, blue) to match the axis they're perpendicular to. Finally, a cyan rectangle in the centre of the manipulator allows you to translate the item in screen space.

- **Local/World** space transforms: Press **X** to toggle between transforms in local space of the selected item, or world space. The current mode will be reflected in the attribute editor's tool options (press the **Active Tool** button represented by a transform icon in the toolbar). Also, the viewport displays an orientation indicator in the bottom left corner that you can compare the manipulator orientation to.

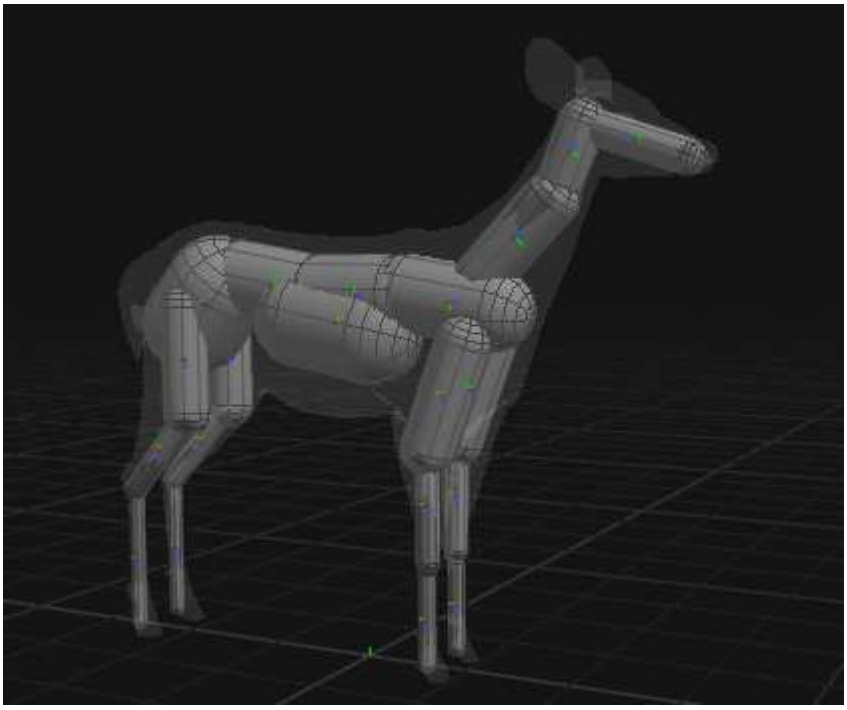
- **Rotate Tool**: Press **E** to bring up the rotate tool. For rotations around the primary axes use the red, green and blue colour-coded circular manipulators. For rotation around the camera's viewing direction use the cyan coloured circle. Again, pressing **X** toggles between local and worlds space rotations.

- **Scale Tool**: Press **R** to switch to the scale tool. The manipulator works analogously to the move tool, with the exception of the central cyan-coloured box, which will scale the element uniformly along all three dimensions.

- **Deselect** transform tool: Press **Q** to leave the transform tools.

  Note: You can multi-select several objects in order to equally transform them in their respective local spaces. For example, if you want to move two feet of a character such as to align them with the ground, you can select them both and use to move tool to translate them along their local vertical axes.

3. Move, rotate and scale all the rig's shapes to match your desired collision surfaces.



Example of body part matched to a mesh. The joint hierarchy was also simplified.

Step 8: Create and author joint limits

In this last step we will create joint limits to constrain the motion of joints to reasonable ranges. Joint limits are the most complex elements to author in a physics rig, so it's crucial to understand how they're represented in *endorphin*. You're therefore adviced to read the section on Working with Joints and Joint limits before proceeding.

Creating limits

endorphin supports both **Hinge** (1-dof) and **Ball Socket** (3-dof) limits. There are two ways for adding these to your joints:

- Option 1: use the **Physicalize** command. Select the root node of your physics rig and click the options box next to the physicalize command in the **Modify** menu. Change the options to create only joint limits and run the command. You'll see that *endorphin* creates ball-socket limits for all joints in your rig.

- Option 2: select any physics joint and press the **Hinge/Ball Socket button**  in the toolbar, or select **Create > Physics Joint Limit**. This will create a limit only for the selected joint.

endorphin creates limits with default ranges and orients them with their twist axes along the child bone of the joint. So after you've created a joint limit, you'll usually want to author its range and orientation to give you the desired range of motion. It's up to you whether you prefer to create and author all joint limits at once and incrementally shape their ranges, or whether you author them one at a time. Creating all limits at once will allow you to simulate the character with more plausible results right away. Often

it's difficult to aesthetically judge the effect of a single joint limit when its neighbouring joints are totally unconstrained. Let's look at the limits created more closely.

Limit Alignment

1. Deactivate visibility of everything other than joints and limits (you might want to create a display layer containing your joint limits for this purpose).

2. Select an arbitrary ball-socket joint limit and make sure no transform tool is active (press Q).

3. In the attribute editor change the value of **Swing 2 Angle** to be different from swing 1, say 15 degrees. This will make it easier to visually examine the orientation of the limit.

4. Now, to get a better feel for endorphin's limit setup, enable **Show Axes** in the Joint Limit Display category. Notice that the cone specifying the swing limits is always symmetric about the fixed frame's x-Axis (red), and that by default, this x-Axis is itself aligned with the bone.

5. In order to change this alignment, press the options box in **Modify > Align Joint Limit**. The attribute editor's tool options will now allow you to choose an axis and a direction for alignment. Other than along the bone, you can also choose to align to the midpoint of a joint's children (if there are more than one). Try a couple of different alignments, but make sure to eventually revert to the initial setup.

Fixed and moving frame

1. Select the joint (not the limit), activate the rotate tool, and turn its **Auto Rotate Joint Limit** option off.

2. Rotate the joint arbitrarily. You'll see that the fixed frame and hence the limiting cone remain fixed; this is because they're siblings of the joint. The moving frame in contrast, as represented by the white indicators, moves along with the joint. Though this rotation of the joint itself shouldn't be necessary when matching a physics- to a kinematic rig, it can be useful as a visual representation of what the skeleton looks like when a joint hits its limit, i.e. of the most extreme poses. Just make sure that you revert any changes you made to the orientation of the joint.

Limit orientation

1. Select the limit again and press **E** to activate the rotate tool.

2. In the tool's options turn **Rotate Joint Limit Offset** *off*.

3. Use the tool to arbitrarily orient the limit and observe how only the fixed (or limiting) cone moves, while the white indicators representing the moving (or limited) frame remain fixed. This is how you can specify limit orientations that are not symmetric with respect to the child bone.

4. In the rotate tool's options turn **Rotate Joint Limit Offset** *on*.

5. Rotate the limit around the x-Axis (red), which in this case should be the twist axis. This will allow you to specify the directions of the two swing axes. The twist limit will be unaffected, other than its wedge graphics being drawn in a different position.

6. Now try rotating the limit around any axis other than the twist. You should see that the twist axis comes out of alignment with the bone. This might or might not be something you want, but it's usually more intuitive if the twist axis is not oriented arbitrarily but aligned down the bone for instance.

Hinges

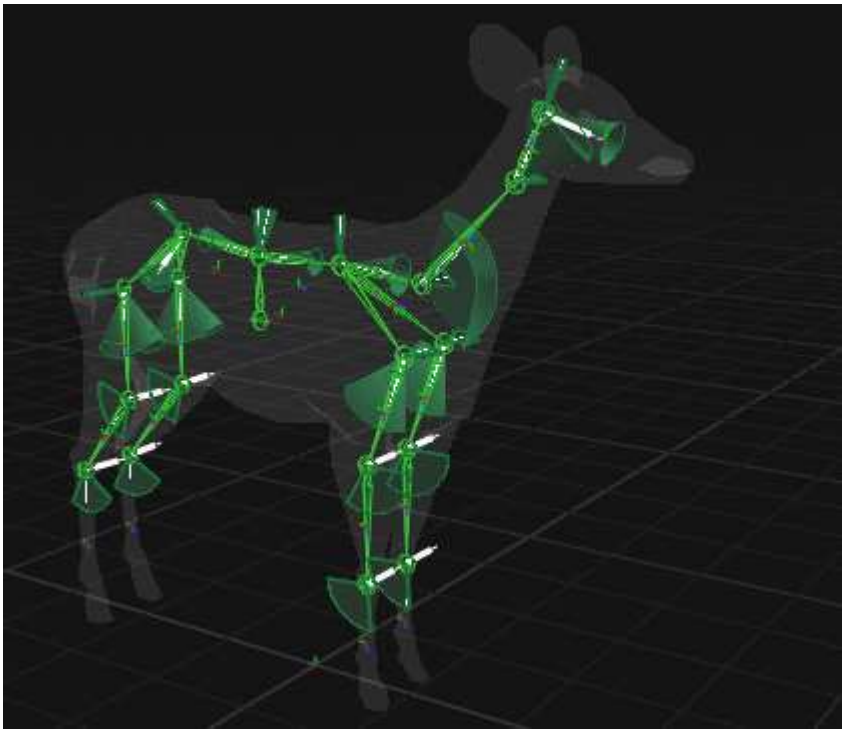
Hinge limits work exactly like the twist part of a ball-socket joint limit, the only difference being that the limit and moving frame can't be rotated independently. To create asymmetric hinge limits, like knees that can only bend in one direction, you use the **Twist Offset** value in the attribute editor instead. (Note: though you can explicitly specify a Moving Offset value in the attribute editor, this isn't guaranteed to work and hence should be avoided. If however you have changed those values accidentally, you can use the **Modify > Align Hinge Offset** command to align them again.)

Adjust all physics joint limits

You should now be able to use the joint limit tools in order to constrain the motion of your joints as you desire. Go ahead and adjust all the limits, changing the type from ball-socket to hinge where necessary. The mirror tool, explained in the appendix, can come in handy when trying to match joint limits on opposite sides of a character, so it's good to get familiar with.

Simulate at any point to see the effect of your adjustments. It can also be helpful to change the joint's **Spring** and **Damping** values in the attribute editor to see the rig simulating with different dynamics. Be aware though that these parameters might be different from the ones existing in other physics engines and hence won't have the same effect, and in most cases won't even be exported.

If you're done, you can compare your rig to the final version of the deer provided with *endorhin*.



Joint limits of the finished deer rig.

Step 9: Running scripts

Endorphin supports scripting of its scenegraph and GUI using both Lua and Python. It's not in the scope of this tutorial to completely document the necessary API, so we'll just mention a particular script that will be helpful in the context of character rigging.

You'll probably have noticed that the only way to change the weight of a character in endorphin is to modify the density of individual body parts. Sometimes however it would be desirable if you could scale the overall weight directly. This is a perfect candidate for a script:

1. Select the **Scene** node in the explorer.

2. Make sure that **Python** is selected as the scripting language in the attribute editor.

3. Open both a script window (**Window > Scripting > Application Script**) and a log to see the script's output (**Window > Utility > Log**).

4. In the script window, open the printMasses.txt file. Inspecting the code you'll see that it simply iterates over all physics bodies in the scene, printing out individual part masses as well as total weight. It's not a very smart script. E.g. it doesn't discriminate between the character rig and the environment. It assumes that the first physics body it finds in the scene is the ground object and will skip this in the loop.

5. **Run** the script by pressing the arrow button. You should see the output of the script in the log window, in this case the masses of the character. To clear the log, give focus to its pane and press the button next to the run command (hand icon).

6. Next, open scaleMasses.txt. Change the desired mass for your character in the first line of the script and run it. The script will uniformly scale all part densities so as to achieve the desired total weight. If you want to change the relative weight of some parts, you can change their density attribute directly in the attribute editor and then run the script again. So if e.g. you want to make the belly lighter without changing the overall weight, reduce the belly's density manually first, then run the script. Although this will also alter the belly's mass, the difference between its old and new mass will be distributed across all other body parts.

Finishing up

When you're done with the previous steps, export your character by first selecting it in the scene explorer, then choose **File > Export**. If the character is not selected, the whole scene will be exported, including nodes you won't need, like the environment or camera perspectives.

You should at this point have created your own physics character based on an existing animation file. If you have any problems you can open the endorphin file containing the finished deer rig used throughout this tutorial for comparison.

You don't, of course, have to always follow the exact order of steps as used in this tutorial when creating a new rig. You could also e.g. create all joints, shapes, and limits at once using the Physicalize command with all options enabled, and only later decide to remove certain joints. Equally you don't have to create the same arrangement of display layers. Just change the workflow to whatever suits your needs best.

Remember to keep backup copies of your file at various stages of the rigging process though. Your copy of *endorphin* is not a finished product and as such is not guaranteed to work flawlessly all the time. If some action inadvertently breaks your rig, you'll want be able to go back to the last working version of your character.

Appendix: Keyboard shortcuts

- **ctrl + N** : create a new scene.
- **ctrl + O** : open a scene.
- **ctrl + S** : save the scene.
- **ctrl + Z** : undo. Don't use multiple undo at the same time, it could crash Endorphin
- **ctrl + Y** : redo.
- **ctrl + M** : mirror tool.
- **ctrl + G** : run command.
- **ctrl + space** : simulate the scene.
- **space** : play/stop the scene.
- **ctrl + shift + left** : go to first frame of the scene.
- **ctrl + left** : go to previous frame of the scene.
- **ctrl + shift + right** : go to last frame of the scene.
- **ctrl + right** : go to next frame of the scene.
- **X** : toggle between global and local coordinates Endorphin 3.0.

- **W** : translation tool.
- **E** : rotation tool.
- **R** : resize tool.
- **Q** : leave current used tool.
- **P** : parent the first selected joint to the second selected joint as a child to his father.
- **U** : unparent the selected joint to his father.
- **1**,**2**,**3**,**4** : respectively the perspective, front, side, top camera view.

Appendix: Mirror tool

The Mirror command (Edit > Mirror, Ctrl+M) lets you quickly create a duplicate object hierarchy that is the mirror of a selected object hierarchy. All transform-type objects (such as transforms, joints, joint limits, physics joints, physics joint limits, cameras and lights) are mirrored, as well as shape-type objects (such as physics bodies).

1. Select an object to act as the mirror root

2. Bring up the options for the mirror tool in the Edit > Mirror menu.

3. In the Mirror Name Replacement category, change the strings to the appropriate identifiers for your rig. If e.g. you use a naming convention for your joints such that on the left half of the character joints are named jointX_l and on the right jointY_r, then enter the strings "_l" and "_r". This will allow *endorphin* to cleverly find the joints to be removed or replaced by the mirror command, or to name the newly created parts appropriately. Leave the other options at their defaults. (Particularly, the Joint Limit Mode option is not yet fully functional, so leave it at "None".)

4. Run the command. If you selected a limit or a shape, only that single element will be mirrored. If on the other hand you selected a joint, its whole subtree will be mirrored. This can be used e.g. to mirror a whole leg hierarchy, including joints, shapes and limits.

Appendix: Working with joints and joint limits

endorphin scenes are composed of a hierarchy of **transform** objects. A transform is essentially a reference frame that specifies a **local position** and a **local orientation** in terms of the reference frame of its parent transform. Iin addition, transforms also specify a scale, but for most character rigging purposes scaling will always be set to 1.0.

- The **local position** is described by six attributes: PreTranslationX, PreTranslationY, PreTranslationZ, TranslationX, TranslationY,TranslationX. The pre-translation offset is applied, followed by the translation offset. Typically, any translation animation is applied to the translation, rather than the pre-translation.
- The **local orientation** is described by nine attributes: PreRotationX, PreRotationY, PreRotationZ, RotationX, RotationY, RotationZ, PostRotationX, PostRotationY, PostRotationZ. Rotation attributes are Euler angles (usually in degrees), where the pre-rotation is applied first, followed by the rotation,

followed by the post-rotation. Typically, any rotation animation is applied to the rotation, rather than the pre-rotation or post-rotation.

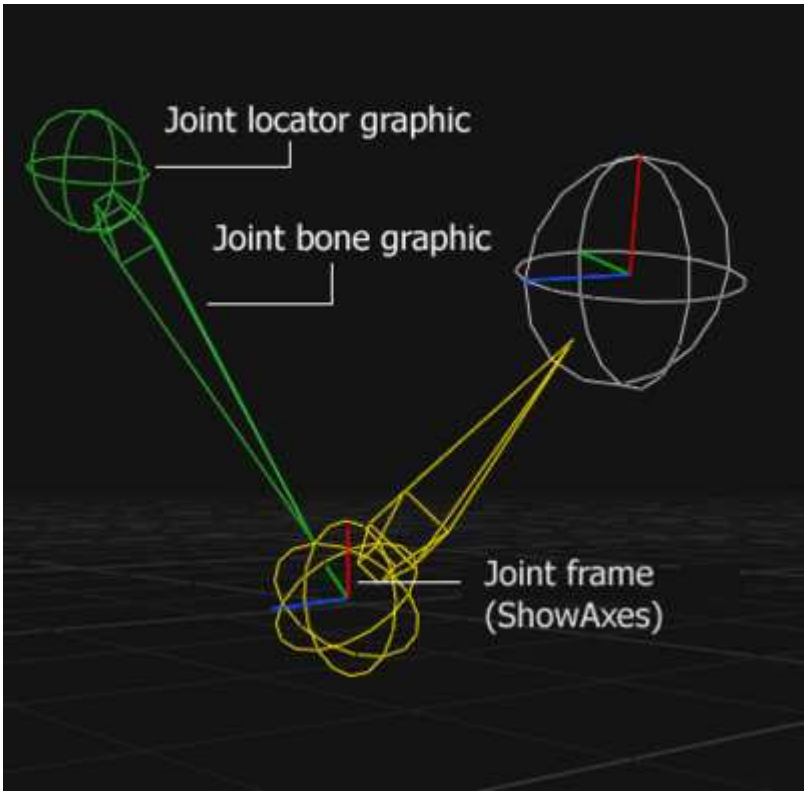
Many types of scene object are derived from transform objects. This means that they are actually transform objects that also contain additional attributes. For example, joints, joint limits, physics joints, physics joint limits and characters are all types of transform object. You can use manipulators (such as the Move, Rotate and Scale tools) with all transform objects--and objects derived from transform objects--to modify the TranslationX, TranslationY, TranslationZ, RotationX, RotationY, RotationZ, ScaleX, ScaleY and ScaleZ attributes. Transform objects are displayed via a small axes graphic, which indicates the position and orientation of the transform in terms of the scene.

Joint reference frames

Joints and physics joints are derived from transform objects, and include additional attributes that specify the display of the joint bone graphic (such as BoneRadius), as well as physical parameters (such as SwingDamping), in the case of physics joints. However, in most other respects they can be thought of as simply transforms.

It can be useful to visualise the local transform axes for a joint or physics joint. This visualisation is controlled by the **AxesStyle** attribute. When this attribute is set to **ShowSelected**, the joint transform axes are visualised when the joint, or any of its parent joints, is selected. When this attribute is set to **Show**, the joint transform axes are always shown. When this attribute is set to **Hide**, the joint transform axes are never shown.

It is important to remember that joint bones are not necessarily aligned to any particular joint axis. When new joints are created, the default translation of each child joint is along the local x-axis of the parent joint, but this is simply a useful default.



Joint limit reference frames

Joint limits and physics joint limits are derived from transform objects, and include additional attributes that specify the display of the joint limit graphic (such as DisplayScale), as well as physical parameters (such as UpperSwingLimit), in the case of physics joint limits. However, in other respects the can be thought of as simply local transforms.

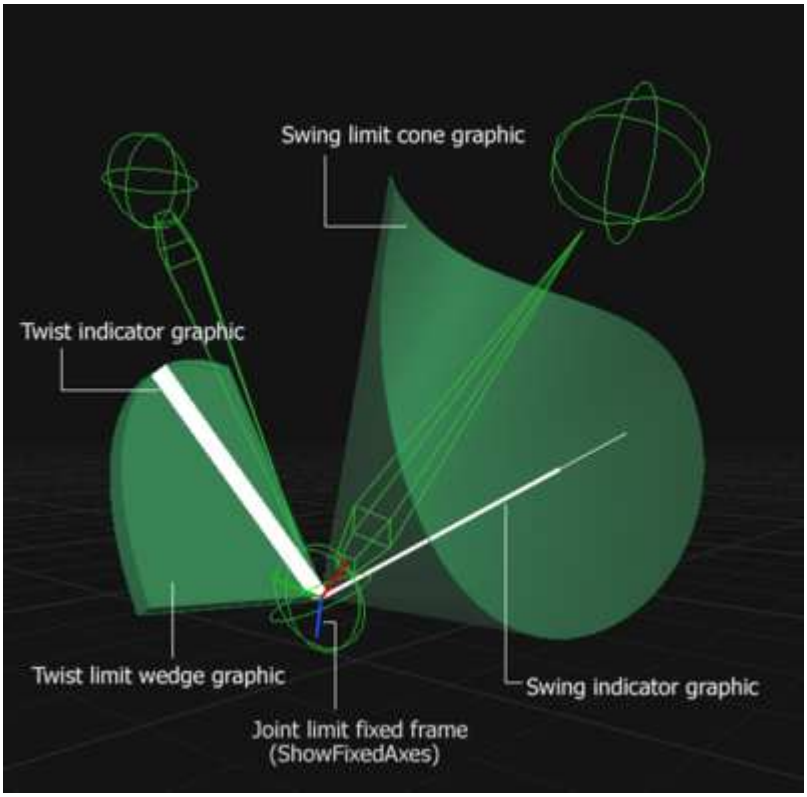
The role of a joint limit is to constrain the range of motion (both swing and twist) of a specific joint. A common terminology is to refer to the reference frame of the joint limit as the **fixed frame** and the reference frame of the joint being limited as the **moving frame**. In addition, a fixed offset between this child joint frame and the moving frame is supported---this is referred to as the **moving frame offset**. Also note that joint limits are used to constrain joints, whereas physics joint limits are used to constrain physics joints.

In terms of the transform hierarchy, a key concept is that a joint limit is a **sibling of the joint that it limits**. That is, the joint limit and the joint that it limits both share the same parent joint. Both the joint and its joint limit have their own local transforms and can be rotated independently of each other. However, note that joint limits cannot be translated, as their local translation is automatically constrained to be the same as the local translation of the joint that it limits. This is to ensure the joint and its joint limit appear at the same world position, and is why the various translation attributes for joint limits are not shown in the Attribute Editor.

A joint limit is composed of a **swing limit cone** and a **twist limit wedge**. The role of the swing limit cone is to limit the swing motion of moving frame (as indicated by the swing

indicator), whereas the role of the twist limit wedge is to limit the twist motion of the moving frame (as indicated by the twist indicator). Both the swing indicator and twist indicator are associated with the moving frame (that is, the joint being limited), and rotate as this joint rotates.

Usually, the fixed reference frame (that is, the reference frame of the joint limit) is not displayed. However, you can display this axis by turning on the **ShowFixedAxes** attribute of the joint limit.



Swing limit cone

The swing limit cone is used to limit the swing of the joint being limited, where swing is defined as rotations about the local x-axis and z-axis of this joint. Swing is indicated by the white **swing indicator**. The swing limit cone is defined symmetrically about the local x-axis of the joint limit frame, and does not depend on the orientation of the joint being limited. There are separate attributes, Swing1Angle and Swing2Angle, that define the range in the local xz-plane and xy-planes respectively. The swing limit cone graphic depends only on the orientation of the joint limit frame, and does not depend at all on the orientation of the frame of the joint being limited. Note that the swing angles are *half-angles*. For example, when a given swing angle is 90 degrees, this actually means that the joint is able to swing from +90 degrees to -90 degrees (and so has a neffective range which is double the specified swing angle).

Twist limit wedge

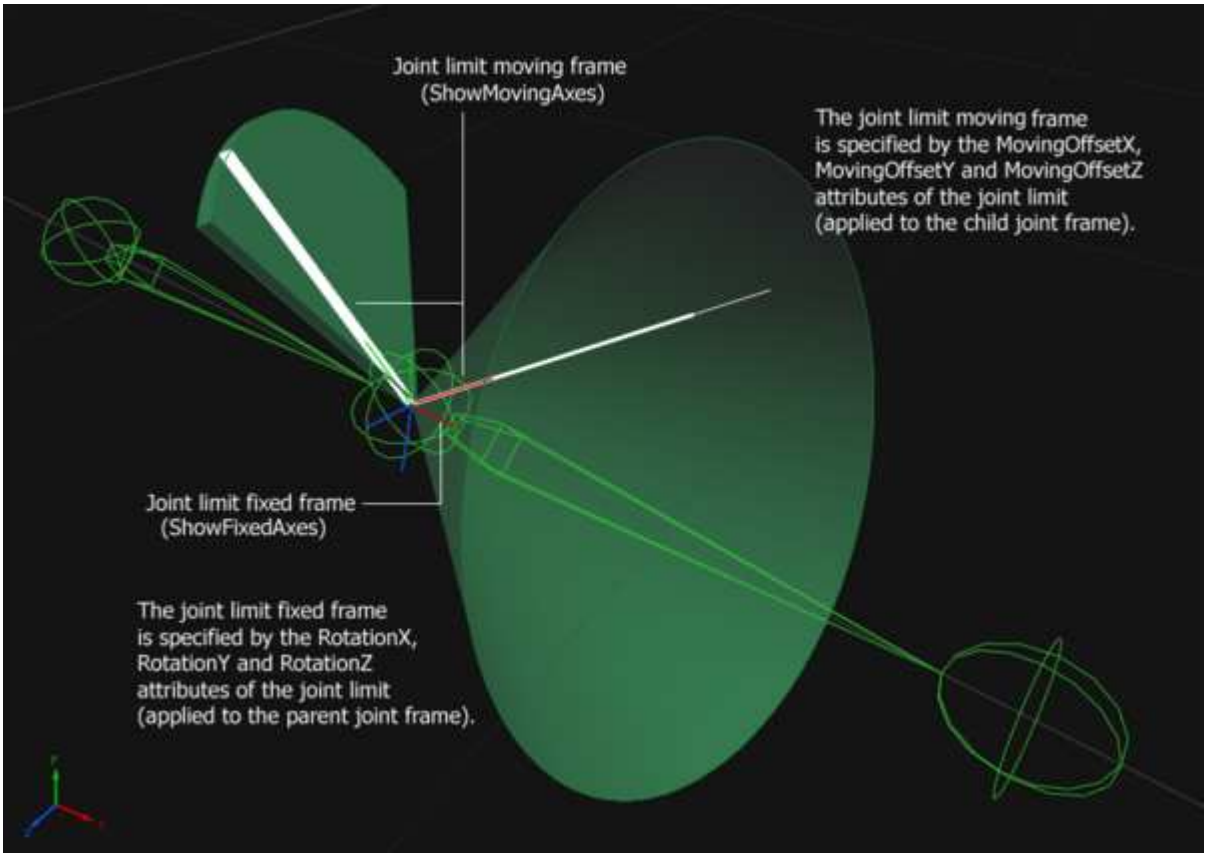
The twist limit wedge is always located in the local yz-plane of the joint being limited. This is because the role of the twist limit wedge is to limit the twist of the joint being

limited, where twist is defined as rotation about the local x-axis of this joint. Twist is indicated by the white **twist indicator**. However, the actual orientation of the twist limit wedge display graphic within this plane depends upon the twist of the joint limit frame, and so in practise the twist limit wedge appears to move if either the joint or joint limit are rotated. However, the key thing to note is that the twist limit wedge is always drawn perpendicular to the x-axis of the joint. The twist limit wedge is defined symmetrically about the zero twist direction, with a range specified by the TwistAngle attribute, and an optional offset (in the joint's yz-plane) specified by the TwistOffset attribute. Note that the twist angle is a full angle, rather than a half-angle. For example, when a given twist angle is 90 degrees, this actually means that the joint is able to twist from +45 degrees to -45 degrees. The addition of a twist offset changes this range. For example, if a twist offset of 20 degrees is applied, the joint is able to twist from +65 degrees to -25 degrees.

Joint limit moving frame

The swing and twist indicators are associated with the moving frame (that is, the frame of the joint being limited). By default, these indicators point down the local x-axis and local y-axis of the limited joint, respectively, and so rotate (with respect to the joint limit frame) as the downstream joint rotates. These indicators specify the joint limit's **moving frame**, which is by default the same as the child joint frame itself. However, you can choose to **offset** the moving frame by specifying non-zero **MovingOffsetX**, **MovingOffsetY** and **MovingOffsetZ** joint limit attributes. Although these attributes are stored as part of the joint limit definition, they are applied to the downstream joint frame in order to obtain an effective downstream joint frame (which is the joint frame with the additional rotation offset applied).

When a non-zero moving frame offset is specified, the moving frame axes will no longer be aligned with the joint axes. Usually, the moving frame axes are not displayed. However, you can choose to display these by turning on the **ShowMovingAxes** setting.



Additional rotation modes

Usually, you will use the Rotate tool to rotate joint frames or joint limit frames. Typically, joints and joint limits are treated like any other type of transform object, and the Rotate tool will simply update their RotationX, RotationY and RotationZ attributes. However, there are two additional modes worth keeping in mind:

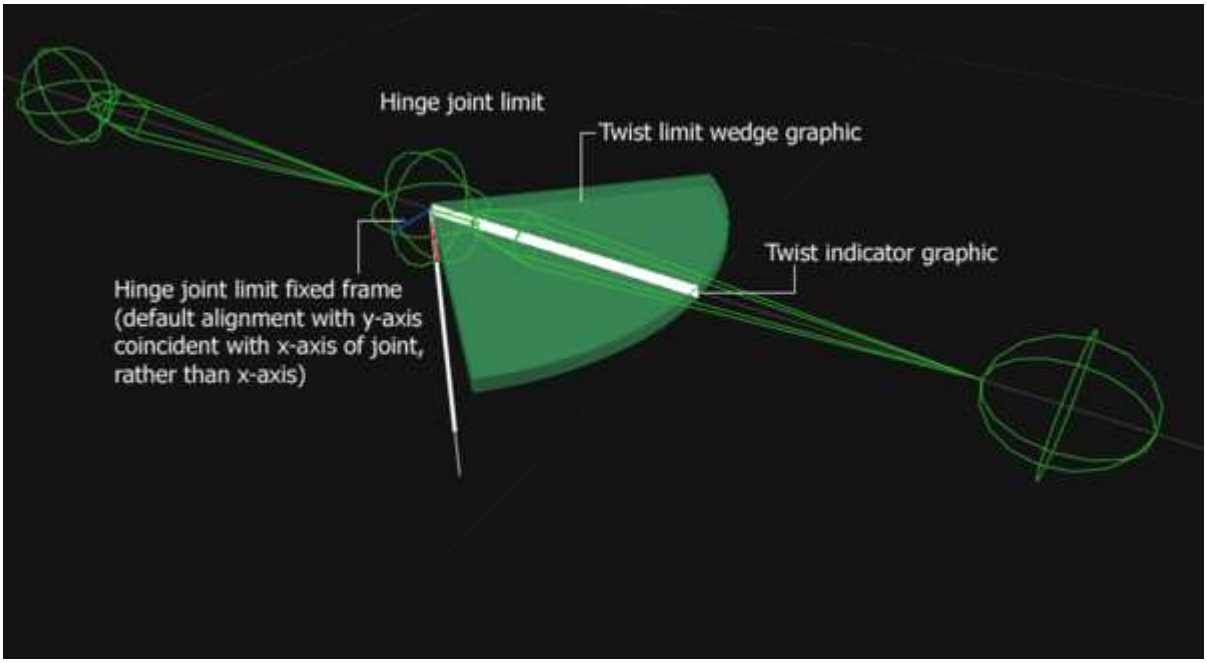
- **Auto Rotate Joint Limits:** Usually, when you rotate a joint, you do *not* want its joint limit to rotate (since you're typically editing the orientation of the joint with respect to its joint limit). Sometimes, though, it is useful to be able to edit a character's joints by rotating them and have their joint limits automatically rotate as well, without the need to explicitly select them. You can do this by activating the Rotate tool and selecting a joint or physics joint, then selecting the Active Tool button in the Attribute Editor, then turning on the AutoRotateJointLimits attribute.
- **Rotate Moving Frame:** Usually, when you rotate a joint limit, jou do *not* want its moving offset frame to rotate (since you're typically editing the orientation of the joint limit fixed frame with respect to its moving frame). Sometimes, though, it is useful to be able to edit a joint limit's moving frame offset attributes (which modify the effective joint frame by adding this rotation offset to the joint frame) using the Rotate tool. You can do this by activating the Rotate tool and selecting a joint limit or physics joint limit, then selecting the Active Tool button in the Attribute Editor, then turning on the RotateMovingFrame attribute. Note that in the case of hinge joint limits, this option is always used, regardless of the actual setting in the Attribute Editor, since in the case of hinges the fixed and moving frames are swing-aligned (although their relative twist will vary).

Hinge joint limits

There are two types of joint limit: **ball-socket** joint limits and **hinge** joint limits. A ball-socket joint limit allows some swing and some twist. In constrast, a hinge joint limit only allows twist, and prevents all swing. You can change the type of joint limit by setting the LimitType attribute. You will typically use hinge joint limits for certain kinds of character joints such as elbows and knees. Note that this attribute is a setting of the joint limit, and is **not** a setting of the joint. That is, joints themselves are not categorised as ball-socket joints or hinge joints--this distinction is only made at the joint limit level.

There are two main distinctions between a ball-socket joint limit and a hinge joint limit:

- Hinge joint limits do not allow swing, so they do not have a swing cone displayed, and all swing-related attributes are suppressed in the Attribute Editor.
- Hinge joint limits only allow twist, which means their moving and fixed frames are coincident (except for any twist). This means that in general, when rotating a joint limit via the Rotate tool, the moving frame is automatically rotated as well--effectively, the RotateMovingFrame setting is always on in the case of hinge joint limits.



Summary of joint limit reference frames

- **Parent joint frame:** The reference frame of the parent joint that is the parent of both the joint and joint limit.
- **Child joint frame:** The reference frame of the joint that is being limited.
- **Joint limit fixed frame:** The reference frame of the joint limit.
- **Joint limit moving frame:** The reference frame of that is being limited, which is composed of the child joint frame plus any additional offset rotation as specified by the MovingOffsetX, MovingOffsetY, MovingOffsetZ attributes of the joint limit.